

Protein Conformation, Ligand Similarity, and Activity: A Multi-Metric Analysis of SAR Landscapes

By

Raina Rempel

A Thesis submitted to the Faculty of Graduate and Postdoctoral Studies
at The University of Manitoba

in partial fulfillment of the degree requirements of

MASTER OF SCIENCE

Department of Chemistry

University of Manitoba

Winnipeg

Copyright © 2026 by Raina Rempel

Abstract

Computationally derived structure–activity relationship (SAR) metrics are increasingly incorporated into medicinal chemistry workflows to prioritize compounds during early-stage drug discovery. However, the effectiveness of these approaches depends on how accurately computational similarity measures capture the underlying SAR landscape of a target protein. This work uses cyclin-dependent kinase 2 (CDK2) and its associated crystallographic ligand–protein complexes as a model system to evaluate the ability of commonly employed virtual screening metrics, including: ligand similarity, protein–ligand interaction similarity, and molecular docking, to reproduce both SAR continuity and SAR discontinuity. The effects of protein conformational variability and ligand flexibility are also investigated to assess the importance of accounting for receptor dynamics during virtual screening. The SAR discontinuity that is observed within the CDK2 ligand library is also explored within this study, with the mechanism of activity cliff formation between several ligand pairs evaluated with the goal of characterizing their presence in a SAR landscape. This work establishes a framework for evaluating the ability of virtual screening methodologies to describe SAR landscapes and provides practical guidance for selecting computational metrics that more accurately reflect experimentally observed structure–activity relationships during early-stage lead discovery.

Acknowledgements

First, I would like to acknowledge and thank my advisor, Dr. Rebecca Davis, for the time and effort she has put into assisting me with this work. I deeply appreciate your advice and support over the years. I would like to additionally acknowledge and thank the entire Davis Research Group for their support and assistance with this work: Kaitlin Isfeld, Kirsten Hurdal, Hunter Sturm, and Andrea Minutti-Simoni.

I would like to acknowledge and thank my excellent committee for their guidance and support throughout my masters degree: Dr. Zev Ripstein and Dr. Katie Mitchell-Koch. I appreciate your feedback and assistance in my endeavours.

I would like to thank the Faculty of Graduate Studies and the Department of Chemistry for their financial support throughout the duration of my study.

Finally, I would like to thank my family for their continued love and support, during the duration of this study and beyond: Steven, Bronwyn, Kadri, and, of course, Gatsby. You mean more to me than words can describe.

Table of Contents

Chapter 1 – Introduction.....	6
1.1 Introduction to Structure-Activity Relationships and Discontinuity.....	6
1.2 Protein of Interest – Cyclin-Dependent Kinase 2.....	9
1.3 Purposes of this Work.....	13
Chapter 2 – Methods.....	19
2.1 Curation and Preparation of Datasets.....	19
2.1.1 Curation and Preparation of Representative Co-crystallized CDK2 Dataset.....	19
2.1.2 Active Site RMSD-based Clustering of Crystal-Posed CDK2 Dataset.....	23
2.1.3 Curation and Preparation of Representative ChEMBL Sample Sets.....	27
2.2 Ligand Similarity Metric Computations and Molecular Docking Computations.....	30
2.2.1 Ligand Similarity Metric Definitions.....	30
2.2.2 Ligand Similarity Metric Computations.....	34
2.2.3 Autodock Vina and GNINA Molecular Docking Methodology.....	36
2.2.4 CDK2 Crystal Posed Dataset Molecular Docking Computation.....	39
2.3 Statistical Methodologies.....	41
2.3.1 The Pearson r coefficient.....	41
2.3.2 The p-value test (with respect to Pearson r).....	42
2.3.3 Kernel density estimation (KDE).....	43
2.3.4 The Fisher’s exact test.....	43
2.3.5 The Fisher’s Z test.....	44
2.3.6 The Stieger’s Z test.....	45
Chapter 3 – Similarity Metrics.....	54
3.1 Analysis of Ligand-Based Similarity Metrics on a Crystal-Posed Dataset.....	54
3.1.1 Similarity Metric Tanimoto Comparison for Crystal-Posed Ligand Dataset.....	55
3.1.2 Cutoff Similarity Metric Comparison for Crystal-Posed Ligand Dataset.....	60
3.2 Analysis of Ligand-Based Similarity Metrics on a ChEMBL Dataset.....	63
3.2.1 Similarity Metric Tanimoto Comparison for ChEMBL Ligand Database.....	63
3.3 Scoring-Function Analysis of Ligand Binding on a Crystal-Posed Dataset.....	67
3.3.1 Molecular Docking Score and Activity Comparison for RCSB Ligands.....	68
3.3.2 Δ Molecular Docking Score and Δ Activity Comparison for RCSB Ligands.....	73
3.4 Discussion and Conclusions.....	76
Chapter 4 – Ligand Cross-Docking and Scoring Function Evaluation.....	84
4.1 Scoring Function Analysis on a Cross-Docked CDK2 Dataset.....	84
4.1.1 Mean-Ligand Cross-Docking Scoring Function Analysis on CDK2 Dataset.....	85
4.1.2 Effects of Active Site Conformation on Mean-Ligand CDK2 Cross-Docking.....	90
4.1.3 Cross-Docking Analysis of Scoring Functions for Ligand Ranking Accuracy.....	96
3.4 Discussion and Conclusions.....	98

Chapter 5 – Identification and Structural Analysis of Activity Cliffs.....	103
5.1 Similarity- and Conformational-based Definition of Activity Cliffs.....	103
5.1.1 CDK2 Similarity Metric-based Classification of Activity Cliff Thresholds.....	105
5.1.2 Active-Site Conformation-based Analysis of Activity Cliff Distribution.....	107
5.2 Examination of Scoring Function’s Activity Cliff Prediction Capabilities.....	112
5.2.1 Analysis of Vina and GNINA’s Prediction of Activity Cliffs.....	112
5.3 Structural Analysis of α C-helix-out-A Activity Cliff Pairs.....	115
5.3.1 Structural Analysis of CDK2 Ligands using KLIFS Interaction Fingerprints.....	115
5.3.2 Structural Analysis of CDK2 α C-out-A Ligand Activity Cliff Pairs in PyMOL...	118
5.3.3 Conclusions about CDK2 α C-out-A Activity Cliff Pairs.....	126
5.3 Discussion and Conclusions.....	127
Chapter 6 – Conclusions and Future Work.....	133
6.1 Conclusions.....	133
6.2 Future Work.....	136

Chapter 1 – Introduction

1.1 Introduction to Structure-Activity Relationships and Discontinuity

The principle that structurally similar molecules tend to exhibit similar biological activity forms the basis of modern medicinal chemistry and informs the development of new drugs. The structure–activity relationship (SAR) describes the relationship between a molecule's chemical structure and its biological activity, indicating that modification of the chemical structure of a bioactive compound will result in a modification of its potency, or biological activity.¹ Further, SAR landscapes imply that the modification of a drug's scaffold is proportional to its activity in some way, meaning that altering key properties of the ligand result in a proportional increase or decrease in biological activity.² According to this principle, SAR can be classified as smooth landscapes in which ligand similarity is proportional to its observed activity *in vivo*.³ This proportionality is defined by medicinal chemistry as quantitative SAR (QSAR), which is a computational or mathematical model that correlates some calculated molecular property of a scaffold group to its experimentally-derived activity.¹ QSAR heavily informs virtual screening (VS) methods, which are *in silico* drug design experiments that screen for potential drug candidates using computationally calculable QSAR-based metrics. These computational drug-screening trials are leveraged in medicinal chemistry to reduce the cost and time typically associated with curating and synthesizing a large library of testable biological compounds.⁴ As such, it is highly beneficial to be capable of predicting drug activity from ligand similarity within such an experiment, which is why classifying SAR landscapes is of high import.

The structure-activity relationship is heavily leveraged in medicinal chemistry in other ways, as well. Ligand-based drug design is a method of lead development that occurs without knowing the 3D structure of the biological target, but rather altering the ligand scaffold based on

the properties the ligand already possesses⁵. Another commonly-used medicinal chemistry technique is bioisosteric replacement—the substitution of one functional group or ring system with a chemically or electronically equivalent moiety that preserves key binding interactions while improving pharmacokinetic or physicochemical properties to create a drug analog⁶. These replacements are intentionally performed to retain and improve specific properties of the pharmacophoric groups which contribute to the drug-like behaviour of the ligand, such as sterics and electronics. These swaps are typically informed by SAR data – identifying regions of key interactions can allow these to be exchanged with similar groups to give better drug-like properties such as binding affinity, stability, or membrane-permeability, without sacrificing drug potency.⁴ The assumption that there is a relationship between ligand structure and activity underpins most drug design workflows, and thus the existence of this phenomenon forms an important basis for the field of medicinal chemistry.

Despite the conceptual clarity of the SAR framework, its predictive power has been shown to have many exceptions. This concept, called SAR discontinuity, occurs when small, localized structural modifications produce unexpectedly large changes in biological potency (Figure 1). Activity cliffs are defined as pairs or sets of highly structurally similar compounds that are active against the same target but exhibit large potency differences. These are the embodiment of SAR discontinuity, confounding activity prediction via QSAR modelling while simultaneously revealing small chemical modifications with disproportionately large potency consequences.⁷ Activity cliffs are frequently observed in certain proteins such as kinases because they are highly flexible and undergo large conformational changes upon ligand binding, while having a highly conserved binding pocket shape across the kinome.⁸ Activity cliffs are both a challenge and an opportunity – during early lead identification they highlight key bioisosteres

and protein-ligand interactions, but during later-stage optimization they introduce noise that destroys the predictive capability of QSAR models. Activity cliffs can reveal small chemical modifications that have an exponential influence on activity, which is a desirable property when it leads to drastic improvement in drug-like properties in a drug candidate.⁹ In the context of CDK2 inhibitor series, activity cliffs have been identified across structurally diverse scaffold classes, where single heteroatom substitutions or minor ring regioisomerism can produce order-of-magnitude differences in IC₅₀ values.¹⁰ However, the existence of these non-linear regions logically contradicts the rationale of QSAR-based drug design approaches and thus renders these strategies to be case-dependent rather than broadly applicable.⁹ Within drug design studies, the presence of activity cliffs are typically considered to be local exceptions of the SAR landscape rather than global failures. Given that the root cause for these phenomena has not been explained satisfactorily, however, it is unclear whether we can confidently assign these cliffs to be local anomalies, or if they indicate larger limitations of SAR reasoning.³

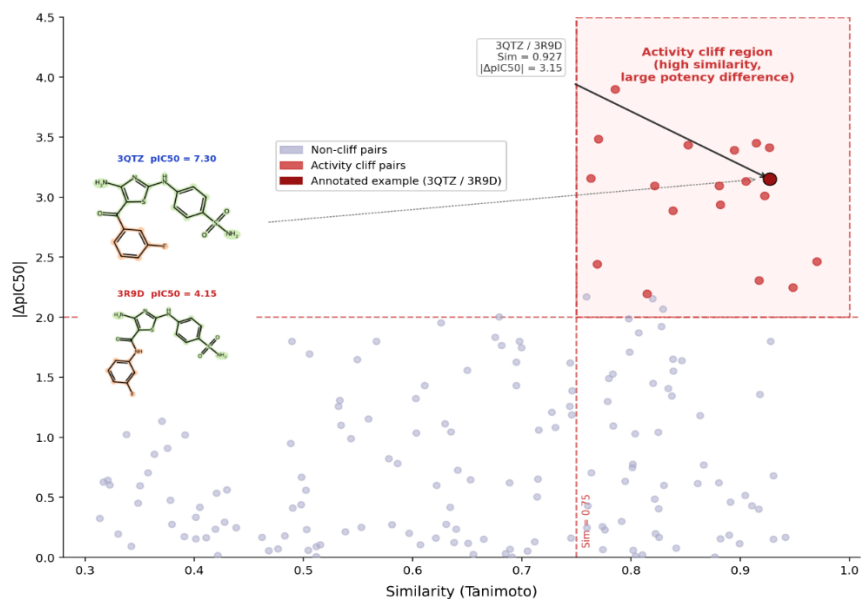


Figure 1. Representative activity cliff pair present within the CDK2 dataset, pictured in the high-similarity high-potency difference region.

The most common causes of SAR discontinuity are altered ligand binding modes and changes in protein conformation upon binding.¹¹ In a protein active site such as found in the kinase family, both phenomena are consistently observed upon ligand binding. The ATP-binding cleft undergoes induced-fit rearrangements upon inhibitor binding – particularly involving the glycine-rich loop, the hinge region, the α C-helix, and the DFG motif – meaning that the introduction of a different substituent can remodel the binding pocket and significantly alter the geometry of interactions.¹² Kinases are flexible proteins with many distinct conformations, leading to numerous instances of observable SAR discontinuities that manifest as activity cliffs in VS screens.¹³ Entropic effects compound this complexity: desolvation penalties, changes in ligand conformational entropy upon binding to the flexible kinase active site, and alterations in the dynamics of the protein are all non-additive contributions to the overall free energy of binding in ways that confound linear QSAR models.^{11,14} Activity cliffs in protein-ligand binding events present a major challenge for the application of ligand structural similarity-based data analysis techniques upon flexible protein datasets.¹⁴

1.2 Protein of Interest – Cyclin-Dependent Kinase 2

CDK2 (cyclin-dependent kinase 2) is a serine/threonine kinase whose activity gatekeeps the G1/S checkpoint of the cell cycle (DNA synthesis phase), and whose dysregulation is thus implicated in a broad spectrum of human cancers, making it a therapeutically important target.¹⁵ CDK2 was selected as the protein target system for this study for several reasons that make it uniquely well-suited to SAR-focused investigation in a flexible system. Due to the great interest in CDK2 for drug development as an anti-cancer target, there is a large dataset of CDK2 crystal structures with co-crystallized ligands available from the Protein Data Bank,¹⁶ which makes it an ideal target system for computational studies of structure-based ligand similarity.¹⁷ This

structural abundance, combined with extensive bioactivity and fingerprint data curated in databases such as ChEMBL and KLIFS^{18,19} means that CDK2 offers one of the most complete and well-validated SAR landscapes available for any kinase target and is ideal for computational descriptor analysis. The binding modes of CDK2 offer a novel method of study as well, given that CDK2 is a flexible system with several well-established, highly variable conformations, and its few ligand binding modes are highly conserved. Structurally, CDK2 inhibitors predominantly bind within the ATP-binding cleft, anchoring to the flexible kinase hinge region through hydrogen bonds to the backbone carbonyl of Glu81 and the amide nitrogen of Leu83—interactions that are preserved across virtually all known co-crystal structures of CDK2–inhibitor complexes.²⁰ The ATP-binding cleft undergoes induced-fit rearrangements upon inhibitor binding – particularly involving the glycine-rich loop, the α C-helix, and the DFG motif – meaning that the introduction of a new substituent can remodel the binding pocket and alter the geometry of interactions significantly (Figure 2). These conformational changes are well-represented within the available crystal pose data, and thus an evaluation of this database using QSAR metrics will yield a comprehensive SAR analysis of a flexible protein with multiple well-defined binding modes. This study directly correlates with many biological phenomena and thus provides valuable data for evaluating these metrics for use in lead-design studies. The structural diversity of CDK2 has been well characterized in terms of inhibitor binding mode classification, yet its relationship to activity cliff formation remains unexplored. Whether receptor conformations and the ligand scaffold specificity these conformations display contribute to SAR discontinuity beyond the ligand structural changes traditionally studied represents an open question this thesis addresses.

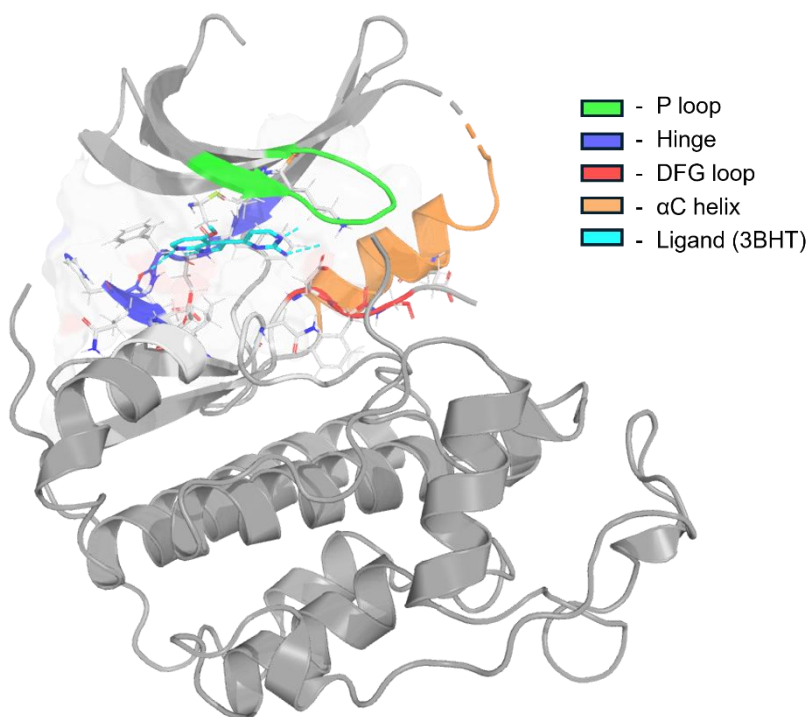


Figure 2. Monomeric CDK2 (chain A) ligand-receptor complex 3BHT (PDB ID: <https://doi.org/10.2210/pdb3BHT/pdb>) with key flexible ligand-binding regions colour-coded, and less key areas gray-scaled.²¹ P-loop contains the glycine-rich portion of the CDK2 chain. 3BHT ligand is coloured teal. Image created using PyMOL.²²

In this work, the extent to which molecular similarity predicts biological activity in a flexible protein is systematically evaluated using a range of QSAR-based molecular similarity metrics. Both ligand-based and structure-based methods were assessed to make this study more comprehensive and account for both ligand and ligand-protein interaction similarity. The Tanimoto coefficient is a commonly used cheminformatics method to compare the SAR between two ligands and was implemented to explore several aspects of ligand similarity.²³ In this work, Morgan Fingerprints were used to assess overall ligand similarity, given that they build a 2D vector-based representation of the model and indicate substructural similarity.^{24,25} Multiple 3D descriptors such as volumetric shape overlays and electrostatic potential maps were also examined to account for 3D ligand similarity.¹ ROCS Shape and Color Tanimoto metrics were used to evaluate 3D shape similarity and 3D chemical similarity of ligands, and electrostatic

potential (ESP) map Tanimotos were used to examine the electron density similarity of each ligand.^{26,27} Finally, ligand-receptor complex-based metrics such as molecular docking with AutoDock Vina and GNINA were examined.^{28,29} Molecular docking involves the docking of a potential drug ligand into its 3D crystal protein target, followed by the evaluation and ranking of the drug via physics- and knowledge-based scoring functions (Figure 3).^{30,31} KLIFS fingerprints were also used to assess protein-ligand complex similarity, being consensus fingerprints derived specifically for kinases that give information about ligand binding location, protein geometry, and residue identity.^{19,32} These ligand similarity-based and ligand-receptor metrics are used in this study to examine the capabilities of VS methods to predict QSAR relationships, as well as to predict instances of QSAR discontinuity in the form of activity cliffs. The capability of *in silico* methods to accurately predict instances of activity cliff formation is highly valuable and desirable, given that cliff-like modifications are capable of exponentially improving the drug-like properties of a ligand. However, the ability of computational methods to predict these occurrences remains largely elusive, and thus continues to be a subject of high relevance within the field of medicinal chemistry.^{33,34}

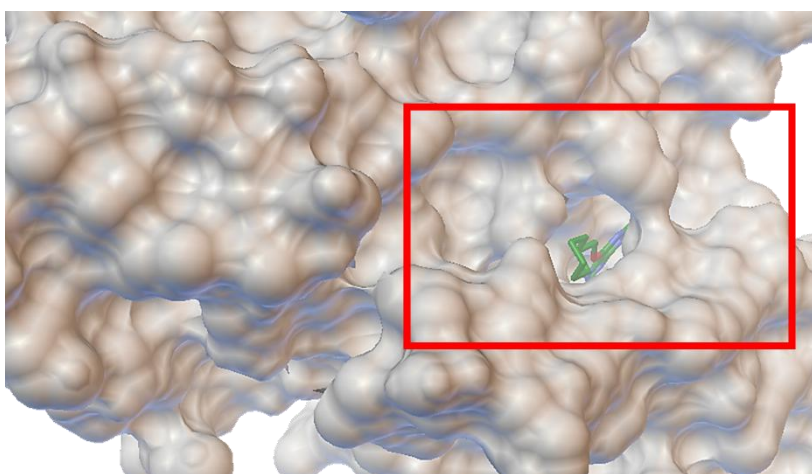


Figure 3. Molecular docking representation using CDK2 (ligand 1E1V, PDB ID: <https://doi.org/10.2210/pdb1E1V/pdb>).³⁵ Ligand highlighted within active site with a red box. Image created using Autodock Vina.³⁶

1.3 Purposes of this Work

2D, 3D, and protein-ligand structure-activity metrics provide a framework for forming QSAR relationships, enabling ligand-based drug design strategies such as scaffold hopping, pharmacophore modelling, and analogue series optimization.⁶ Recent progress in artificial intelligence, rapid growth of molecular databases, and dramatic improvements in computational power have led to the emergence of the use of deep learning to inform lead design experiments and indicate promising ligands. However, predictive models are only as good as the data that they are trained on. The integration of computationally derived QSAR metrics into prospective lead design workflows represents a growing area of methodological development, but it is important to note that the quality of the resulting models is critically dependent on a thorough understanding of where SAR is continuous and where it breaks down. The purpose of this work is to evaluate the use of such metrics and their validity as data to guide *in silico* drug design workflow protocols, as well as determine whether a wide range of computationally calculable metrics is more useful in describing an SAR landscape than one or two metrics, which is what is typically implemented in virtual screening experiments. This work will also seek to establish the role of protein flexibility and active site conformation in SAR landscapes, and determine if a representative ensemble of CDK2 receptors shows improved QSAR predictive power within molecular docking protocols for lead prediction. Finally, this study will explore the role of activity cliffs and SAR discontinuity in confounding SAR landscape prediction, and evaluate several methods of addressing the issues these present within VS screening methodologies, as well as assess the capabilities of the proposed similarity metrics to predict instances of SAR discontinuity within the CDK2 dataset.

While SAR forms the basis for most early-stage lead design projects, both *in silico* and *in vivo*, there are not many instances where SAR and QSAR methods have been tested systematically across multiple metrics and representations of these properties. Intuitively, one would expect that increasing understanding of each inhibitor's properties through the compilation of various QSAR metrics would lead to a better description of the overall SAR and thus better guide lead design. However, these metrics have rarely been evaluated in regard to their capabilities of describing the ligand-receptor complex and its accompanying biological activity as an amalgamation of different structural metrics.³⁷ It is even rarer to observe this sort of study in a flexible protein system such as the kinases, due to the confounding properties of different conformational arrangements, which can lead to activity cliffs and non-additivity in the dataset. The presence of activity cliffs within the CDK2 dataset is a known phenomenon, but one which proves elusive to prediction using molecular docking methodology and therefore serves to confound VS prediction of SAR continuity. Thus the purpose of this work is to perform such a study, to evaluate whether the use of multiple activity metrics can improve the understanding of the SAR of a flexible protein and resolve some of the observable SAR discontinuity which is present within the CDK2 SAR landscape.

Sources

- (1) Kumar, A.; Zhang, K. Y. J. Advances in the Development of Shape Similarity Methods and Their Application in Drug Discovery. *Front. Chem.* **2018**, *6*, 1–21. <https://doi.org/10.3389/fchem.2018.00315>.
- (2) Guha, R. On Exploring Structure-Activity Relationships. *Methods in Molecular Biology* **2013**, *993*, 81–94. https://doi.org/10.1007/978-1-62703-342-8_6.
- (3) Maggiora, G. M. On Outliers and Activity Cliffs - Why QSAR Often Disappoints. *J. Chem. Inf. Model.* **2006**, *46* (4), 1535. <https://doi.org/10.1021/ci060117s>.
- (4) Lionta, E.; Spyrou, G.; Vassilatis, D. K.; Cournia, Z. Structure-Based Virtual Screening for Drug Discovery: Principles, Applications and Recent Advances. *Curr. Top. Med. Chem.* **2014**, *14*, 1923–1938.
- (5) Acharya, C.; Coop, A.; Polli, J. E.; MacKerell, A. D. Recent Advances in Ligand-Based Drug Design: Relevance and Utility of the Conformationally Sampled Pharmacophore Approach. *Curr. Comput. Aided Drug. Des.* **2011**, *7* (1), 10–22.
- (6) Chang, Y.; Hawkins, B. A.; Du, J. J.; Groundwater, P. W.; Hibbs, D. E.; Lai, F. A Guide to In Silico Drug Design. *Pharmaceutics* **2023**, *49* (15), 1–52. <https://doi.org/10.3390/pharmaceutics15010049>.
- (7) Stumpfe, D.; Hu, H.; Bajorath, J. Advances in Exploring Activity Cliffs. *J. Comput. Aided. Mol. Des.* **2020**, *34* (9), 929–942. <https://doi.org/10.1007/s10822-020-00315-z>.
- (8) Daoud, S.; Taha, M. Protein Characteristics Substantially Influence the Propensity of Activity Cliffs among Kinase Inhibitors. *Sci. Rep.* **2024**, *14* (1). <https://doi.org/10.1038/s41598-024-59501-w>.
- (9) Bajorath, J. Duality of Activity Cliffs in Drug Discovery. *Expert Opin. Drug Discov.* **2019**, *14* (6), 517–520. <https://doi.org/10.1080/17460441.2019.1593371>.
- (10) Jorda, R.; Havlíček, L.; McNae, I. W.; Walkinshaw, M. D.; Voller, J.; Šturc, A.; Navrátilová, J.; Kuzma, M.; Mistrik, M.; Bártek, J.; Strnad, M.; Kryštof, V. Pyrazolo[4,3-d]Pyrimidine Bioisostere of Roscovitine: Evaluation of a Novel Selective Inhibitor of Cyclin-Dependent Kinases with Antiproliferative Activity. *J. Med. Chem.* **2011**, *54* (8), 2980–2993. <https://doi.org/10.1021/jm200064p>.
- (11) Gomez, L.; Xu, R.; Sinko, W.; Selfridge, B.; Vernier, W.; Ly, K.; Truong, R.; Metz, M.; Marrone, T.; Sebring, K.; Yan, Y.; Appleton, B.; Aertgeerts, K.; Massari, M. E.; Breitenbucher, J. G. Mathematical and Structural Characterization of Strong Nonadditive Structure-Activity Relationship Caused by Protein Conformational Changes. *J. Med. Chem.* **2018**, *61* (17), 7754–7766. <https://doi.org/10.1021/acs.jmedchem.8b00713>.

- (12) Li, Y.; Zhang, J.; Gao, W.; Zhang, L.; Pan, Y.; Zhang, S.; Wang, Y. Insights on Structural Characteristics and Ligand Binding Mechanisms of CDK2. *Int. J. Mol. Sci.* **2015**, *16* (5), 9314–9340. <https://doi.org/10.3390/ijms16059314>.
- (13) Stumpfe, D.; Hu, H.; Bajorath, J. Evolving Concept of Activity Cliffs. *ACS Omega* **2019**, *4* (11), 14360–14368. <https://doi.org/10.1021/acsomega.9b02221>.
- (14) Kramer, C.; Fuchs, J. E.; Liedl, K. R. Strong Nonadditivity as a Key Structure-Activity Relationship Feature: Distinguishing Structural Changes from Assay Artifacts. *J. Chem. Inf. Model.* **2015**, *55* (3), 483–494. <https://doi.org/10.1021/acs.jcim.5b00018>.
- (15) Wang, S.; Meades, C.; Wood, G.; Osnowski, A.; Anderson, S.; Yuill, R.; Thomas, M.; Mezna, M.; Jackson, W.; Midgley, C.; Griffiths, G.; Fleming, I.; Green, S.; McNae, I.; Wu, S. Y.; McInnes, C.; Zheleva, D.; Walkinshaw, M. D.; Fischer, P. M. 2-Anilino-4-(Thiazol-5-Yl)Pyrimidine CDK Inhibitors: Synthesis, SAR Analysis, X-Ray Crystallography, and Biological Activity. *J. Med. Chem.* **2004**, *47* (7), 1662–1675. <https://doi.org/10.1021/jm0309957>.
- (16) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. The Protein Data Bank. *Nucleic Acids Res.* **2000**, *28* (1), 235–242.
- (17) Pisani, P.; Caporuscio, F.; Carlino, L.; Rastelli, G. Molecular Dynamics Simulations and Classical Multidimensional Scaling Unveil New Metastable States in the Conformational Landscape of CDK2. *PLoS One* **2016**, *11* (4). <https://doi.org/10.1371/journal.pone.0154066>.
- (18) Hunter, F. M. I.; Ioannidis, H.; Bento, A. P.; Bosc, N.; Corbett, S.; Felix, E.; Magarinos, M. P.; Manners, E.; Smit, I. A.; de Veij, M.; O’Boyle, N. M.; Zdrazil, B.; Leach, A. R. Drug and Clinical Candidate Drug Data in ChEMBL. *J. Med. Chem.* **2025**, *68* (19), 19800–19827. <https://doi.org/10.1021/acs.jmedchem.5c00920>.
- (19) Van Linden, O. P. J.; Kooistra, A. J.; Leurs, R.; De Esch, I. J. P.; De Graaf, C. KLIFS: A Knowledge-Based Structural Database to Navigate Kinase-Ligand Interaction Space. *J. Med. Chem.* **2014**, *57* (2), 249–277. <https://doi.org/10.1021/jm400378w>.
- (20) Noble, M. E. M.; Endicott, J. A. Chemical Inhibitors of Cyclin-Dependent Kinases. *Pharmacol. Ther.* **1999**, *82* (2–3), 269–278.
- (21) Bettayeb, K.; Tirado, O. M.; Marionneau-Lambot, S.; Ferandin, Y.; Lozach, O.; Morris, J. C.; Mateo-Lozano, S.; Drueckes, P.; Schächtele, C.; Kubbutat, M. H. G.; Liger, F.; Marquet, B.; Joseph, B.; Echalié, A.; Endicott, J. A.; Notario, V.; Meijer, L. Meriolins, a New Class of Cell Death-Inducing Kinase Inhibitors with Enhanced Selectivity for Cyclin-Dependent Kinases. *Cancer Res.* **2007**, *67* (17), 8325–8334. <https://doi.org/10.1158/0008-5472.CAN-07-1826>.

- (22) Schrödinger, L. PyMOL - The PyMOL Molecular Graphics System.
- (23) Tanimoto, T. T. An Elementary Mathematical Theory of Classification and Prediction; International Business Machines Corporation: New York, 1958; pp 1–10.
- (24) Zhou, H.; Skolnick, J. Utility of the Morgan Fingerprint in Structure-Based Virtual Ligand Screening. *Journal of Physical Chemistry B* **2024**, *128* (22), 5363–5370. <https://doi.org/10.1021/acs.jpcb.4c01875>.
- (25) Morgan, H. L. The Generation of a Unique Machine Description for Chemical Structures. *J. Chem. Doc* **1965**, *5* (2), 107–113.
- (26) OpenEye Scientific Software, S. F. N. *EON 2.4.0.1*. <http://www.eyesopen.com>.
- (27) OpenEye, C. M. S. S. F. N. *ROCS 3.5.1.1*. <http://www.eyesopen.com>.
- (28) Eberhardt, J.; Santos-Martins, D.; Tillack, A. F.; Forli, S. AutoDock Vina 1.2.0: New Docking Methods, Expanded Force Field, and Python Bindings. *J. Chem. Inf. Model.* **2021**, *61* (8), 3891–3898. <https://doi.org/10.1021/acs.jcim.1c00203>.
- (29) McNutt, A. T.; Francoeur, P.; Aggarwal, R.; Masuda, T.; Meli, R.; Ragoza, M.; Sunseri, J.; Koes, D. R. GNINA 1.0: Molecular Docking with Deep Learning. *J. Cheminform.* **2021**, *13* (1). <https://doi.org/10.1186/s13321-021-00522-2>.
- (30) Stepišnik, T.; Škrlić, B.; Wicker, J.; Kocev, D. A Comprehensive Comparison of Molecular Feature Representations for Use in Predictive Modeling. *Comput. Biol. Med.* **2021**, *130*. <https://doi.org/10.1016/j.combiomed.2020.104197>.
- (31) Meng, X.-Y.; Zhang, H.-X.; Mezei, M.; Cui, M. Molecular Docking: A Powerful Approach for Structure-Based Drug Discovery. *Curr Comput Aided Drug Des* **2011**, *7* (2), 146–157.
- (32) Kanev, G. K.; de Graaf, C.; Westerman, B. A.; de Esch, I. J. P.; Kooistra, A. J. KLIFS: An Overhaul after the First 5 Years of Supporting Kinase Research. *Nucleic Acids Res.* **2021**, *49* (D1), D562–D569. <https://doi.org/10.1093/nar/gkaa895>.
- (33) Furtmann, N.; Hu, Y.; Bajorath, J. Comprehensive Analysis of Three-Dimensional Activity Cliffs Formed by Kinase Inhibitors with Different Binding Modes and Cliff Mapping of Structural Analogues. *J. Med. Chem.* **2015**, *58* (1), 252–264. <https://doi.org/10.1021/jm5009264>.
- (34) Guha, R.; Van Drie, J. H. Structure - Activity Landscape Index: Identifying and Quantifying Activity Cliffs. *J. Chem. Inf. Model.* **2008**, *48* (3), 646–658. <https://doi.org/10.1021/ci7004093>.
- (35) Arris, C. E.; Boyle, F. T.; Calvert, A. H.; Curtin, N. J.; Endicott, J. A.; Garman, E. F.; Gibson, A. E.; Golding, B. T.; Grant, S.; Griffin, R. J.; Jewsbury, P.; Johnson, L. N.;

- Lawrie, A. M.; Newell, D. R.; Noble, M. E. M.; Sausville, E. A.; Schultz, R.; Yu, W. Identification of Novel Purine and Pyrimidine Cyclin-Dependent Kinase Inhibitors with Distinct Molecular Interactions and Tumor Cell Growth Inhibition Profiles. *J. Med. Chem.* **2000**, *43* (15), 2797–2804. <https://doi.org/10.1021/jm990628o>.
- (36) Trott, O.; Olson, A. J. AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading. *J. Comput. Chem.* **2010**, *31* (2), 455–461. <https://doi.org/10.1002/jcc.21334>.
- (37) Maggiora, G.; Vogt, M.; Stumpfe, D.; Bajorath, J. Molecular Similarity in Medicinal Chemistry. *J. Med. Chem.* **2014**, *57* (8), 3186–3204. <https://doi.org/10.1021/jm401411z>.

Chapter 2 – Methods

In this work, several datasets of computationally modelled CDK2 (cyclin-dependent kinase 2) protein-ligand complexes were curated. The crystal-posed dataset described below were derived from the coordinates of real co-crystallized receptor-ligand structures derived using X-ray crystallographic and adjacent techniques, and obtained from the RCSB PDB (Research Collaboratory for Structural Bioinformatics Protein Data Bank, [RCSB.org](https://www.rcsb.org)) where they were deposited.¹ Their accompanying potency IC₅₀ values were obtained from the ChEMBL database, which is a manually curated open-access database of bioactive molecules with accompanying drug-like properties.² The ChEMBL database was also used to generate the sets of data which did not have crystal-posed structures later described here. Table 4 at the end of this chapter may be referred to for a summary of all modules and software implemented in this chapter.

2.1 Curation and Preparation of Datasets

2.1.1 Curation and Preparation of Representative Co-crystallized CDK2 Dataset

The purpose of curating this dataset and putting together the preparation pipeline was to organize a series of high-quality 3D crystal-posed CDK2 ligand-receptor complexes for downstream ligand similarity and molecular docking analysis. Retaining the 3D crystal pose of the ligand-receptor complex was highly important to the study, given that the effects of a known native ligand pose and receptor active site configuration on ligand structure-activity relationships and molecular docking was one of the central ideas for this work.

A search was performed for RCSB PDB deposited crystal structures within the KLIFS (Kinase-Ligand Interaction Fingerprints and Structures) resolution cutoff of 0.0-2.0Å that had co-crystallized ligands within the receptor.^{1,3} The results of the preliminary search were filtered

to exclude entries which had missing residues or missing atoms, any duplicate entries resulting from duplicate chains or multiple different PDB deposits of the same PDB structure, structures lacking an orthosteric ligand and structures which contained allosteric ligands, and structures whose pocket sequence did not match the KLIFS defined sequence for an active site pocket. This series of filters produced a preliminary dataset of 180 CDK2 crystal structures.

For each structure, the bound ligand was identified using its three-digit PDB ID code, which was mapped to its InChI (International Chemical Identifier) key using the RCSB Ligand Expo ligand table.¹ These keys were then mapped to ChEMBL molecular identifiers using UniChem. ChEMBL identifiers were found for 119 of the 180 ligands, and the other 66 ligands were discarded. The CDK2 ligand IC50 data values for each ligand represented by the ChEMBL identifiers within the remaining 119-crystal-ligand dataset were collected, resulting in IC50 data for only 31 ligands, meaning that to obtain a more sizeable dataset the ChEMBL IC50 data records needed to be manually curated. Any values which were not exact (e.g. less than, greater than) were removed. Cell-based assays and assays described as using mutant protein were excluded, and one outlier with an IC50 conversion error was excluded. When ligands had more than one accompanying IC50 value, the values were averaged to obtain one representative score.

These sets from the KLIFS and RCSB databases were then combined into a single dataset containing their PDB ID, ligand ID, and IC50 in nM. The dataset consisted of 98 protein-ligand complexes. The activity values were then further checked against primary literature sources, which resulted in 19 IC50 values being updated. One ligand was discarded because its activity was reported as greater than 100000 nM, making the final curated CDK2 activity dataset 97 ligands with exact or averaged IC50 values.

This dataset was then processed for analysis via ligand similarity methods and molecular docking: PDB structures were downloaded using ProDy and parsed, removing any additional conformers for each ligand, then removing any non-residue molecules present in the protein (waters, metals, other amino acids), and retaining the largest molecule, which is presumably the ligand with its atomic coordinates, for analysis.⁴ Then the protein was parsed using ProDy and any capping groups and modified residues were removed, which cannot be read by downstream analysis platforms – these were cut out and the protein was then rebuilt using the retained atoms and atom coordinates.⁴ At this point ProDy was used to fill in any missing residues or atoms in order to ensure the protein can be read, and a Python writer rewrites the file to as an output PDB file in the same format as the input PDB file so it can be read by alignment and evaluation platforms down the pipeline.

The protein-ligand dataset was then superimposed using MUSTANG (Multiple STructural AligNment AlGorithm), which aligns the individual protein crystal structures in space using the α C-RMSD (Root Mean Square Deviation) of corresponding residues in the template so they can be directly compared later in the pipeline.⁵ MUSTANG requires a reference protein crystal structure, so all proteins in the dataset were aligned to the CDK2 protein crystal structure of PDB ID 1E1V (PDB ID:<https://doi.org/10.2210/pdb1E1V/pdb>).⁶ The transformation matrices for each protein structure alignment were then obtained by parsing the transformed files and applying the Kabsch transformation to them – aligning the cloud of RMSD values temporarily at the origin, rotating the two clouds to find a rotational matrix match, then finding the actual transformation for each matrix.^{7,8} The resulting rotation and translation matrix is applied to the query ligand, allowing it to be translated back into its place in the superimposed protein active site. Next, ProDy was used to retain only the protein chain which contained the most KLIFS

active site-defined residues, in the cases where more than one chain existed within the RCSB PDB-downloaded structures.^{3,4}

The next concern was properly generating the ligand 3D structure, which is more difficult than one might expect. Openeye's OEToolkit was used to query the RCSB database (RCSB.org) for the SMILES string associated with each ligand PDB ID, which contains the full ligand connectivity pattern.^{1,9} Dimorphite-DL was then used to protonate the resulting ligand structures to physiological pH (pH = 7.4, RDKit was used as a fallback in this step).^{10,11} Then the downloaded PDB coordinate files for each ligand were loaded and the SMILES string connectivity ligand was threaded into each set of ligand 3D coordinates corresponding to each ligand using RDKit, which was then used to add hydrogens to the ligand based on its 3D structure.¹¹ This step was done in an attempt to preserve ligand connectivity for docking, because the connectivity of ligands is often not retained during the step of converting a ligand from a PDB (3D crystal structure file) to a PDBQT (molecular docking-ready) file. Gastieger partial charges were assigned to each atom in the ligands using RDKit and written into the prepared ligand PDB file.¹²

The proteins were then prepared: PDBFixer was used to fix partial residues that had not been fully resolved, add capping atoms to chains if they were cut off (because only the chain with the KLIFS pocket was retained), and protonate to physiological pH (pH = 7.4).¹³ The hydrogens were relaxed using the AMBER14 forcefield and a Langevin integrator at 298K, with a 0.9nm cutoff for van der Waals interactions.^{13,14}

Finally, both proteins and ligands were converted from PDB files to molecular-docking-ready PDBQT files, adding information about partial charges, AutoDock atom types, and which atoms can rotate (torsional flexibility). This step is the most error-prone, as the conversion

sometimes causes the loss or mishandling of aromaticity in the ligand, causing issues with ligand posing during molecular docking.¹⁵ Meeko was used to perform this operation, as it is integrated with parsing RDKit-prepared ligands and is a modern, actively maintained tool, with AutoDock Tools used as a fallback option given that it has its own internal parsing function and would re-interpret the process described here.^{16,17}

Two more ligands were dropped here, one because of ligand parsing errors (the ligand was registered as a non-ligand junk molecule and removed), and the other because it was almost entirely aromatic, and therefore was mishandled by all of the PDBQT conversion softwares used – the aromaticity was re-interpreted as single carbon-carbon bonds. This left a final database of 95 ligand-receptor complexes.

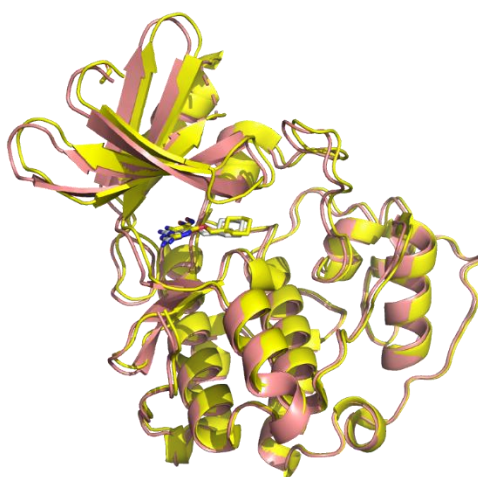


Figure 1. Picture of ligand-receptor complexes for CDK2 dataset protein 1E1X (PDB ID: <https://doi.org/10.2210/pdb1E1X/pdb>) before alignment and preparation (yellow), and after alignment and preparation (salmon).⁶ Images generated using PyMOL.¹⁸

2.1.2 Active Site RMSD-based Clustering of Crystal-Posed CDK2 Dataset

To determine whether the dataset of CDK2 crystal posed ligand-receptors contained several distinct active site conformations, the α C helix RMSD was determined for all of the receptors present in the dataset (residues 43-54, specifically) using RDKit.¹¹ Here, the non-

protonated and relaxed form of the proteins were used, directly after MUSTANG alignment, because PDBFixer and OpenMM treatment can sometimes shift backbone residues. Because the active sites have already been aligned, the α C helix RMSD can easily be calculated between the receptors. The comparison required a reference protein, so the CDK2 protein crystal structure of PDB ID 1E1V (<https://doi.org/10.2210/pdb1E1V/pdb>) was again used as a reference.⁶ A kernel density estimation plot of the raw RMSD data was first plotted, to visualize whether the data had distinct clusters – it appeared to have two, a major one with a database density of 0.8 and a median RMSD value of $\sim 0.75 \text{ \AA}$ from the reference structure.

Based on these results, a Gaussian Mixture Model (GMM) was used to model the data for a candidate set of clusters (n=1 through 8 was tested) to determine what the best cluster fit was for the dataset RMSD, which was found to be n = 3. This was performed through a series of “expectation” steps, where each data point’s probability of belonging to each cluster was calculated; and a series of “maximization” steps, where the clusters are updated to best fit the weight and spread of the data points which have been assigned to them.¹⁹ Scikit-learn was used to refit the data 20 times, starting from random points in the dataset to ensure that GMM was not getting stuck in local minima.²⁰ A scoring criterion was used to evaluate the data fit in order to determine number of clusters, BIC (Bayesian Information Criterion), to prevent the data being overfit. It weighs the benefit of adding additional clusters to better fit the data with a complexity penalty, allowing the fitting of the data to several clusters without each point becoming its own cluster (which would be the optimal data fit).²¹ GMM was then used to define the data points within each cluster and classify which PDB structure fit in each of the three clusters which were generated, and these clusters were again visualized using KDE. Scikit-learn was used to perform all GMM and BIC calculations.²⁰ It should be noted that because the clustering is performed in

reference to a structure, clustering will be slightly different if a different reference structure is chosen – 1E1V falls into the α C-helix-out-B cluster, which is a non-active conformation of the CDK2 binding pocket, so choosing a reference receptor with a binding pocket conformation closer to the native binding conformation (α C-helix-in) may be the most intuitive way to cluster these in the future.

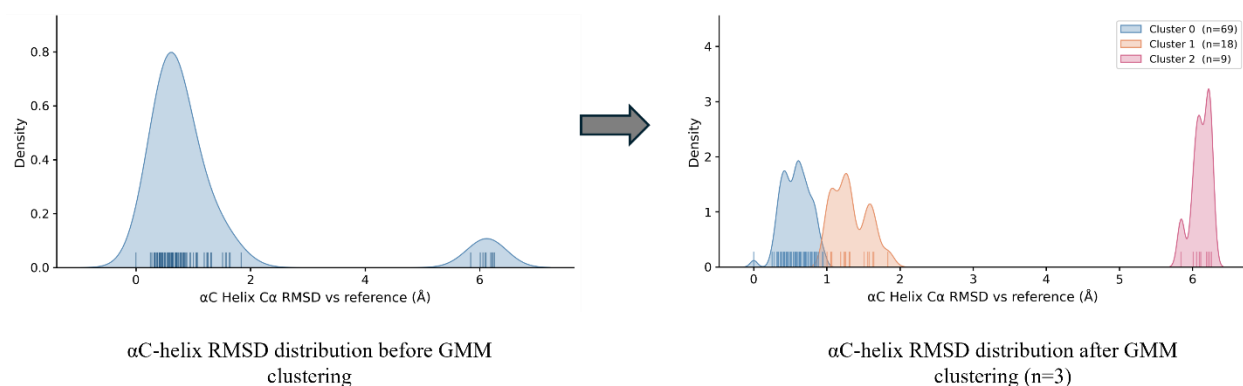


Figure 2. The α C helix RMSD structural data distribution before and after GMM clustering was applied to the data.

These clusters were then evaluated based on binding mode to determine two large differences in binding modes across all three conformational clusters:

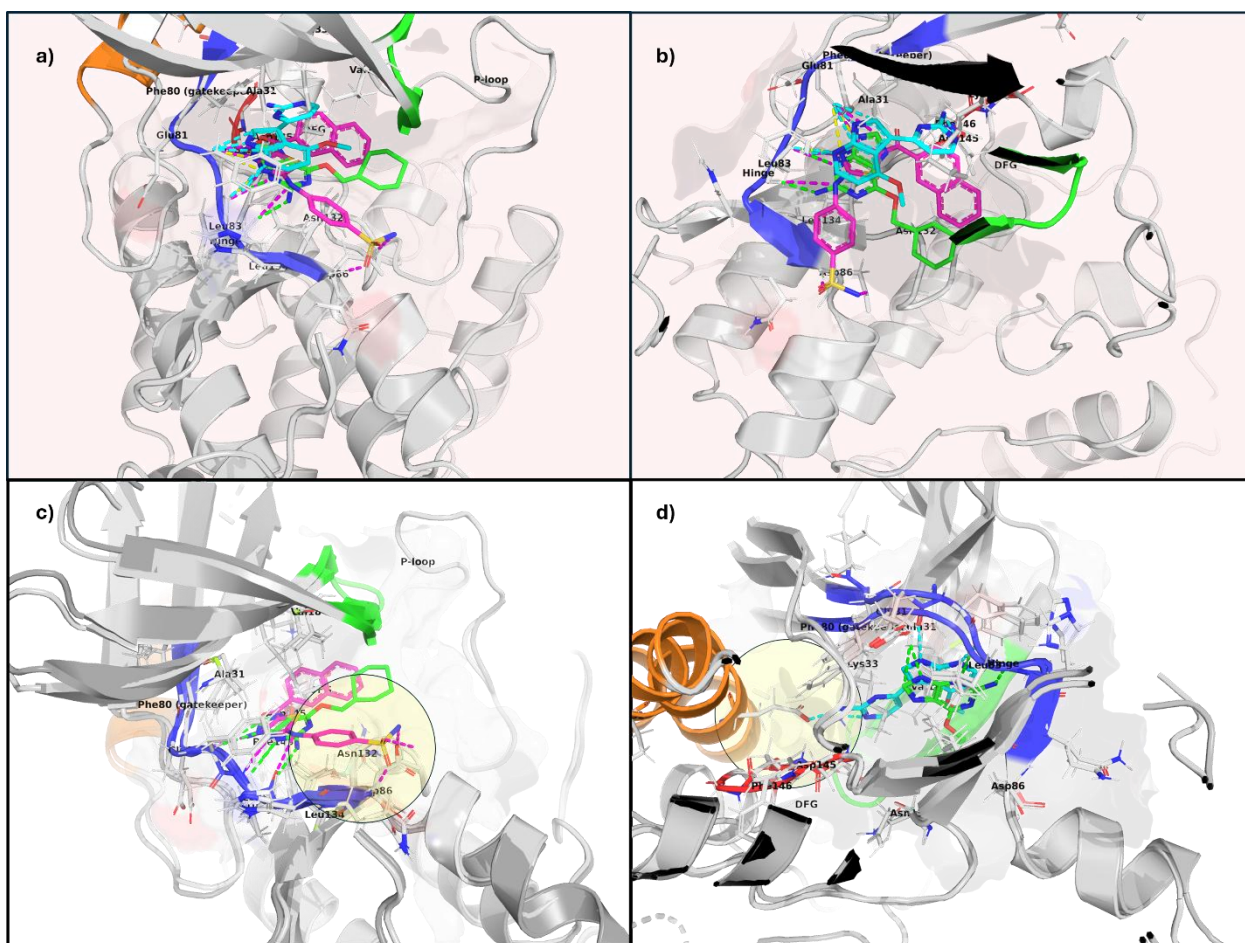


Figure 3. Visualizations of binding mode clusters in accordance with the α C helix RMSD clustering. Green ligand corresponds to ligand 1E1V, binding mode α C-out-B (PDB ID: <https://doi.org/10.2210/pdb1E1V/pdb>),⁶ blue ligand corresponds to ligand 3BHT, binding mode α C-in (PDB ID: <https://doi.org/10.2210/pdb3BHT/pdb>),^{22,23} pink ligand corresponds to ligand 3RMF, binding mode α C-out-A (PDB ID: <https://doi.org/10.2210/pdb3RMF/pdb>).²⁴ a) and b) show the three ligands overlaid within the native 1E1V receptor. c) shows the overlap between 1E1V (α C-out-B) and 3RMF (α C-out-A) ligand-receptor complexes, with areas of large difference circled in yellow. d) shows the overlap between 1E1V (α C-out-B) and 3BHT (α C-in) ligand-receptor complexes, with areas of large difference circled in yellow. Images modelled using PyMOL.¹⁸

The large differences in binding mode across the three conformational groupings shown in Figures 3a and 3b correspond with what is observed in the α C-helix RMSD. The α C-helix RMSD observed between the α C-out-B and the α C-out-A classifications is very small and not very noticeable upon overlay of the two receptors, but the ligands tend to make “back-pocket” contacts in a region perpendicular to the hinge region within the active site within the α C-out-A group (Figure 3c). In contrast, the α C-helix RMSD observed between the α C-out-B and α C-in conformations is much more prominent, with the α C-in α C-helix being positioned visibly closer

to the active site and thus positioning the DFG loop to form additional contacts with the bound ligand (Figure 4d). It is evident that three distinct binding modes are present within the CDK2 crystal pose dataset which correspond with the three different observed receptor conformations, grouped based on α C-helix RMSD.

2.1.3 Curation and Preparation of Representative ChEMBL Sample Sets

The purpose of curating these sample sets and putting together the preparation pipeline was to organize a series of non-posed CDK2 ligands with which to contrast the CDK2 RCSB crystal posed dataset ligand similarity results against (Figure 4). The goal was to determine if knowing the native ligand pose improves understanding of structure-activity relationships within the CDK2 ligand chemical space; essentially, to act as decoy sets with which to assess the utility of 3D ligand-receptor pose in establishing SARs.

First, the ChEMBL database was queried for all CDK2-related bioactivity records.² For every unique molecule referenced in these bioactivity records, its ChEMBL ID was obtained and the corresponding SMILES string for the ligand structure was fetched. RDKit MolStandardize was used to remove salts and counterions, standardize functional groups (e.g. consistent carboxylic acid group encoding), re-ionize, and collapse tautomers into a singular structure so these are not counted as two different structures¹¹. The resulting SMILES strings of the CDK2 ligands were retained for further filtering. Very few data quality filters were placed upon the ChEMBL dataset, given that these sample sets were generated as a low-structural information, low-activity data information contrast to the high-quality curated sets generated in the RCSB crystal-posed CDK2 dataset described in Section 3.1.1. As such, no confidence score filter or activity data validity filter was placed on this ligand search.

To exclude any ligand which had a corresponding crystal pose, the RCSB PDB database's search API was queried for all human CDK2 ligands (Uniprot Accession no: P24941, RCSB.org) and these were listed into an output text file.¹ These are sent through a series of filters: first, removal of any water, protein modifications that had been read and retained as ligand molecules incorrectly, crystallization solutions, salts, and heavy atoms, to isolate for true ligand molecules. Second, the retained molecules were required to have at least four heavy (non-hydrogen) atoms, and to exceed a molecular weight of 150 Da. The ligands which pass through these filters are to be considered a ligand with a 3D crystal-posed structure. Their corresponding SMILES strings were then fetched from the RCSB database.¹

The set of ChEMBL ligands SMILES strings with accompanying activity data were compared to the set of gathered RCSB SMILES strings, and any ligands which appeared within the RCSB set were removed from the ChEMBL set, to ensure that all ligands in these sample sets did not have accompanying 3D crystal poses. All other ChEMBL ligands were retained. The ChEMBL ligands were then filtered based on activity data assay type and only those with accompanying IC₅₀, pIC₅₀, K_i, or pK_i values were retained. The activity data were then calculated to uniform pIC₅₀ and pK_i values in nM units and split into 2 categories – those ligand SMILES strings with accompanying pIC₅₀ data, and those with accompanying pK_i data. Ligands with more than one activity data record were collapsed by assigning the lowest reported number for that ligand as its activity value. A rough 3D-structure generation was performed on these SMILES strings using RDKit's ETKDG ("Experimental-Torsion Knowledge Distance Geometry") conformer generator, which generates a reasonable conformation from the SMILES string as a PDB file, upon which the UFF (Universal Force Field) is applied to relax the 3D structure to a low-energy conformation.^{25,26} Three random sample sets of 100 ligands were drawn

from the resulting two groups of CDK2 ligand 3D structures (pIC50 activity data group, pKi activity data group) using random sampling of the group. Note that ligands can be repeated in different representative sets (Set 1, Set 2, Set 3), but not in the same set. Repeating this sampling left six groups of random sample sets, three with accompanying pIC50 data, and three with accompanying pKi data. Three random sample sizes were chosen for each activity data assay method because they allowed a robust check of the reproducibility of the results of ligand similarity testing upon this dataset. The random samples are then output as a series of PDB files generated from the RDKit conformer generation and relaxation step, and an Excel csv containing all ligand ChEMBL ID's, SMILES strings, and activity data, for further processing.

The final preparation step for this dataset was conformer generation for the ChEMBL ligands. Because they had no pre-existing 3D structure, their ligand similarity metrics were to be instead calculated upon the best ligand conformer match for each ligand. As such, Openeye OMEGA was used to generate up to 200 conformers for each ligand, with a threshold of 5Å – that is to say, each ligand conformation's heavy atom RMSD needed to be $\geq 5\text{\AA}$ from a previously generated conformation to be retained as a valid conformation for that ligand.^{27,28} OMEGA's default energy window was used, which is 10kcal/mol – meaning it energetically minimizes the initial conformer generation using MMFF94 (Merck Molecular Force Field) to find the lowest energy conformer, then discards any conformers it finds to be 10 kcal/mol above this baseline in energy²⁷. It should be noted that OMEGA's strict stereochemistry filter was turned off to expand sampling of the dataset, but this may mean sampling at ambiguous stereocenters will result in the wrong enantiomer being retained. No OMEGA macrocycle package was available so there were two fallbacks: using OEChem to generate a single conformer for macrocycles which frustrated OMEGA's conformer generation, and, failing that,

generating a single conformer using OpenBabel.^{29,30} In all three sample sets with accompanying IC50 data, nine out of three hundred ligands failed the error handling due to macrocycle mishandling and were discarded – four in Set 1, two in Set 2, three in Set 3. In all three sample sets with accompanying K_i data, three out of three hundred ligands failed due to macrocycle mishandling and were discarded – one in each set.

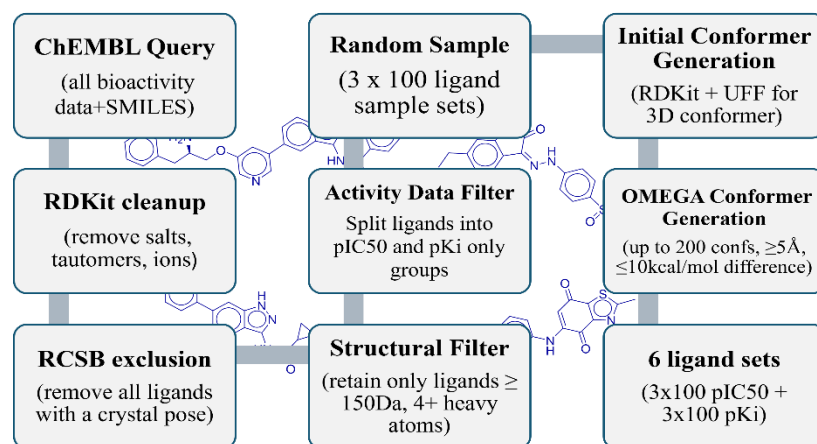


Figure 4. Workflow showing generation and preparation of CDK2 ChEMBL ligand sample sets. Ligands are generated from ChEMBL-obtained SMILES strings from Set 1.²

2.2 Ligand Similarity Metric Computations and Molecular Docking Computations

2.2.1 Ligand Similarity Metric Definitions

Within this work, four different ligand similarity metrics were leveraged to compare different properties of ligand similarity: ligand shape/steric similarity (Shape Tanimoto), ligand pharmacophoric similarity (Color Tanimoto), ligand electrostatic potential similarity (ESP Tanimoto), and ligand 2D chemical similarity (Morgan ECFP4 Tanimoto).

The Tanimoto coefficient was applied as a quantitative measure of ligand similarity in this study. The Tanimoto coefficient, represented by the formula $T_{A,B} = \frac{c}{a+b-c}$ where a is the number of “on” bits in one molecule and b is the number of “on” bits in the comparison molecule, and c is the number of overlapping “on” bits (the number of “on” bits in both

molecules), is the cheminformatics standard metric for representing ligand similarity.^{31,32} The Tanimoto coefficient ranges from values of 0 to 1, with a higher Tanimoto coefficient value corresponding to a greater ligand similarity. In the literature it has been proposed that Tanimoto coefficients of ≥ 0.85 reflect a high probability of the two compounds having the same activity, however, more recently this idea has been heavily questioned and disproven.³³

ROCS (**R**apid **O**verlay of **C**hemical **S**tructure**S**) Shape Tanimoto is a QSAR measurement of the shape-based superposition of two molecules which have been aligned through space to optimize the overlap between them.^{34,35} The ROCS algorithm calculates the volume of each atom as a Gaussian function, rather than a hard-edged sphere volume, which allows the combination of atoms in each molecule to be combined into a smooth Gaussian representation which is fast and computationally efficient to calculate in comparison to calculating the volumetric overlap of many spheres.^{36,37} One molecule is thus aligned upon the other in such a way as to maximize the overlap between both molecules (Figure 5). The Shape Tanimoto is then calculated using a version of the Tanimoto coefficient where the two molecules own self-overlap volumes (a , b , this is literally calculating the overlap of the combination of all atom Gaussians in the molecule with itself) and the overlap between the molecules (c) is contrasted.^{34,35}

ROCS Color Tanimoto is a QSAR measure of ligand pharmacophoric similarity which is also an extension of the ROCS Shape overlay.^{34,36} From the volumetric overlaid position of two molecules, ROCS uses SMARTS, a substructural grouping language, to identify pharmacophoric groups.^{38,39} The Color force-field sorts these groups into one of six designated chemical group categories: hydrogen bond donors, acceptors, cations, anions, hydrophobic groups, and aromatic groups.³⁴ Mills-Dean force fields are used to properly charge the molecules, and the Color groups are placed as harder Gaussian functions around the designated pharmacophoric groups.⁴⁰

ROCS then calculates the self-overlap of the ligands (a , b) and the volumetric overlap between the two aligned ligands (c) in much the same way as the Shape Tanimoto is calculated, but with the specific color Gaussians the only subject of comparison. This means that Color Tanimoto has a much stricter basis for comparison than Shape Tanimoto, however, because overlapping color atoms of different types do not contribute to overlap – color types must be the same type within the same 3D space to be considered “overlapping”.^{34,39}

Openeye EON ESP (Electrostatic potential) Tanimoto is calculated similarly to ROCS Shape Tanimoto, and in fact needs to be calculated using the 3D-aligned ligands resulting from the Shape Tanimoto comparison as it does not perform a through-space alignment step. First, MMFF is used to assign charges through a bond-charge-increment-scheme (BCI), where each atom is defined based on its chemical environment, given a formal charge, and the charge of the bonds surrounding it to calculate the partial charge for each atom.⁴¹ The charge density of the ligand is then modelled as the sum of these partial atom charges, forming a smooth Gaussian function of charge density around the molecule.⁴² The electrostatic potential is then calculated using the Gaussian charge-density map using Zap (EON’s internal ESP solver), which calculates the Poisson-Boltzmann electrostatic potential for the molecule as a 3D grid around the molecule. The Poisson-Boltzmann equation simulates the condition in which the ligand is surrounded by a polar solvent (such as water), where there is a low-dielectric region in the ligand and a high-dielectric region immediately outside the molecule.⁴³ EON then solves for implicit salt modelling around ionic groups which gives them a more realistic charge distribution. These calculations together produce the electrostatic potential field for two molecules $A(r)$ and $B(r)$ from which the ESP Tanimoto is calculated using the field overlap integrals: $ESP\ Tan(A, B) =$

$\frac{\int A(r) \cdot B(r)}{\int A(r) \cdot A(r) + \int B(r) \cdot B(r) - \int A(r) \cdot B(r)}$ which is the same Tanimoto formula used throughout the ligand similarity metrics.^{42,43}

The Morgan ECFP4 (extended connectivity fingerprint 4) Tanimoto is perhaps the most straightforward to understand, as it depends on 2D, graph-based similarity and does not implement any 3D through-space overlap. Each atom in the ligand is assigned an initial bit-wise identifier which combines the atomic properties such as atomic number, valence shell, number of hydrogen and non-hydrogen neighbours^{44,45}. Then, in an iterative search process centred on each atom, the atomic neighbours in a radius of a single bond away are obtained and the atom's new identity is re-assigned as a combination of all the new information in bits. In an ECFP4 fingerprint, this iterative process is repeated twice to collect information from nearest neighbour atoms that are up to 2 bonds away.⁴⁴ All the collected atom identifiers are then collected and compressed into a fixed-length vector of 2048 bits which represents the molecule, its chemical composition, and its connectivity.^{44,45} These are then compared using the Tanimoto coefficient: $T_{A,B} = \frac{c}{a+b-c}$ where a is the number of “on” bits (populated with a 1 rather than 0) in one molecule and b is the number of “on” bits in the comparison molecule, and c is the number of overlapping “on” bits (the number of “on” bits in both molecules).³²

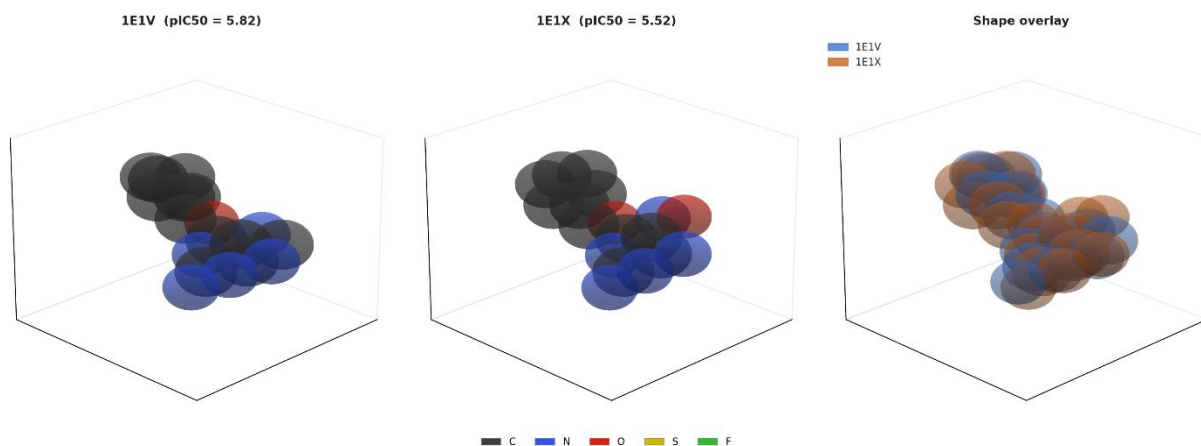


Figure 5. Image showing the Shape Gaussian distribution for two ligands within the CDK2 dataset, 1E1V (PDB ID: <https://doi.org/10.2210/pdb1E1V/pdb>), and 1E1X (PDB ID: <https://doi.org/10.2210/pdb1E1X/pdb>)⁶ based on the RDKit modelling of 3D atomic coordinates.¹¹

2.2.2 Ligand Similarity Metric Computations

Both CDK2 datasets (RCSB crystal poses and ChEMBL sample sets) ligand similarity metrics were calculated in largely the same way, the only distinction being that the ChEMBL set had no 3D coordinates and were conformationally-searched using OMEGA beforehand.^{25,28} The generated conformers for each ligand (≤ 200) were then all compared to maximize shape overlap, rather than comparing two singular native crystal poses for each ligand, as was done for the RCSB crystal-pose derived dataset. Similarly, at the end of the CDK2 RCSB crystal-posed dataset pipeline, the generated ligand files were saved in SDF (structural data format) format using RDKit, which is an accepted input format for ROCS which preserves the connectivity and 3D structure of the ligand, which was one of the focusses during the pipeline.¹⁵

To calculate all the ligand Tanimoto similarity metrics used within this work, Openeye ROCS, Openeye EON, and RDKit were implemented. First Openeye ROCS was used to compare the shape overlap of all the ligands in a pairwise manner.³⁴ In the crystal-posed dataset this was done with ligand pose fixed, as the ligands had already been aligned in space using MUSTANG and the goal was to examine the shape difference of their native crystallized binding pose. The ChEMBL sample sets were computed using the best overlay between all generated conformers of the ligand pair being examined. Both Shape and Color Tanimoto were parsed from the resulting output files generated by ROCS for each ligand pair, as ROCS internally performs this calculation (Figure 6).^{34,36} The aligned output OEB (Openeye Binary, proprietary 3D molecular storage format developed by Openeye) file is retained for each ligand pair, for downstream Openeye EON Tanimoto calculation.³⁴ Openeye ROCS is then run again in a self-overlay mode: overlaying each ligand (*i*) with itself. This serves no purpose except generating a necessary OEB file for comparison for the Openeye EON calculations, as OEB files encode specific connectivity and charge information used by EON.⁴² Openeye EON is then run, using

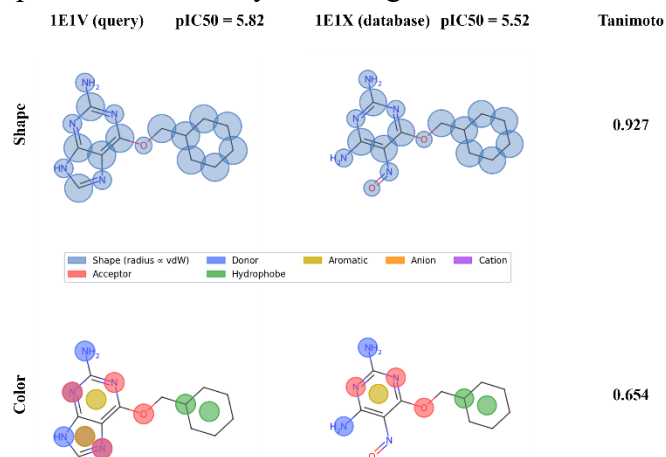


Figure 6. Artistic representation of Shape and Color Tanimoto coefficients as calculated using ROCS. Ligands are 1E1V (PDB ID: <https://doi.org/10.2210/pdb1E1V/pdb>) and 1E1X (PDB ID: <https://doi.org/10.2210/pdb1E1X/pdb>).⁶ The blue circles are modelled proportional to the van der Waals radii of their atoms, and colours correspond to pharmacophoric designations used by ROCS. Image created using RDKit.¹¹

the self-overlaid ligand as a query file (the molecule being compared to) and the output OEB file from each pairwise ROCS Shape and Color comparison (*i* vs *j*) as the database file (the molecule we are comparing). Because the (*i* vs *i*) comparison in EON is redundant, this is skipped in the pipeline, leaving the retained comparison pair the (*i* vs *j*) pair, for which

the electric field overlap Tanimotos are calculated by EON using the process described above.^{41–}
⁴³ Because the calculations are performed by Openeye EON, the output files are simply parsed for the electrostatic Tanimoto calculated using the Poisson-Boltzmann electrostatic field for each pairwise comparison.^{42,43} Finally, RDKit is used to compute Morgan ECFP4 using the SDF files output during the protein preparation pipeline, which contain key bonding information from the SMILES strings pulled from the RCSB database for each ligand described earlier.^{1,11,15} 2048-bit Morgan ECFP4 2D fingerprints are generated using the process described above (two iterations of single-bond radius scans), which are then compared pairwise using RDKit’s internal Tanimoto comparison formula – exactly the calculation described above.¹¹

2.2.3 Autodock Vina and GNINA Molecular Docking Methodology

In this work, molecular docking calculations are performed using Autodock Vina and GNINA, all using only the RCSB-obtained crystal pose dataset. Several different studies of the ligand-receptor complexes were conducted using various conditions: first, simply computing the native ligand pose “docking” score without changing the ligand at all; second, re-docking the ligands back into their native receptors; third, cross-docking the CDK2 ligands into the non-native receptors within the CDK2 ligand-receptor complex dataset, that is to say, all of the ligands are docked into all of the non-native receptors in the dataset. The rationale and specific docking strategy for each of these studies is discussed in more detail in Chapters 3 and 4, and will not be detailed here as it is slightly confusing without context.

To perform a molecular docking computation, the ligand’s docking pose must be identified. For the scenarios where the ligand-receptor is simply being scored, without any pose prediction, the molecular docking program simply scores the complex using specific built-in scoring functions, either empirical scoring functions (Vina) or convolutional neural network

(CNN) tensors (GNINA).^{46,47} In all other cases, however, this means posing the ligand must account for six degrees of pose freedom, three for position (x,y,z), and three for orientation (rotation around the x,y,z axes), as well as one additional degree of freedom for every rotatable bond present in the ligand.^{46,48} These are explored to find the “best” ligand pose using what is called an Iterated Local Search global optimizer. This samples ligand pose to find globally optimized docking pose through repeating three steps: mutation, where the ligand is randomly perturbed at its degrees of freedom; local optimization, where a Monte Carlo-type conformational search samples Vina’s scoring function from these perturbations to determine whether these are approaching a global minima (without getting stuck in a local minima); and acceptance, where a Metropolis score criterion accepts or rejects the new pose based on whether it provides a better score than the previous – if it does, it is accepted, and if it does not, it is sometimes accepted, especially if the pose is not significantly worse than the previous.^{46,48–50} This is highly important for the conformational search as the sampling of “worse” poses prevents the conformational search from getting stuck in local minima and thus failing to find the global minimum pose. This conformational sampling strategy is used for both Vina and GNINA to find the optimal ligand pose for molecular docking into a receptor.

Where Vina and GNINA diverge is their scoring functions. Vina’s scoring function estimates the interactions between the receptor and ligand score by summing every per-atom-pair interaction term across every pair of atoms between the ligand and the receptor, where each term is defined by what types of atoms are interacting, and how far apart they are⁵¹. Vina calculates the surface distance between ligand-receptor atom pairs by accounting for van der Waals radii, then accounting for atom-type interactions using five different terms: Gauss1, which is a Gaussian function which determines favourable short-range inter-atomic interactions; Gauss2,

which is a Gaussian function that determines favourable long-range inter-atomic interactions; repulsion, which penalizes steric clashing of inter-atomic interactions; hydrophobic, which rewards contact between hydrophobic atoms due to these being sequestered from solvated areas; and hydrogen-bonding term, which rewards contact between residues that can form hydrogen bonds – this term is non-directional, and determines bonding based on inter-atomic distance without accounting for bond angle.^{46,51} Finally, the Autodock Vina applies an "entropic" rotatable bonds penalty which is directly proportional to the number of rotatable bonds present in the ligand – this is why correctly determining the ligand connectivity is crucial. These scoring terms are empirical, meaning they have been weighted using linear regression analysis upon experimentally-derived protein-receptor biological activity data.^{46,51}

GNINA's scoring function is performed on the results of the Vina conformational ligand pose search, and its methodology is very different from Vina's empirically derived scoring functions. GNINA uses a deep-learning scoring function to score the ligand pose and binding through pattern recognition. First, the final ligand-receptor pose is divided into a 3D grid of "voxels" (a 3D version of pixels), where each voxel contains atom-type densities which can be summed to make a Gaussian function over the ligand (like how ROCS represents ligand shape).^{47,52,53} Deep learning CNN layers then learn patterns from the voxel grid which grow in complexity and can recognize ligand-receptor binding interactions like hydrogen-bonding through pattern-recognition. These are then translated into two outputs, the CNNscore, which is a score of the pose-binding probability, and the CNNAffinity, which is the predicted binding score using GNINA's scoring function.^{47,52} GNINA's CNN models were trained on many known ligand-receptor binding complexes and cross docking them, then having the CNN scoring functions predict whether the binding pose is correct to improve binding pose scoring. Similarly,

the CNN scoring functions were trained on DUD-E (Directory of Useful Decoys, Enhanced), which is a database of known active binding compounds and “decoy” molecules which resemble these active compounds - the scoring functions assign binding scores to these molecules, and learn to discriminate between binders and non-binders.^{47,54} All of this allows the GNINA scoring functions to evaluate binding score using a much different methodological approach, and the use of deep-learning scoring functions in molecular docking allows the study of the utility of these ML models in computational drug-design experiments.

2.2.4 CDK2 Crystal Posed Dataset Molecular Docking Computations

Vina and GNINA require that the user defines a “docking box” to perform any molecular docking calculations – this is the region of 3D space in which the ligand can explore with perturbations to its 3D structure, orientation, and rotation (Figure 7). The docking box prevents the computationally expensive alternative, which is that the ligand explores being docked to every part of the surface area of the protein – a much too large sampling space, and very prone to finding local energetic binding minima rather than global. The position and size of the docking box has been shown to affect the results of molecular docking studies, with too small boxes unable to sample chemical spaces and too large boxes finding unrealistic binding locations and poses. One study conducted by Feinstein et al. showed that the ideal box size was calculated to be $2.86 \times R_g$, the radius of gyration for the ligand.⁵⁵ Because of this, it made sense to rationally choose the box size and location for docking based on the MUSTANG alignment of the ligands and the ligand size and location within the CDK2 crystal-posed dataset. The ligand coordinates were gathered in an array and the minimum and maximum values for all three coordinate planes were recorded as a boundary condition for the box. The box centre was calculated at the midpoint of these values, and then a padding of 8Å was added in each dimension to define the

size of the box to not constrain the ligand conformational search. As all three of these dimensions were found to be relatively close to 30Å so the box size was rounded to be a 30x30x30Å box with a centre at the midpoint centre of the ligand dataset ($x = -1.79, y = 30.96, z = 9.00$). This single set of box coordinates was used as the global set of box coordinates through all the scoring and self-docking calculations in this work.

Using this box definition, the native ligand-receptor complexes were scored using the Autodock Vina scoring functions. The same calculation was then performed on the crystal complexes using the GNINA CNN scoring functions. Following these, the ligands were self-docked using Vina – that is to say, the ligands were removed from their native receptor and re-docked into the native receptor, allowing Vina to sample the binding pose. The pose and binding affinity search was performed with an exhaustiveness of 8, which means it performs the conformational search eight times starting from randomized ligand positions. This was set to output the top three binding modes discovered by Vina ($\text{num_modes} = 3$). The condition “rmsd = 2” was placed on the docking, which indicates that all output ligand poses must be $\geq 2\text{\AA}$ in RMSD from other to be retained, otherwise they are considered to be the same pose. The top scoring binding pose out of the three generated was then re-scored using GNINA’s CNN scoring functions. Finally, the RMSD values for the three Vina-generated binding poses from the self-docking was calculated and recorded along with the Vina-scored energies, for later evaluation of the effects of pose RMSD on the molecular docking scoring functions.

The ligands and receptors in the RCSB CDK2 crystal-posed database were also cross-docked (each ligand docked into every protein receptor) under similar central box coordinates ($x = -1.12, y = 30.54, z = 17.87$) but with a larger box size (x size = 56.49, y size = 44.17, z size = 60.50). This was done to ensure proper ligand pose search took place within the active site, and was calculated based on the locations of the KLIFS residues of the aligned receptors to ensure they were all within the docking boundaries. These were scored both using the Vina empirical scoring functions and the GNINA CNN scoring functions.

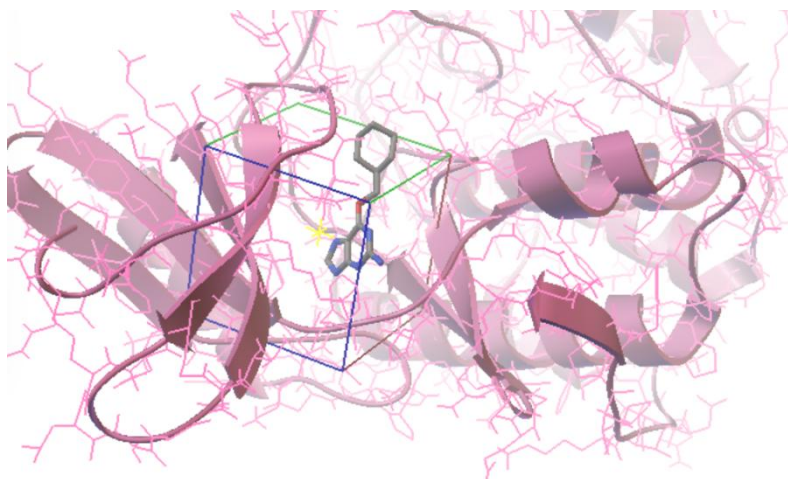


Figure 7. CDK2 ligand-receptor complex with active site docking-box, coordinates used for self-docking. Complex is RCSB protein-ligand 1E1V (PDB ID: <https://doi.org/10.2210/pdb1E1V/pdb>).⁶ Visualized using AutoDockTools.¹⁷

2.3 Statistical Methodologies

2.3.1 The Pearson r coefficient

The Pearson r reports the strength and direction of a linear relationship. Because one of the primary focuses of this study is to determine whether a trend can be established between ligand structure and biological activity, this is a useful statistical representation of the overall relationship between the two variables of interest.^{56,57} Pearson's r ranges from -1 to 1, with 1 representing a perfect positive linear relationship where as one variable increases, the other increases by a proportionate amount. A Pearson r value of 0 represents the case where no linear relationship exists at all between the two variables in question. A Pearson r value of -1 indicates a

perfect negative relationship between the two variables – as one variable increases, the other decreases by a proportional amount.⁵⁶ The Pearson’s r formula is represented by: $r = \frac{\sum((x_i - \bar{x})(y_i - \bar{y}))}{\sum\sqrt{\sum(x_i - \bar{x})^2} \cdot \sqrt{\sum(y_i - \bar{y})^2}}$ where the numerator calculates how far away a point (x,y) exists from the average, then multiplies these values together, and finally sums the series of values for the entire dataset. This means that points which are consistently above or below the average in both x and y contribute positively to the numerator, while points which are mixtures of above and below average in x and y contribute negatively.^{56,57} Thus if points consistently match the “above or below average in both”, the numerator will be large, and if the dataset displays mixed behaviour the summation of positives and negatives in the numerator will begin to approach zero. The denominator is a normalizing term which allows the Pearson r to be applied on datasets with different measures; it ensures that the value always appears between -1 and 1 regardless of the units of measurement or the specific variables being measured. It should be noted that Pearson r only captures linear relationships, and it does not describe the causation of a relationship, only the correlation.

2.3.2 The p-value test (with respect to Pearson r)

The p-value, calculated with respect to Pearson r values reported throughout this work, measures the probability that the same relationship between the two variables being measured is observed based on pure chance.⁵⁸ The p-value accounts for dataset size, which Pearson r does not directly account for. These are calculated using the formula: $t = r \sqrt{\frac{n-2}{1-r^2}}$, where n is the number of datapoints in the set and r is the Pearson r score for the raw correlation data.^{58,59} This equation shows that stronger correlations and larger datasets lead to a larger t value. The p-value is integrated from the t-distributions curve, and as t-values increase, p-values decrease, showing

that the probability of the correlation is very low for datasets with many datapoints and high correlation.^{58,60}

2.3.3 Kernel density estimation (KDE)

The kernel density estimation provides a method of estimating the underlying probability distribution of a dataset. Rather than displaying the distribution of data as a histogram, KDE allows the modelling of distribution as a smooth curve. In this way, the uncertainty associated with “binning” the data as must be performed to convert a 2D plot into a histogram is avoided, by graphing distribution as a smooth curve of the much smaller curves which correspond to the data points. KDE is calculated by the formula: $\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)$ where x is any point along the axis of interest where density is being measured, x_i are the data points on the axis, h is the bandwidth, and n is the number of points in the series.⁶¹ These are calculated by placing a small Gaussian curve (a kernel) upon each data point, then tuning the bandwidth (h) to minimize the average squared error between the estimated density and the true density if the true distribution were actually Gaussian.^{61–63} Areas of high datapoint clustering will have many kernels overlapping and reflect this density with a high peak in the overall density curve, as shown by the formula. The KDE analysis performed in this work was calculated using SciPy’s Gaussian KDE module with its built-in implementation of Scott’s rule.⁶⁴

2.3.4 The Fisher’s exact test

In Chapter 5, the Fisher’s exact test is implemented to determine whether the overrepresentation of α C-helix-out-A conformation pairs within the activity cliff population is statistically significant, and the two variables are correlated, or whether this relationship is observed due to pure chance. Fisher’s exact tests determine the probability that a statistical

relationship between two variables exists beyond the likelihood that the observed relationship would exist due to random chance.^{65,66} Data is sorted into a 2x2 contingency table:

	Activity Cliff	Not Activity Cliff	Row total	Where the possibility of all different combinations of these 4 assignments are assigned to the data points and the probability of each permutation actually
α C-helix- out-A	a	b	a + b	
NOT α C- helix-out-A	c	d	c + d	
Row total	a + c	b + d	n	

appearing within the data is calculated according to the formula: $P = \frac{\binom{a+b}{a}\binom{c+d}{c}}{\binom{n}{a+c}}$.^{20,65,66} To

simplify this, the equation calculates the number of ways to arrive at the exact observed table of results divided by the total number of ways to scatter the cliffs randomly. Small probability values ($p < 0.05$) indicate the observed relationship is very unlikely to be due to chance.⁶⁵ SciPy's Fisher exact function was used to perform this analysis.⁶⁴

2.3.5 The Fisher's Z test

The Fisher's Z test is used to compare two Pearson r values which belong to datasets that are unrelated and determines whether the two are significantly different from each other. Raw Pearson r values cannot be directly compared because their sampling distribution is skewed, especially around the bounds (-1 and 1).⁵⁸ The Fisher's Z transformation is used to stretch the bonds near ± 1 to an unbounded scale $(-\infty, \infty)$ with a normal distribution and known standard error, so comparison is possible.⁵⁸ It is calculated in three steps: first, transformation of Pearson r

to Fisher's z value, $z = \frac{\ln\left(\frac{1+r}{1-r}\right)}{2}$, which removes the bounds.^{67,68} Second, determining standard

error of difference, $SE = \sqrt{\frac{1}{n_1-3} + \frac{1}{n_2-3}}$, which combines the uncertainty from both samples into

one number.^{67,68} Finally, calculating the test statistic, $Z = \frac{z_1 - z_2}{SE}$ where Z can be referenced

against a standard normalization curve to give a p-value.^{67,68} Various SciPy and NumPy functions were used to perform these calculations.^{7,64}

2.3.6 The Steiger's Z test

The Steiger's Z test determines whether two Pearson r values from the same set are significantly different, specifically when one of the variables is the same. In the case of this study, this is the CDK2 crystal-posed dataset and the accompanying pIC50 values associated with the ligand-crystal receptor complexes.⁶⁹ First, a Fisher's Z transform is applied to the data for each of the Pearson r values: $z_x = \frac{\ln\left(\frac{1+r_x}{1-r_x}\right)}{2}$ in order to transform the relationships to a normal, unbounded distribution, as detailed in the previous section. Then, because the data is not independent, the correction factor k must be applied: $k = 1 - \frac{r_{12}^2 - r_{12} \cdot \bar{r} \cdot r_1 \cdot r_2}{2(1-r_1^2)(1-r_2^2)}$ where r is the average of r_1 and r_2 .^{67,69} This considers how correlated to the pIC50 both Pearson r values are, and how correlated to each other they are (r_{12}).^{67,69} Next, calculate uncertainty: $SE = \sqrt{\frac{2(1-r_{12}) \cdot k}{n-3}}$ as in the Fisher's z test.^{67,68} Finally, calculate the test statistic (Z) to determine the p-value: $Z = \frac{z_1 - z_2}{SE}$. A high |Z| value corresponds to a low p-value, as the p-value indicates the probability of calculating that |Z| value due to chance.^{67,68} Various SciPy and NumPy functions were used to perform these calculations.^{7,64}

Table 4. Software, modules, and databases used in RCSB crystal pose dataset, ChEMBL sample sets, and molecular docking, similarity metrics, and statistical analyses methods.

Method Category	Tool	Role	Method Category	Tool	Role
RCSB Prep	MUSTANG v3.2.4 ⁵	Aligns protein chain-A C α to reference (1E1V) for a common frame; transform re-derived for ligand via Kabsch.	RCSB Prep	KLIFS ⁷⁰	85-residue CDK2 pocket definition for chain selection and C α RMSD.
RCSB Prep	Dimorphite-DL ¹⁰	Protonates ligand SMILES at pH 7.4 via pKa-aware ionization.	RCSB Prep	PDBFixer ¹³	Adds missing heavy atoms, terminal caps, and H at pH 7.4.
RCSB Prep	OpenMM	H-only energy minimization (AMBER14, Langevin, heavy atoms frozen). ¹³	RCSB Prep	AMBER14 ¹⁴	Force field for the OpenMM H-relaxation step.
RCSB Prep	scikit-learn (GaussianMixture) ²⁰	Fits GMM (1–8 components) to α C RMSD; assigns clusters and probabilities.	RCSB Prep	Bayesian Information Criterion (BIC) ²¹	Selects optimal GMM component count (lowest BIC, n=1–8).
RCSB Prep	Meeko	Converts prepared PDB files to PDBQT files with the correct coordinates and connectivity	RCSB Prep	OpenBabel ³⁰	Fallback PDBQT/format conversion when the primary tool fails.
RCSB Prep	AutoDockTools / MGLTools 1.5.7 ¹⁷	Receptor PDBQT conversion; ligand PDBQT fallback if Meeko fails.	ChEMBL Sets	ChEMBL web resource client ²	Queries ChEMBL for CDK2 bioactivity data and ligand SMILES.
ChEMBL Sets	RDKit MolStandardize ¹¹	Standardizes ChEMBL SMILES: strips salts, normalizes, fixes charge/tautomer.	ChEMBL Sets	RDKit ETKDG conformer generator ²⁵	Generates an initial 3D conformer from 2D ChEMBL SMILES.
ChEMBL Sets	OEChem Python API / OEOmega (OpenEye Toolkits) ⁹	Single-conformer fallback when omega2 fails (e.g. macrocycles).	ChEMBL Sets	OpenEye OMEGA (omega2) ^{28,71}	Generates up to 200 diverse conformers per ligand for ROCS.
RCSB + ChEMBL	RCSB PDB ¹	Source of all raw crystal structures; PDB ID list for ChEMBL exclusion.	ChEMBL Sets	OpenBabel ³⁰	Fallback PDBQT/format conversion when the primary tool fails.
RCSB + ChEMBL	ProDy ⁴	Parses PDBs, atom/chain selections, coordinate transforms; computes C α RMSD.	RCSB + ChEMBL	RCSB Chemical Component Dictionary (CCD) ⁷²	SMILES source: bond-order template (RCSB pipeline) / exclusion check (ChEMBL).
RCSB + ChEMBL	NumPy ⁷	Coordinate arrays, Kabsch SVD, correlation/statistics calculations.	RCSB + ChEMBL	RDKit (core Chem module) ¹¹	Bond-order assignment, sanitization, charges (RCSB); SMILES parsing (ChEMBL).
Docking / Sim.	Mills-Dean color force field (SMARTS rules) ⁴⁰	SMARTS rules defining H-bond donor/acceptor/ring/hydrophobe features for Color Tanimoto.	Docking / Sim.	ROCS 3.5.1.1 ^{34,35,37}	Shape/Color Tanimoto via Gaussian overlap; rigid-body, gradient-optimized alignment.
Docking / Sim.	Zap (Poisson-Boltzmann electrostatics toolkit) ^{42,73}	EON’s internal PB solver; models charge density as atom-centered Gaussians.	Docking / Sim.	EON 2.4.0.1 ⁴²	ESP (electrostatic) Tanimoto via Poisson-Boltzmann; requires ROCS pre-alignment.
Docking / Sim.	RDKit Morgan/ECFP4 fingerprints ^{45,74}	Circular fingerprints (radius 2, 2048 bits); 2D graph-based similarity metric.	Docking / Sim.	Tanimoto coefficient (2D fingerprint similarity) ³²	Shared/total bit-overlap formula underlying Morgan fingerprint similarity.
Docking / Sim.	Metropolis acceptance criterion ⁵⁰	Acceptance rule in Vina/GNINA’s Monte Carlo search; escapes local minima.	Docking / Sim.	AutoDock Vina ^{46,51}	Monte Carlo/BFGS pose search + empirical scoring (steric, hydrophobic, H-bond).

Method Category	Tool	Role	Method Category	Tool	Role
Docking / Sim.	GNINA CNN architecture (3D voxel-grid CNN) ⁴⁷	3D CNN over voxelized atom-density grid; predicts pose quality + affinity using Vina pose	Statistics	t-distribution significance test ^{58,60}	Converts Pearson r to a t-statistic to obtain a p-value for correlation significance;
Statistics	Pearson correlation coefficient ⁵⁷	Strength/direction of linear relationships (similarity vs. Δ IC50, etc.).	Statistics	Fisher's Z test ^{58,68}	Compares the Pearson r scores of unrelated datasets and determines the significance of their relationship.
Statistics	Kernel Density Estimate (KDE) ^{61,63}	Represents the density of a dataset as a smooth density curve.	Statistics	Stieger's Z Test ^{67,69}	Compares the Pearson r of related datasets and determines significance of their relationship.
Statistics	Fisher's exact test ⁶⁵	Tests whether two categorical variables are statistically associated.			

"RCSB Crystal Pose Prep" refers to the crystal-posed receptor-ligand complex dataset, "ChEMBL Sample Sets" refers to ChEMBL-derived sample sets. Where a tool appears in both pipelines but performs a different role in each, it is listed in two separate cells. Where it appears in both preparation pipelines it is in the category "RCSB + ChEMBL". Modules used in molecular docking or ligand similarity metric calculations are categorized as "Docking/Sim", and statistical methods used in analysis are classified as "Statistics".

Sources

- (1) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. The Protein Data Bank. *Nucleic Acids Res.* **2000**, *28* (1), 235–242.
- (2) Zdrazil, B.; Felix, E.; Hunter, F.; Manners, E. J.; Blackshaw, J.; Corbett, S.; de Veij, M.; Ioannidis, H.; Lopez, D. M.; Mosquera, J. F.; Magarinos, M. P.; Bosc, N.; Arcila, R.; Kizilören, T.; Gaulton, A.; Bento, A. P.; Adasme, M. F.; Monecke, P.; Landrum, G. A.; Leach, A. R. The ChEMBL Database in 2023: A Drug Discovery Platform Spanning Multiple Bioactivity Data Types and Time Periods. *Nucleic Acids Res.* **2024**, *52* (D1), 1180–1192.
<https://doi.org/10.1093/nar/gkad1004>.
- (3) Van Linden, O. P. J.; Kooistra, A. J.; Leurs, R.; De Esch, I. J. P.; De Graaf, C. KLIFS: A Knowledge-Based Structural Database to Navigate Kinase-Ligand Interaction Space. *J. Med. Chem.* **2014**, *57* (2), 249–277. <https://doi.org/10.1021/jm400378w>.
- (4) Bakan, A.; Meireles, L. M.; Bahar, I. ProDy: Protein Dynamics Inferred from Theory and Experiments. *Bioinformatics* **2011**, *27* (11), 1575–1577.
<https://doi.org/10.1093/bioinformatics/btr168>.
- (5) Konagurthu, A. S.; Whisstock, J. C.; Stuckey, P. J.; Lesk, A. M. MUSTANG: A Multiple Structural Alignment Algorithm. *Proteins: Structure, Function and Genetics* **2006**, *64* (3), 559–574.
<https://doi.org/10.1002/prot.20921>.
- (6) Arris, C. E.; Boyle, F. T.; Calvert, A. H.; Curtin, N. J.; Endicott, J. A.; Garman, E. F.; Gibson, A. E.; Golding, B. T.; Grant, S.; Griffin, R. J.; Jewsbury, P.; Johnson, L. N.; Lawrie, A. M.; Newell, D. R.; Noble, M. E. M.; Sausville, E. A.; Schultz, R.; Yu, W. Identification of Novel Purine and Pyrimidine Cyclin-Dependent Kinase Inhibitors with Distinct Molecular Interactions and Tumor Cell Growth Inhibition Profiles. *J. Med. Chem.* **2000**, *43* (15), 2797–2804.
<https://doi.org/10.1021/jm990628o>.
- (7) Harris, C. R.; Millman, K. J.; van der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; Kern, R.; Picus, M.; Hoyer, S.; van Kerkwijk, M. H.; Brett, M.; Haldane, A.; del Río, J. F.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; Oliphant, T. E. Array Programming with NumPy. *Nature* **2020**, *585* (7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>.
- (8) Arun, K. S.; Huang, T. S.; Blostein, S. D. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Trans. Pattern Anal. Mach. Intell.* **1987**, *PAMI-9* (5), 698–700.
- (9) OpenEye Scientific Software, S. F. N. *OEToolkits (OpenEye Toolkits)*. <https://www.eyesopen.com>.
- (10) Ropp, P. J.; Kaminsky, J. C.; Yablonski, S.; Durrant, J. D. Dimorphite-DL: An Open-Source Program for Enumerating the Ionization States of Drug-like Small Molecules. *J. Cheminform.* **2019**, *11* (14), 1–8. <https://doi.org/10.1186/s13321-019-0336-9>.
- (11) Landrum, G.; Tosco, P.; Kelley, B.; Rodriguez, R.; Cosgrove, D. *RDKit: Open-source cheminformatics*. . <https://www.rdkit.org>.

- (12) Gasteiger, J.; Marsili, M. Iterative Partial Equalization Of Orbital Electronegativity - A Rapid Access to Atomic Charges. *Tetrahedron Lett.* **1980**, *36*, 3219–3288.
- (13) Eastman, P.; Swails, J.; Chodera, J. D.; McGibbon, R. T.; Zhao, Y.; Beauchamp, K. A.; Wang, L. P.; Simmonett, A. C.; Harrigan, M. P.; Stern, C. D.; Wiewiora, R. P.; Brooks, B. R.; Pande, V. S. OpenMM 7: Rapid Development of High Performance Algorithms for Molecular Dynamics. *PLoS Comput. Biol.* **2017**, *13* (7). <https://doi.org/10.1371/journal.pcbi.1005659>.
- (14) Maier, J. A.; Martinez, C.; Kasavajhala, K.; Wickstrom, L.; Hauser, K. E.; Simmerling, C. Ff14SB: Improving the Accuracy of Protein Side Chain and Backbone Parameters from Ff99SB. *J. Chem. Theory Comput.* **2015**, *11* (8), 3696–3713. <https://doi.org/10.1021/acs.jctc.5b00255>.
- (15) Greg Landrum. *The RDKit Book - Aromaticity*. https://www.rdkit.org/docs/RDKit_Book.html.
- (16) Santos-Martins, D.; He, Y.; Eberhardt, J.; Sharma, P.; Bruciaferri, N.; Holcomb, M.; Llanos, M. A.; Hansel-Harris, A.; Barkdull, A. P.; Tillack, A. F.; Bianco, G.; Paulsen, M. L.; Mato, J.; Taneja, I.; Forli, S. Meeko: Molecule Parametrization and Software Interoperability for Docking and Beyond. *J. Chem. Inf. Model.* **2025**, *65* (24), 13045–13050. <https://doi.org/10.1021/acs.jcim.5c02271>.
- (17) Morris, G. M.; Ruth, H.; Lindstrom, W.; Sanner, M. F.; Belew, R. K.; Goodsell, D. S.; Olson, A. J. Software News and Updates AutoDock4 and AutoDockTools4: Automated Docking with Selective Receptor Flexibility. *J. Comput. Chem.* **2009**, *30* (16), 2785–2791. <https://doi.org/10.1002/jcc.21256>.
- (18) Schrödinger, L. PyMOL - The PyMOL Molecular Graphics System.
- (19) Dempster, A. P.; Laird, N. M.; Rubin, D. B. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* **1977**, *1* (39), 1–22.
- (20) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Cournapeau, D.; Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Thirion, B.; Grisel, O.; Dubourg, V.; Passos, A.; VanderPlas, J.; Brucher, M.; Perrot, M.; Duchesnay, É. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.
- (21) Schwarz, G. Estimating the Dimension of a Model. *The Annals of Statistics* **1978**, *6* (2), 461–464.
- (22) Ultsch, M.; Lokker, N. A.; Godowski, P. J.; de Vos, A. M. Crystal Structure of the NK1 Fragment of Human Hepatocyte Growth Factor at 2.0 Å Resolution. *Structure* **1998**, *6* (11), 1383–1393.
- (23) Bettayeb, K.; Tirado, O. M.; Marionneau-Lambot, S.; Ferandin, Y.; Lozach, O.; Morris, J. C.; Mateo-Lozano, S.; Drueckes, P.; Schächtele, C.; Kubbutat, M. H. G.; Liger, F.; Marquet, B.; Joseph, B.; Echalié, A.; Endicott, J. A.; Notario, V.; Meijer, L. Meriolins, a New Class of Cell Death-Inducing Kinase Inhibitors with Enhanced Selectivity for Cyclin-Dependent Kinases. *Cancer Res.* **2007**, *67* (17), 8325–8334. <https://doi.org/10.1158/0008-5472.CAN-07-1826>.
- (24) Schonbrunn, E.; Betzi, S.; Alam, R.; Martin, M. P.; Becker, A.; Han, H.; Francis, R.; Chakrasali, R.; Jakkaraj, S.; Kazi, A.; Sebt, S. M.; Cubitt, C. L.; Gebhard, A. W.; Hazlehurst, L. A.; Tash, J.

- S.; Georg, G. I. Development of Highly Potent and Selective Diaminothiazole Inhibitors of Cyclin-Dependent Kinases. *J. Med. Chem.* **2013**, *56* (10), 3768–3782. <https://doi.org/10.1021/jm301234k>.
- (25) Riniker, S.; Landrum, G. A. Better Informed Distance Geometry: Using What We Know to Improve Conformation Generation. *J. Chem. Inf. Model.* **2015**, *55* (12), 2562–2574. <https://doi.org/10.1021/acs.jcim.5b00654>.
- (26) Rappe, A. K.; Casewit, C. J.; Colwell, K. S.; Goddard III, W. A.; Skiff, W. M. UFF, a Full Periodic Table Force Field for Molecular Mechanics and Molecular Dynamics Simulations. *J. Am. Chem. Soc.* **1992**, *114* (25), 10024–10035.
- (27) Hawkins, P. C. D.; Skillman, A. G.; Warren, G. L.; Ellingson, B. A.; Stahl, M. T. Conformer Generation with OMEGA: Algorithm and Validation Using High Quality Structures from the Protein Databank and Cambridge Structural Database. *J. Chem. Inf. Model.* **2010**, *50* (4), 572–584. <https://doi.org/10.1021/ci100031x>.
- (28) OpenEye, C. M. S. S. F. N. *OMEGA 4.2.1.1*. <http://www.eyesopen.com>.
- (29) *Open Babel, version 3.3.1*. <http://openbabel.org>.
- (30) O’Boyle, N. M.; Banck, M.; James, C. A.; Morley, C.; Vandermeersch, T.; Hutchison, G. R. Open Babel: An Open Chemical Toolbox. *J. Cheminform.* **2011**, *3* (10). <https://doi.org/10.1186/1758-2946-3-33>.
- (31) Bajusz, D.; Rácz, A.; Héberger, K. Why Is Tanimoto Index an Appropriate Choice for Fingerprint-Based Similarity Calculations? *J. Cheminform.* **2015**, *7* (1). <https://doi.org/10.1186/s13321-015-0069-3>.
- (32) Tanimoto, T. T. An Elementary Mathematical Theory of Classification and Prediction; International Business Machines Corporation: New York, 1958; pp 1–10.
- (33) Martin, Y. C.; Kofron, J. L.; Traphagen, L. M. Do Structurally Similar Molecules Have Similar Biological Activity? *J. Med. Chem.* **2002**, *45* (19), 4350–4358. <https://doi.org/10.1021/jm020155c>.
- (34) OpenEye, C. M. S. S. F. N. *ROCS 3.5.1.1*. <http://www.eyesopen.com>.
- (35) Hawkins, P. C. D.; Skillman, A. G.; Nicholls, A. Comparison of Shape-Matching and Docking as Virtual Screening Tools. *J. Med. Chem.* **2007**, *50* (1), 74–82. <https://doi.org/10.1021/jm0603365>.
- (36) Grant, J. A.; Pickup, B. T. A Gaussian Description of Molecular Shape. *Journal of Physical Chemistry A* **1995**, *99*, 3503–3510.
- (37) Grant, J. A.; Gallardo, M. A.; Pickup, B. T. A Fast Method of Molecular Shape Comparison: A Simple Application of a Gaussian Description of Molecular Shape. *J. Comput. Chem.* **1996**, *17* (14), 1653–1666.
- (38) Ehrt, C.; Krause, B.; Schmidt, R.; Ehmki, E. S. R.; Rarey, M. SMARTS.plus – A Toolbox for Chemical Pattern Design. *Mol. Inform.* **2020**, *39* (12), 1–4. <https://doi.org/10.1002/minf.202000216>.

- (39) Putta, S.; Landrum, G. A.; Penzotti, J. E. Conformation Mining: An Algorithm for Finding Biologically Relevant Conformations. *J. Med. Chem.* **2005**, *48* (9), 3313–3318. <https://doi.org/10.1021/jm049066l>.
- (40) Mills, J. E. J.; Dean, P. M. Three-Dimensional Hydrogen-Bond Geometry and Probability Information from a Crystal Survey. *J. Comput. Aided. Mol. Des.* **1996**, *10*, 607–622.
- (41) Halgren, T. A. Merck Molecular Force Field. I. Basis, Form, Scope, Parameterization, and Performance of MMFF94. *J. Comput. Chem.* **1995**, *17* (5 & 6), 490–519. [https://doi.org/10.1002/\(SICI\)1096-987X\(199604\)17:5/6<490::AID-JCC1>3.0.CO;2-P](https://doi.org/10.1002/(SICI)1096-987X(199604)17:5/6<490::AID-JCC1>3.0.CO;2-P).
- (42) OpenEye Scientific Software, S. F. N. *EON 2.4.0.1*. <http://www.eyesopen.com>.
- (43) Honig, B.; Nicholls, A. Classical Electrostatics in Biology and Chemistry. *Classical Electrostatics in Biology and Chemistry* **1995**, 268 (5214), 1144–1149.
- (44) Rogers, D.; Hahn, M. Extended-Connectivity Fingerprints. *J. Chem. Inf. Model.* **2010**, *50* (5), 742–754. <https://doi.org/10.1021/ci100050t>.
- (45) Morgan, H. L. The Generation of a Unique Machine Description for Chemical Structures. *J. Chem. Doc* **1965**, *5* (2), 107–113.
- (46) Trott, O.; Olson, A. J. AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading. *J. Comput. Chem.* **2010**, *31* (2), 455–461. <https://doi.org/10.1002/jcc.21334>.
- (47) McNutt, A. T.; Francoeur, P.; Aggarwal, R.; Masuda, T.; Meli, R.; Ragoza, M.; Sunseri, J.; Koes, D. R. GNINA 1.0: Molecular Docking with Deep Learning. *J. Cheminform.* **2021**, *13* (1). <https://doi.org/10.1186/s13321-021-00522-2>.
- (48) Eberhardt, J.; Santos-Martins, D.; Tillack, A. F.; Forli, S. AutoDock Vina 1.2.0: New Docking Methods, Expanded Force Field, and Python Bindings. *J. Chem. Inf. Model.* **2021**, *61* (8), 3891–3898. <https://doi.org/10.1021/acs.jcim.1c00203>.
- (49) Pillardy, J.; Arnautova, Y. A.; Czaplewski, C.; Gibson, K. D.; Scheraga, H. A. Conformation-Family Monte Carlo: A New Method for Crystal Structure Prediction. *PNAS* **2001**, *98* (22), 12351–12356.
- (50) Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; Teller, E. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **1953**, *21* (6), 1087–1092. <https://doi.org/10.1063/1.1699114>.
- (51) Quiroga, R.; Villarreal, M. A. Vinardo: A Scoring Function Based on Autodock Vina Improves Scoring, Docking, and Virtual Screening. *PLoS One* **2016**, *11* (5), 1–18. <https://doi.org/10.1371/journal.pone.0155183>.
- (52) Ragoza, M.; Hochuli, J.; Idrobo, E.; Sunseri, J.; Koes, D. R. Protein-Ligand Scoring with Convolutional Neural Networks. *J. Chem. Inf. Model.* **2017**, *57* (4), 942–957. <https://doi.org/10.1021/acs.jcim.6b00740>.

- (53) Hochuli, J.; Helbling, A.; Skaist, T.; Ragoza, M.; Koes, D. R. Visualizing Convolutional Neural Network Protein-Ligand Scoring. *J. Mol. Graph. Model.* **2018**, *84*, 96–108. <https://doi.org/10.1016/j.jmgm.2018.06.005>.
- (54) Francoeur, P. G.; Masuda, T.; Sunseri, J.; Jia, A.; Iovanisci, R. B.; Snyder, I.; Koes, D. R. Three-Dimensional Convolutional Neural Networks and a Crossdocked Data Set for Structure-Based Drug Design. *J. Chem. Inf. Model.* **2020**, *60* (9), 4200–4215. <https://doi.org/10.1021/acs.jcim.0c00411>.
- (55) Feinstein, W. P.; Brylinski, M. Calculating an Optimal Box Size for Ligand Docking and Virtual Screening against Experimental and Predicted Binding Pockets. *J. Cheminform.* **2015**, *7* (1). <https://doi.org/10.1186/s13321-015-0067-5>.
- (56) Dodge, Y.; Rousson, V. From Correlation to Direction Dependence Analysis 1888-2018. In *Direction Dependence in Statistical Modeling: Methods of Analysis*; Wiley, 2021; pp 1–7. <https://doi.org/10.1002/9781119523024.ch1>.
- (57) Pearson, K. Notes on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London* **1895**, No. 58, 240–242.
- (58) Fisher, R. A. Frequency Distribution of the Values of the Correlation Coefficient in Samples from an Indefinitely Large Population. *Biometrika* **1915**, *10* (4), 507–521.
- (59) Akoglu, H. User's Guide to Correlation Coefficients. *Turkish Journal of Emergency Medicine*. Emergency Medicine Association of Turkey. September 1, 2018, pp 91–93. <https://doi.org/10.1016/j.tjem.2018.08.001>.
- (60) Student [Gosset, W. S.]. The Probable Error of a Mean. *Biometrika* **1908**, *8* (1), 1–25.
- (61) Parzen, E. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics* **1962**, *33* (3).
- (62) Rosenblatt, M. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics* **1956**, *27* (3), 832–837. https://doi.org/10.1007/978-1-4419-8339-8_13.
- (63) Scott, D. W. Kernel Density Estimators. In *Multivariate Density Estimation: Theory, Practice, and Visualization*; 1992; pp 125–193.
- (64) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; van Mulbregt, P.; Vijaykumar, A.; Bardelli, A. Pietro; Rothberg, A.; Hilboll, A.; Kloeckner, A.; Scopatz, A.; Lee, A.; Rokem, A.; Woods, C. N.; Fulton, C.; Masson, C.; Häggström, C.; Fitzgerald, C.; Nicholson, D. A.; Hagen, D. R.; Pasechnik, D. V.; Olivetti, E.; Martin, E.; Wieser, E.; Silva, F.; Lenders, F.; Wilhelm, F.; Young, G.; Price, G. A.; Ingold, G. L.; Allen, G. E.; Lee, G. R.; Audren, H.; Probst, I.; Dietrich, J. P.; Silterra, J.; Webber, J. T.; Slavič, J.; Nothman, J.; Buchner, J.; Kulick, J.; Schönberger, J. L.; de Miranda Cardoso, J. V.; Reimer, J.; Harrington, J.; Rodríguez, J. L. C.; Nunez-Iglesias, J.;

- Kuczynski, J.; Tritz, K.; Thoma, M.; Newville, M.; Kümmerer, M.; Bolingbroke, M.; Tartre, M.; Pak, M.; Smith, N. J.; Nowaczyk, N.; Shebanov, N.; Pavlyk, O.; Brodtkorb, P. A.; Lee, P.; McGibbon, R. T.; Feldbauer, R.; Lewis, S.; Tygier, S.; Sievert, S.; Vigna, S.; Peterson, S.; More, S.; Pudlik, T.; Oshima, T.; Pingel, T. J.; Robitaille, T. P.; Spura, T.; Jones, T. R.; Cera, T.; Leslie, T.; Zito, T.; Krauss, T.; Upadhyay, U.; Halchenko, Y. O.; Vázquez-Baeza, Y. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17* (3), 261–272. <https://doi.org/10.1038/s41592-019-0686-2>.
- (65) Fisher, R. A. On the Interpretation of X^2 from Contingency Tables, and the Calculation of P. *Journal of the Royal Statistical Society* **1922**, *85* (1), 87–94.
- (66) Kim, H.-Y. Statistical Notes for Clinical Researchers: Chi-Squared Test and Fisher’s Exact Test. *Restor. Dent. Endod.* **2017**, *42* (2), 152–155. <https://doi.org/10.5395/rde.2017.42.2.152>.
- (67) Ming, X.-L.; Rosenthal, R.; Rubin, D. B. Comparing Correlated Correlation Coefficients. *Psychol. Bull.* **1992**, *111*, 172–175.
- (68) Cohen, J.; Cohen, P.; West, S. G.; Aiken, L. S. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*, 3rd ed.; Lawrence Erlbaum Associates, 2003.
- (69) Steiger, J. H. Tests for Comparing Elements of a Correlation Matrix. *Psychol. Bull.* **1980**, *87* (2), 245–251.
- (70) Van Linden, O. P. J.; Kooistra, A. J.; Leurs, R.; De Esch, I. J. P.; De Graaf, C. KLIFS: A Knowledge-Based Structural Database to Navigate Kinase-Ligand Interaction Space. *J. Med. Chem.* **2014**, *57* (2), 249–277. <https://doi.org/10.1021/jm400378w>.
- (71) Hawkins, P. C. D.; Skillman, A. G.; Warren, G. L.; Ellingson, B. A.; Stahl, M. T. Conformer Generation with OMEGA: Algorithm and Validation Using High Quality Structures from the Protein Databank and Cambridge Structural Database. *J. Chem. Inf. Model.* **2010**, *50* (4), 572–584. <https://doi.org/10.1021/ci100031x>.
- (72) Westbrook, J. D.; Shao, C.; Feng, Z.; Zhuravleva, M.; Velankar, S.; Young, J. The Chemical Component Dictionary: Complete Descriptions of Constituent Molecules in Experimentally Determined 3D Macromolecules in the Protein Data Bank. *Bioinformatics* **2015**, *31* (8), 1274–1278. <https://doi.org/10.1093/bioinformatics/btu789>.
- (73) Thorner, D. A.; Wild, D. J.; Willett, P.; Wright, P. M. *Calculation of Structural Similarity by the Alignment of Molecular Electrostatic Potentials*; 1998.
- (74) Rogers, D.; Hahn, M. Extended-Connectivity Fingerprints. *J. Chem. Inf. Model.* **2010**, *50* (5), 742–754. <https://doi.org/10.1021/ci100050t>.

Chapter 3 – Similarity Metrics

3.1 Analysis of Ligand-Based Similarity Metrics on a Crystal Pose Dataset

The idea that similar compounds have similar properties, especially biological activity, is a fundamental principle of medicinal chemistry.^{1,2} Many successful drug design experiments have discovered valuable new leads based on the premise that similarity-activity and structure-activity relationships exist and these relationships can be leveraged to propose novel ligands and predict drug binding events. However, when applying this concept to *in silico* VS for the development of new pharmacological leads, cheminformatics struggles to accurately describe, or capture, this principle. This failure is a result of several different problems which are common across the cheminformatics-VS field. Firstly, there are many ways to quantitatively define molecular similarity – chemical, structural, electronic.¹ Which one metric, or combination of metrics, provides an accurate definition? Secondly, it is a challenging task for computationally-calculated metrics to capture the nuance and multi-property chemistry that amalgamates into observed biological activity with approximations such as molecular docking scoring functions. Thirdly, it is not clear if these similarity metrics can be applied to chemical space at-large, or if they are only effective in capturing ligand similarity when applied to smaller subsets of drug scaffolds.² These issues are not easy ones to address in the world of VS, where there is often a divergence between quick, computationally cheap calculations for large drug libraries, and the ability to accurately reflect biological binding conditions. The field also faces the challenge of this data being streamlined into drug-discovery ML models, as these metrics provide useful information about ligand similarity and are cheap and fast calculations to perform. As such, this analysis provides interesting insight into the implications of using this data to train ML models, and the effects of using cheminformatics data to train these models.³

In this chapter, ligand similarity for CDK2 will be captured via a multi-metric analysis and compared to measured biological activity with the goal of establishing clear structure-activity correlations for CDK2 ligand-receptor binding.

3.1.1 Similarity Metric Tanimoto Comparison for Crystal-Posed Ligand Dataset

A dataset of 95 RCSB-deposited crystallized ligand-receptor pairs (RCSB.org) was used in this study to analyze the structure-activity landscape of CDK2 ligand binding.⁴ The RCSB database of human CDK2 ligands was filtered to select for ligand-receptor complexes which had accompanying pIC50 data. Using the native crystal pose of the ligand was integral to this analysis, as the purpose of the study was to compare ligand similarity to its biological activity upon binding – the native crystal pose eliminates 3D binding pose uncertainty that would otherwise serve as a source of noise in the similarity comparison due to a lack of established 3D coordinates.⁵ Because ligand structural similarity is thought to be directly related to its binding affinity and thus biological activity, comparing the native ligand poses allows the SAR landscape for CDK2 to be analyzed with much more confidence.⁵

Four similarity metrics were implemented and compared via Tanimoto coefficient in this study to analyze and account for multiple aspects of ligand similarity and its relationship with biological activity⁶. The Morgan Fingerprint (ECFP4) is a 2D ligand similarity metric which searches molecule structure in an expanding radius outward from each atom in the molecule to include the information about its two nearest neighbours, and converts the information about chemical environment and connectivity into bit-wise vectors.^{7,8} As such, it fails to give any information on ligand shape, bond angle, or conformation, hence its designation as a 2D metric. To explore steric and volumetric similarity of known CDK2 ligands, Openeye's ROCS Shape Tanimoto was used, which compares the volumetric overlap between two ligands in space.^{9,10}

The structure is represented by a series of Gaussian functions which are combined to create a three-dimensional spatial model of a ligand.^{10–12} Secondly, the Openeye ROCS Color Tanimoto of the ligands was calculated, which examines the pharmacophoric similarity of two ligands aligned in space.^{9,10} The ROCS force field defines 6 atom types (or colors) - hydrogen bond donors and acceptors, cations, anions, ring structures, and hydrophobic groups - and assigns atoms and chemical groups into these categories, then compares their agreement.^{9,13} Finally, the Openeye EON ESP Tanimoto was implemented to compare the electrostatics maps between two ligands.^{14,15} This metric measures electrostatic potential as a continuous function across the molecule, computing electron density as a 3D lattice across the molecule's surface.^{16,17}

The Tanimoto coefficient was applied as a quantitative measure of ligand similarity in this study, and is the cheminformatics standard metric for representing ligand similarity.¹⁸ The standard coefficient ranges from values of 0 to 1, with 1 representing perfect ligand similarity, and 0 representing no similarity present in a ligand comparison pair.⁶ The coefficient allows a numeric representation of similarity and thus can be leveraged to effectively establish and evaluate QSAR landscapes in conjunction with biological activity. As such, this is a measurable method which can be used to directly compare similarity metrics that typically examine significantly different ligand chemical, steric, pharmacophoric, and electrostatic properties. The Tanimoto coefficient for the previously listed metrics was calculated for all ligand pairs and compared to the $|\Delta pIC_{50}|$ belonging to each ligand pair to assess for the presence or absence of QSAR correlations.

The above metrics were applied to the RCSB crystal-posed CDK2 ligand dataset, which was grouped using RMSD as described in Chapter 2 into three distinct active site conformational groupings involving different α C-helix conformations: α C-helix out conformation A (α C-out-B,

18 ligands), α C-helix out conformation B (α C-out-B, 68 ligands), and α C-helix in (α C-in, 9 ligands). This was done to determine if grouping protein active site conformation showed increased selection for ligand similarity. The “Mixed” group in Figure 1 refers to data points which belong to a Tanimoto similarity comparison across conformational groups, e.g. an α C-helix in ligand compared to a α C-helix-out-A ligand.

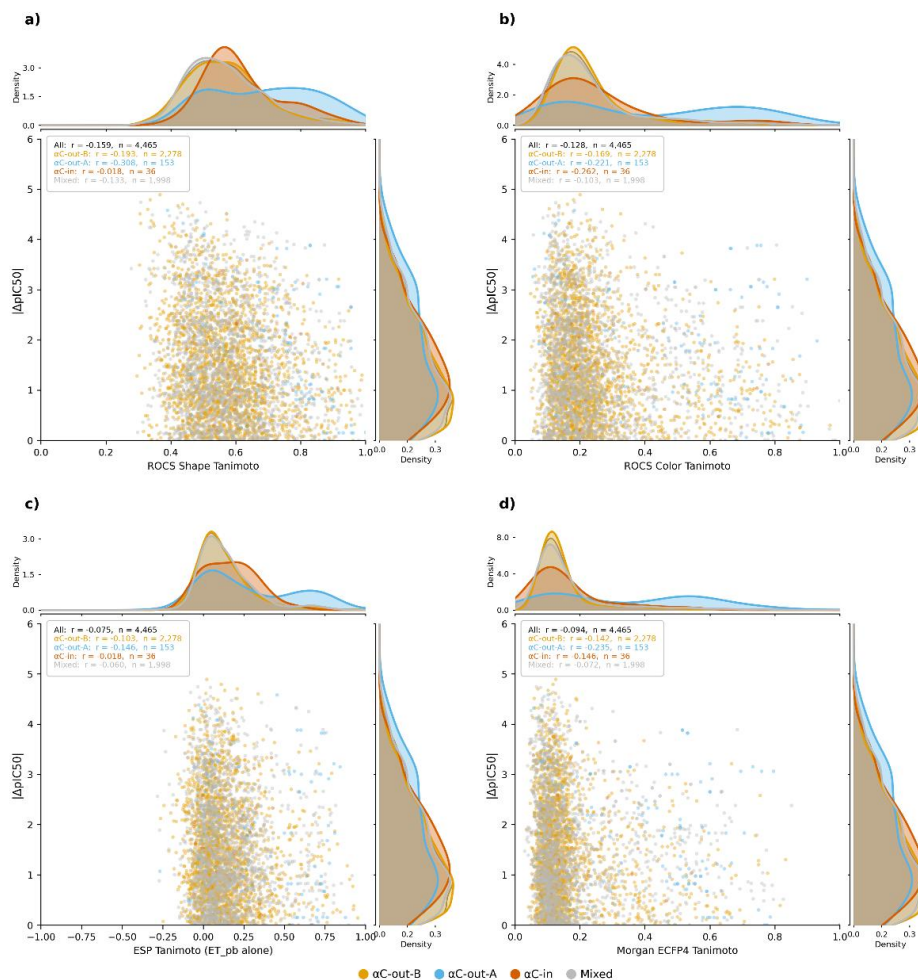


Figure 1. Scatter plots of pair-wise ligand Tanimoto similarity vs $|\Delta pIC_{50}|$ for a dataset of crystal-posed CDK2 ligands ($n = 95$, creating $n = 4,465$ unique comparisons). Data is sorted into 4 different groups: 3 different CDK2 active site conformations based on RMSD of the α C-helix: α C-out-B, α C-out-A, α C-in, and a “Mixed” group, which contains all pairs which compare ligands belonging to two different active site conformations. Marginal kernel density estimates (KDE) display the density distribution of similarity values (top) and activity values (right). (a) Openeye ROCS Shape Tanimoto vs $|\Delta pIC_{50}|$ for all ligands (b) Openeye ROCS Color Tanimoto vs $|\Delta pIC_{50}|$ for all ligands (c) Openeye EON Electrostatic Potential Tanimoto vs $|\Delta pIC_{50}|$ for all ligands. Note that negative ESP Tanimoto similarity values correspond to an in-space overlap of opposing charges. (d) Morgan Fingerprint ECFP4 Tanimoto vs $|\Delta pIC_{50}|$ for all ligands.

All four similarity metrics in Figure 1 give near-zero Pearson r correlation scores, indicating almost no correlation between ligand metric similarity and biological activity for the whole 95-ligand dataset. The weak negative correlations likely arise from closely related scaffold series, in which small structural modifications produce relatively small changes in potency.¹⁹ These localized structure–activity relationships contribute a modest inverse trend but are overwhelmed by the large number of unrelated ligand pairs, which also serves as an example of why this SAR relationship is largely expected to correlate across these metrics. Additionally, large Gaussian-shaped clusters corresponding to each pair dominate each graph and form large clusters at low similarity scores (with the exception of Shape Tanimoto) but wide ranges of $|\Delta pIC50|$, visually representing this statistic and showing little to no correlation between CDK2 ligand similarity and its biological activity.

Despite being unable to characterize a high correlation between similarity and activity within the full CDK2 dataset, there are several observations about CDK2 protein-ligand binding that can be made based on these plots. The Shape Tanimoto distribution (Figure 1a) is roughly unimodal and clustered around a similarity 0.5-0.6, indicating that the native ligand set is moderately shape-similar which makes sense as they all bind to the same active site. The Color Tanimoto is strongly left-distributed, with a clustering at 0.2 Tanimoto similarity (Figure 1b), indicating that while these ligands are somewhat similar in shape, they differ significantly in three-dimensional pharmacophore distribution. The electrostatic potential (ESP) Tanimoto also shows a cluster at a similarity of 0.25, indicating that electrostatic potential similarity across CDK2 ligands is low even in the native ligand pose (Figure 1c). Note that EON ESP Tanimoto does not ascribe to normal Tanimoto coefficient bounds (0 to 1)¹⁸ because it is a Tanimoto of the Poisson-Boltzmann electrostatic Tanimoto term, which can range from -0.3 to 1 due to ligands

aligned in space possessing opposing charges.¹⁶ While the ligand set has very low ESP similarity, it clusters around the origin, indicating that the ligands also do not have opposing charges at the same location in 3D space, which would be evidenced by observing clustering around a negative ESP similarity value. Finally, the Morgan ECFP4 Tanimoto distribution is similarly concentrated at low similarity values, indicating low two-dimensional structural similarity across much of the dataset (Figure 1d). The qualitative agreement between Morgan ECFP4 and Color Tanimoto suggests that many of the ligands are chemically diverse in both 2D connectivity and 3D pharmacophore arrangement, although this comparison alone does not establish whether native ligand pose information improves prediction of activity difference.

In Figure 1 it is shown that for most every metric, the Pearson r correlation coefficient improves by about 50% when separated into the conformational designations (not as statistically relevant as it sounds, given that the coefficient was negligible to begin with in most cases). Similarly, the correlation coefficient of the “Mixed” conformational grouping is lower than the correlation coefficient for the entire, uncategorized dataset across every metric. The kernel density estimation (KDE) distributions in the top marginals do show that α C-out-A (blue) and α C-out-B (dark orange) ligand sets tend to occupy different similarity regions. This can be particularly observed in the Shape Tanimoto similarity graph (Figure 1a), where the ligand α C-out-A similarity distribution is right-shifted, indicating high ligand shape or steric similarity. However, the $|\Delta pIC50|$ distribution (right margin) is essentially the same across groups, confirming that the conformational groups are no better at predicting biological activity than the complete dataset. Notably, the “Mixed” dataset (grey) clusters around low similarity values for all four metrics and has a lower distribution than all other groupings at higher similarity, which suggests that the active-site based conformational groupings do result in sub-categories of higher

similarity ligands. Again, however, this lower similarity does not show a different trend in $|\Delta pIC_{50}|$ than is observed in the within-conformation ligand pair datasets, indicating that stratifying ligands by their native active site conformation does not substantially strengthen the relationship between molecular similarity and biological activity.

3.1.2 Cutoff Similarity Metric Comparison for Crystal-Posed Ligand Dataset

Given that the analysis of the similarity metrics applied to the entire CDK2 dataset largely concluded that no similarity-activity correlations exist within the full set, it became

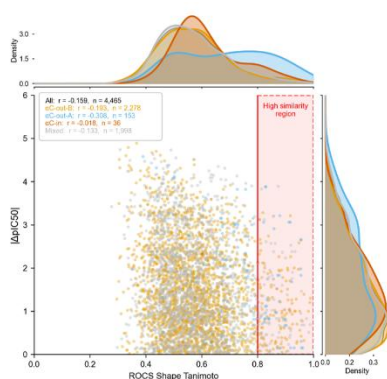


Figure 2. The threshold region used for examination of Shape Tanimoto similarity.

necessary to narrow down the statistical analysis. Based on the principle that similar ligands display similar properties and the work performed by Vogt et al., the dataset plots were filtered for regions of high-similarity, low-activity difference ligand pairs.^{1,20} The rationale was that it should be expected that low-similarity ligands show high variability in potency differences due to these ligands having different ligand properties, and likely different ligand binding modes, in the

case of flexible receptor binding as in CDK2.²¹ The large cluster of low-similarity ligand pairs could possibly be serving no purpose beyond adding noise to the dataset, and thus thresholds were applied to the dataset to examine whether it is possible to establish trends among the highest-similarity, lowest-biological activity difference pairs. The similarity thresholds used were selected for the purpose of excluding the large cluster of low-similarity ligands that appears in the dataset, and thus were selected for visually based on the results shown in Figure 1 (Shape Tanimoto ≥ 0.8 , Color Tanimoto ≥ 0.4 , ESP Tanimoto ≥ 0.5 , Morgan Tanimoto ≥ 0.3) (Figure 2). These were selected for visually as the clusters were very evident and simple to exclude upon

visualization. The exclusion of the clusters meant the retention of only high Tanimoto similarity pairs, which are the pairs of interest when establishing SAR within a dataset.

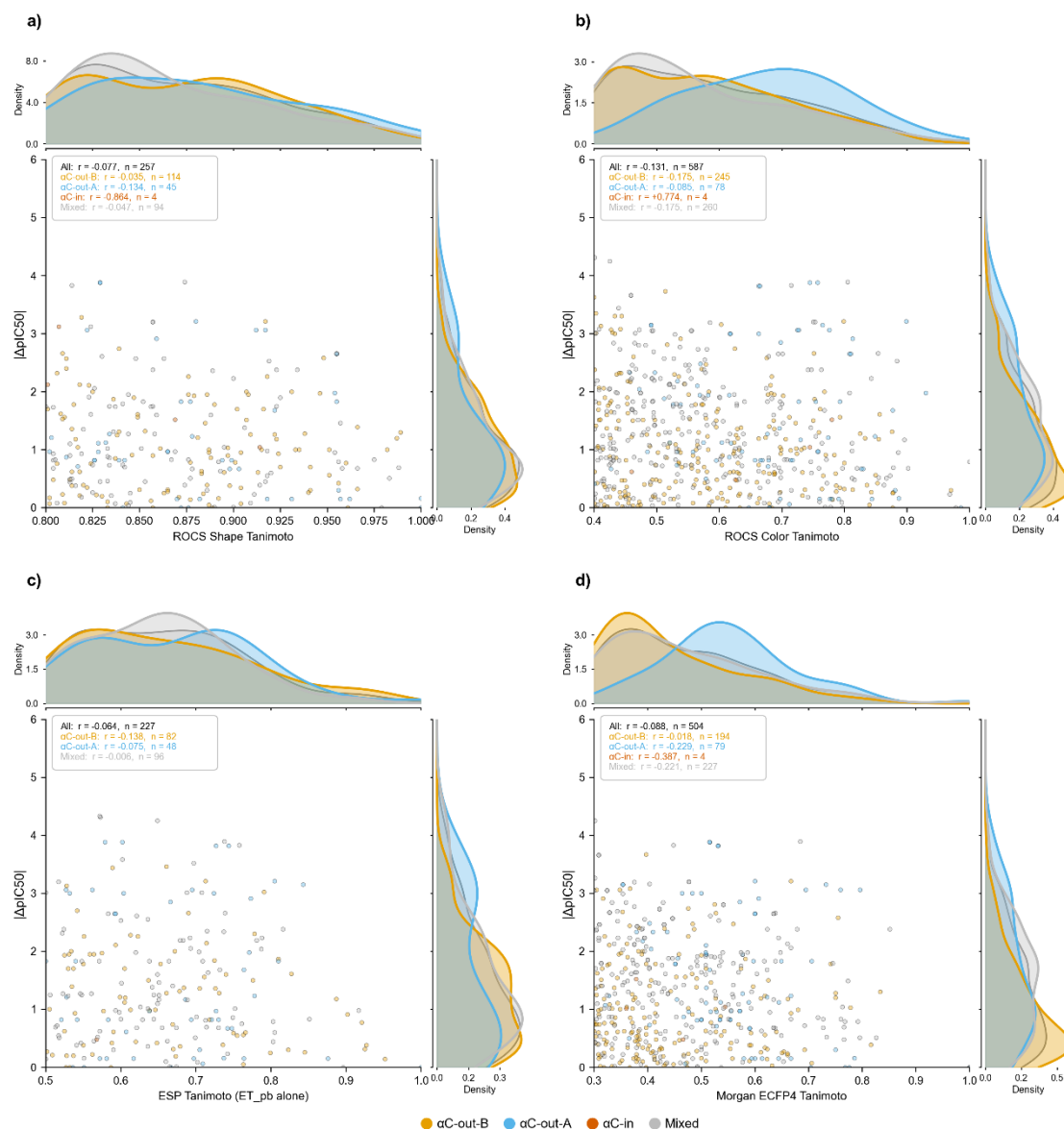


Figure 3. Scatter plots of pair-wise ligand Tanimoto similarity vs $|\Delta pIC_{50}|$ for a dataset of crystal-posed CDK2 ligands ($n = 95$, creating $n = 4,465$ unique comparisons). Data is sorted into 4 different groups: 3 different CDK2 active site conformations based on RMSD of the αC -helix: αC -out-B, αC -out-A, αC -in, and a "Mixed" group, which contains all pairs which compare ligands belonging to two different active site conformations. Marginal kernel density estimates (KDE) display the density distribution of similarity values (top) and activity values (right). A 3.0 $|\Delta pIC_{50}|$ maximum cutoff was applied to all plots to include only low potency difference ligands. (a) Openeye ROCS Shape Tanimoto vs $|\Delta pIC_{50}|$ for ligands above a similarity threshold cutoff of ≥ 0.8 and $|\Delta pIC_{50}|$ of ≤ 3.0 (b) Openeye ROCS Color Tanimoto vs $|\Delta pIC_{50}|$ for ligands above a similarity threshold cutoff of ≥ 0.4 and $|\Delta pIC_{50}|$ of ≤ 3.0 (c) Openeye EON Electrostatic Potential Tanimoto vs $|\Delta pIC_{50}|$ for ligands above a similarity threshold cutoff of ≥ 0.5 and $|\Delta pIC_{50}|$ of ≤ 3.0 . (d) Morgan Fingerprint ECFP4 Tanimoto vs $|\Delta pIC_{50}|$ for ligands above a similarity threshold cutoff of ≥ 0.3 and $|\Delta pIC_{50}|$ of ≤ 3.0 .

It is interesting that selecting for these regions does not display any significant improvement in similarity-activity correlation. The expected outcome of this analysis is that removing the low-similarity, variable activity pairs would allow for the high similarity, low potency difference pairs to dominate and produce correlation, given that the existence of similarity-activity relationships for CDK2 scaffolds have been observed on several occasions in literature^{19,22}. The results show that filtering the dataset results provides only a slight

Table 1. Pearson r correlation between molecular similarity metrics and $|\Delta pIC50|$ for the full pairwise dataset and the high-similarity subset.

Metric	Sim threshold	All pairs r	All pairs n	High-sim r	High-sim n
ROCS Shape	≥ 0.80	-0.159	4,465	-0.077	257
ROCS Color	≥ 0.40	-0.128	4,465	-0.131	587
EON ESP	≥ 0.0	-0.075	4,465	-0.064	227
Morgan ECFP4	≥ 0.30	-0.094	4,465	-0.088	504

High-sim n = number of pairs meeting the similarity threshold. Pearson r computed for the entire dataset (all pairs r) and the cutoff subsets (high sim n). All values near zero confirm that no similarity-potency relationship exists, whether pairs are restricted to the high-similarity region.

improvement when you restrict the data to only highly similar pairs (Table 1). In fact, the Pearson r coefficient shows decrease in the similarity-activity correlation for three out of four metrics when the cutoff filters are applied.

The filtering application displays the difference in similarity distribution for α C-out-A conformational group in comparison with the other groups even more prominently, however (Figure 3). The marginal similarity KDE displays that

α C-out-A almost always has a higher distribution of data points at higher similarity scores even without considering data points which correspond to high potency differences. This indicates that overall this conformational grouping tends to have higher structural similarity over all metrics. However, the $|\Delta pIC50|$ distribution of this group appeared to be the same as all other conformational groupings, destroying any hope of correlation. It should be noted that a very low

number of α C-in pairs are included beyond the threshold cutoffs and thus any useful observations to be made about them are likely to be influenced by the small dataset size. Based on the statistics shown in the high similarity, low potency region of the dataset graphs, these results show that CDK2 similarity shows a negligible correlation with biological activity even among the highest similarity binders.

3.2 Analysis of Ligand-Based Similarity Metrics on a ChEMBL dataset

Given that no similarity-activity pose could be observed within the crystal-pose dataset, it made sense to next explore the effects of native ligand pose on similarity by examining the null hypothesis – if native ligand pose is unknown, will this show a degradation in correlation beyond what is already observed? Many VS computational studies assert that native ligand and receptor pose is the gold standard for drug screening, as these provide a template upon which a novel lead compound binding event can be referenced to.^{5,23} As such, it would be expected that not knowing the native binding pose of the ligand would significantly reduce the similarity-activity correlation beyond what was observed in the RCSB-obtained CDK2 ligand database which was previously explored. Second, the SAR landscape of these datasets would provide a better representation of the conditions of a VS screen, making this analysis more relevant when considering the use of similarity metrics in screening for potential lead compounds in a drug design experiment.¹¹

3.2.1 Similarity Metric Tanimoto Comparison for ChEMBL Ligand Database

Three independent sample sets of 95 ChEMBL-deposited CDK2 ligands were used in this study to analyze the structure-activity landscape of CDK2 ligand binding with an unknown ligand pose.²⁴ The ChEMBL database of human CDK2 ligands was filtered to include ligands which had accompanying activity data, either pKi values or pIC50 data.²⁵ Because the goal of the study was to determine the effects of knowing the native ligand pose on similarity-activity

correlation for CDK2 binding, any ligands which appeared in the RCSB were filtered out of the dataset before randomly choosing 3 sets of 95 ligands (set 1, set 2, set 3). A conformational search was applied to the ligands using Openeye OMEGA, and the top 200 ligand conformations for each ligand were overlaid.^{26,27} The previously described ROCS Shape and Color Tanimoto and EON ESP Tanimoto were computed on the best-scoring conformer overlay across all comparisons of each ligand conformer.^{9,14} These Tanimoto pairs were compared against the $|\Delta \text{ activity data}|$ corresponding to that pair. It should be noted that this three-set approach was applied to two different activity metrics, both pKi and pIC50, to examine whether using a more robust method of activity assay (pKi) improved correlation results. Given that the pKi results largely agreed with those of the pIC50 datasets, only the pIC50 will be displayed here, but the pKi sets can be found in the Supporting Information.

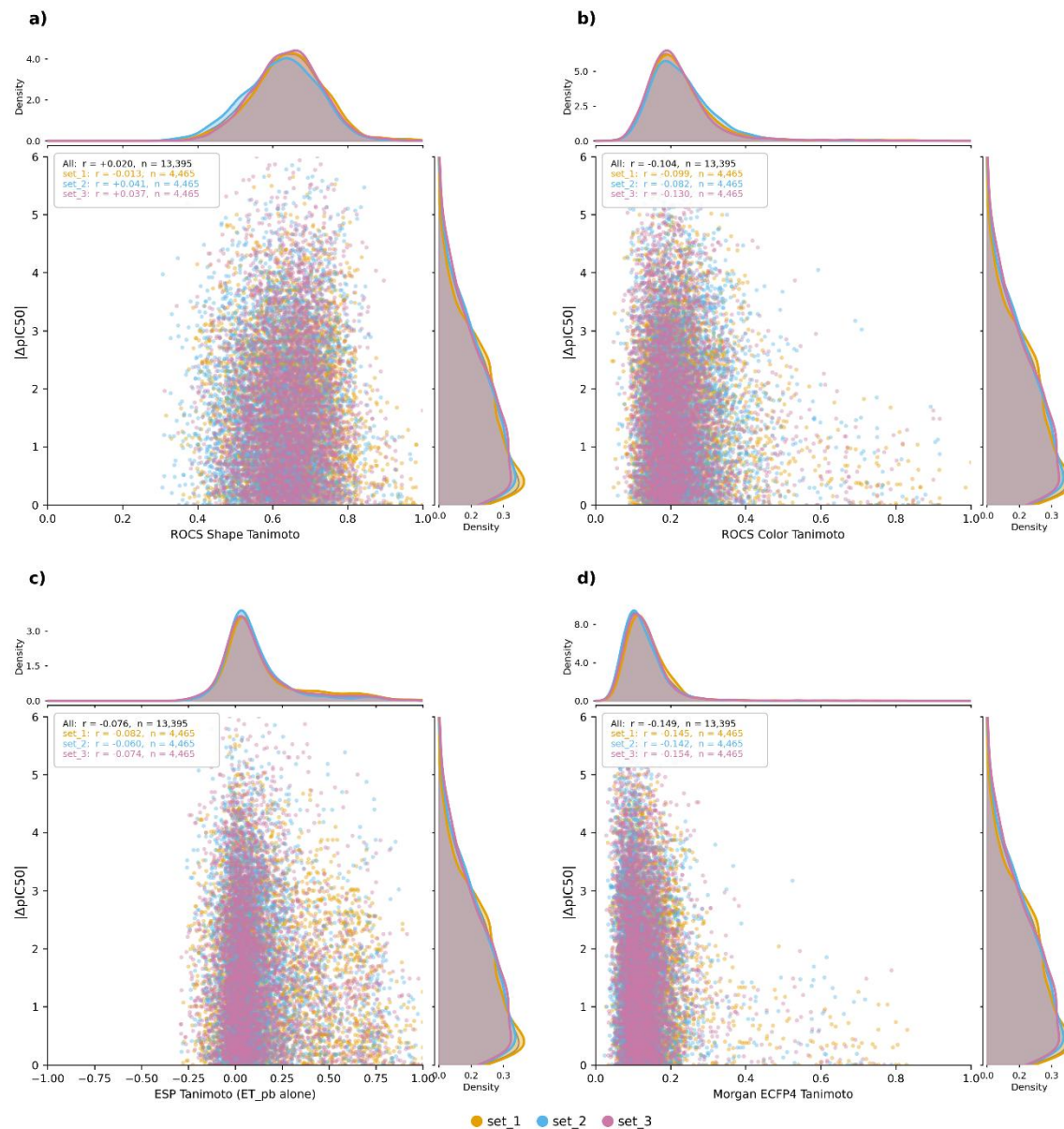


Figure 4. Scatter plots of pair-wise ligand Tanimoto similarity vs $|\Delta pIC_{50}|$ for 3 datasets of ChEMBL CDK2 ligands ($n = 95$, creating $n = 4,465$ unique comparisons per set). Ligand poses were generated from SMILES strings using RDKit and conformationally sampled using Openeye OMEGA for top ligand conformations, then overlaid using ROCS to find a maximum structural overlap. Marginal kernel density estimates (KDE) display the density distribution of similarity values (top) and activity values (right). (a) Openeye ROCS Shape Tanimoto vs $|\Delta pIC_{50}|$ for all ligands (b) Openeye ROCS Color Tanimoto vs $|\Delta pIC_{50}|$ for all ligands (c) Openeye EON Electrostatic Potential Tanimoto vs $|\Delta pIC_{50}|$ for all ligands. Note that negative ESP Tanimoto similarity values correspond to an in-space overlap of opposing charges. (d) Morgan Fingerprint ECFP4 Tanimoto vs $|\Delta pIC_{50}|$ for all ligands.

Between the three random ChEMBL-obtained datasets there is almost no variability in distribution that can be observed. All three sets look essentially identical within their KDE

distribution and have Pearson r values within ± 0.01 of each other (Figure 4). The nearly identical similarity distributions and Pearson correlation coefficients across the three independently sampled ligand sets demonstrate that the observed trends are not artifacts of a

Table 2. Pearson r correlation comparison across ChEMBL and Crystal-Posed CDK2 Sets

Metric	Crystal Pearson r	ChEMBL Pearson r	Difference
ROCS Shape	-0.159	+0.020	-0.179
ROCS Color	-0.128	-0.104	-0.024
EON ESP	-0.075	-0.076	0.000
Morgan ECFP4	-0.094	-0.149	+0.037
Mean	-0.119	-0.078	-0.041

Pearson r for crystal sets calculated upon entire obtained RCSB dataset (all conformations). Pearson r for ChEMBL sets calculated upon entire dataset (sets 1, 2, 3)

particular ligand selection but are reproducible across CDK2 chemical space, However, even more interesting is the comparison that can be made between the ChEMBL data and the RCSB crystal-posed dataset (Figure 1).

The Pearson r coefficients for similarity-activity comparison across all metrics are practically identical for both tested

sets, the ChEMBL-generated ligands and the crystal-posed ligands (Table 2). Similarly, the KDE distribution in both top and right margins for all four graphs is distributed with high similarity in Figures 1 and 4, with the exception of a few structural series in the crystal dataset. This is a strong indication that there is no meaningful increase in similarity-based potency prediction when the precise 3D binding pose of the ligand is known. It is worth noting that the Shape Tanimoto (Figure 4a) shows a more uniform similarity distribution in the top margin compared to the crystal set – this is likely due to the conformational alignment step, which allowed ligands to achieve maximal 3D overlap rather than being constrained to their native binding pose. Within the context of ligand-based similarity analysis, conformationally sampled ligand overlays provide relationships with biological activity that are comparable to those obtained from experimentally-determined crystal poses. Within the ligand-based similarity metrics examined

here, knowledge of the receptor-bound crystal pose provides little improvement in predicting activity differences relative to conformationally sampled ligand overlays.

3.3 Scoring-Function Analysis of Ligand Binding on a Crystal Pose Dataset

Having established that ligand-based similarity metrics fail to predict CDK2 potency differences regardless of whether detailed native pose information was available, the next step was to explore whether modelling protein-ligand interactions via molecular docking would be able to accurately predict differences in biological activity. While similarity metrics capture only ligand-based properties, molecular docking scoring functions evaluate many aspects of the protein-ligand interaction complex and weight these, to approximate the energetic favourability of protein-ligand interactions using empirical-based or machine-learning-based scoring functions.^{28,29} This made it a logical cheminformatics tool to leverage when trying to establish QSAR relationships within the CDK2 ligand-receptor dataset, as it may be able capture some binding properties which are missed in a ligand-focused study. Additionally, molecular docking is one of the most prolific VS early screening tools used for drug discovery campaigns, given that it is fast, computationally efficient, and has been shown to recover ligand native binding poses with some degree of success, as well as accurately rank potential ligand binders as top scorers against drug targets for further examination.^{30,31} As such, its ability to score ligand binders and recover native ligand poses should be evaluated – if these scoring functions fail to correctly handle native co-crystallized ligand complexes, how can one trust their success in ranking compounds for which the native ligand binding pose within the protein is completely unknown?

For this study, two different molecular docking platforms were used. Autodock Vina is a molecular docking platform which uses an empirical-based scoring function to predict pose and

score binding.^{32,33} It does this through scoring-function guided sampling of the ligand conformation within the active site, with weighted terms which evaluate through-space protein-ligand interactions between specific ligands and assign them categories such as hydrophobic, hydrogen-bonding, and ionic interactions.^{32,34,35} The molecular docking software GNINA was also implemented, which uses a convolutional neural network approach to score ligand binding.³⁶ The CNN-scoring method divides the ligand and protein active site into a 3D grid, which is then scored by CNN scoring functions using image pattern recognition that has been trained on many known ligand-receptor crystal complexes.³⁶ The GNINA CNN scoring function uses the Vina conformational search methodology to sample docking conformation, which is guided by the Vina scoring function, so this method is somewhat dependent on the native pose-recovery ability of the Vina molecular docking software in its accuracy.³⁶ Note that the binding scores obtained using Vina scoring functions are increasingly negative for strong binders to reflect “binding affinity” calculations, while GNINA’s binding scores are increasingly positive for strong binders as it scores contributing binding interactions as positive for score simplicity. As such, for the sake of comparison the $|\text{Pearson } r|$ will be studied within this chapter which will indicate the strength of the relationship without adding the confusion of opposite signs.

3.3.1 Molecular Docking Score and Activity Comparison for RCSB Ligands

The capabilities of VS molecular docking softwares were examined using the 95-ligand RCSB crystal pose database previously described, and the binding scores compared with both accompanying pIC50 activity, and $|\Delta\text{pIC50}|$ data. The goal was to evaluate their ability to directly predict activity based on a native ligand-receptor complex, as well as their ability to detect small changes in ligand similarity which lead to large differences in biological activity.

To do this, several analyses were conducted (Figure 5) – first, the ligand-protein interactions were scored using Autodock Vina using the native ligand receptor complex, to evaluate the molecular docking software’s ability to rank the bound ligands in a manner reflective of existing activity data. These native complexes were then re-scored using the GNINA scoring function to evaluate the ability of its scoring functions to rank the bound ligand-receptor complexes in correlation with their activity data. Then the ligands were removed from their receptor and docked back into the native receptor (self-docking), to determine whether the docking software was able to accurately score the ligand binding as well as recover the native ligand pose. The pose produced using Vina self-docking conformational search and scoring was then re-scored using GNINA to examine whether the scoring function was able to correlate binding score with biological activity on the Vina-optimized binding pose.

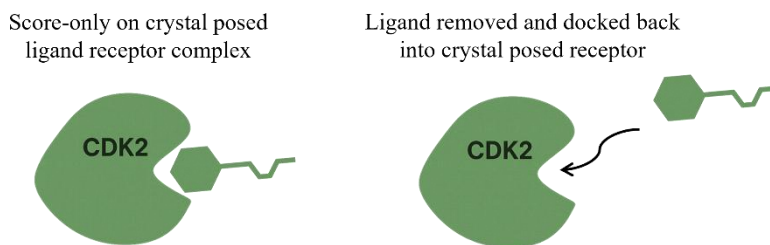


Figure 5. Molecular docking analysis protocols performed on CDK2 ligand-protein complexes. These protocols were performed using both the Autodock Vina and GNINA molecular docking programs.

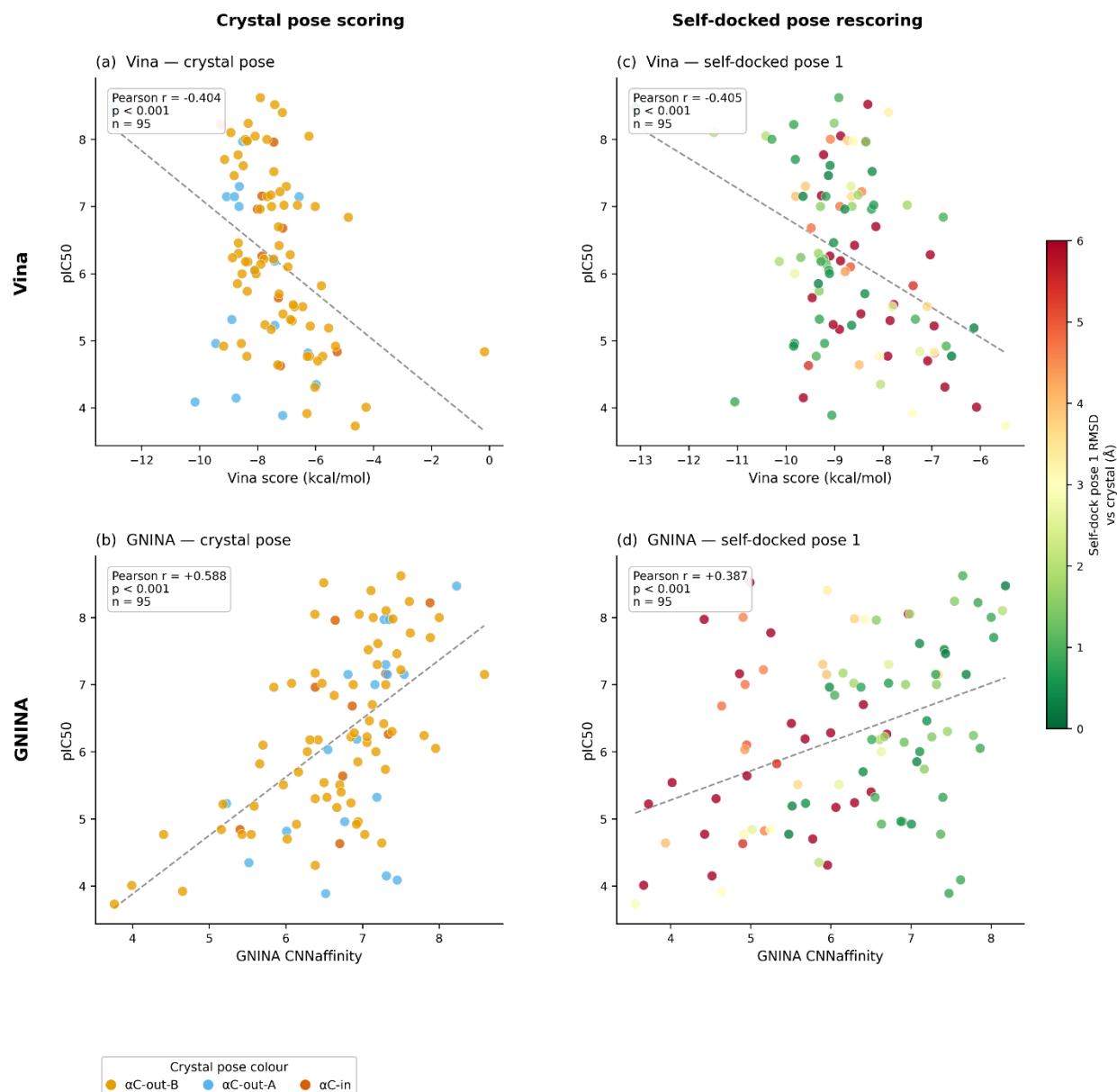


Figure 6. Results of Autodock Vina and GNINA scoring and self-docking on the 95-ligand-receptor complex RCSB crystal posed database. Docking scores are compared directly to pIC50 activity values for each co-crystallized pair. a) Vina score-only applied to the crystal complexes to obtain predicted binding affinity (kcal/mol) on native pose. Datapoint colour corresponds to conformational groupings. b) Vina self-docking of ligand into receptor applied to dataset to obtain predicted binding affinity (kcal/mol). Colours correspond to the RMSD difference between the Vina top scoring pose and the native ligand pose. c) GNINA score-only applied to the crystal complexes to obtain predicted binding score for native pose. Datapoint colour corresponds to conformational groupings. d) GNINA re-score of Vina-posed self-docking of ligand into receptor applied to dataset to obtain predicted binding score. Colours correspond to the RMSD difference between the Vina top scoring pose and the native ligand pose.

Several conclusions can be drawn from the results of this study. First, based on the Pearson r coefficients, GNINA significantly outperforms Vina in predicting the binding score of

co-crystallized ligand structures, with a comparison $|r|$ value of 0.404 to GNINA's 0.588 (Figure 6a, 6c). This GNINA score is by far the highest observed Pearson r correlation between any cheminformatic metric examined in this study and activity, and suggests that GNINA CNN scoring is able to somewhat accurately predict CDK2 ligand potency when provided with the native ligand-receptor pose upon which to score. It should be noted that GNINA was trained on the PDBBind database which does contain many CDK2 crystal complexes³⁶, and thus this result could be an artefact of the training set. Nonetheless, the ability of the molecular docking neural network to potentially reproduce the data upon which it was trained indicates this could be a valuable tool to implement in VS early screening experiments which use CDK2 as a target. In comparison, Vina cannot relate its predicted score to observed CDK2 ligand biological activity with nearly much accuracy. Interestingly, a trend between docking score and pIC50 can only be observed within the GNINA-scored crystal pose α C-out-B subgroup (Figure 6b).

Upon examination of the Vina self-docking results, it is evident that the removal of the ligand from the active site does not affect the $|r|$ in any meaningful way (Figure 6a, 6b). In fact, the difference between the two correlations is -0.001, which is surprising given that allowing the ligand to freely move and sample poses should introduce a lot of noise to the dataset and thus reduce correlation. Upon first glance, this suggests that the Vina algorithm is likely re-capturing the native ligand poses with high agreeability, leading to a very small difference in correlation between the Vina binding score calculated for the native crystal complex versus the self-docked pose binding score. However, the RMSD analysis of the top-scoring pose vs the native ligand shows that this is not the case – many of the ligands have an RMSD of $\geq 5\text{\AA}$ and these are distributed fairly evenly across the dataset, not limited to poor-scoring ligands close to the origin. This indicates that Vina's empirical scoring function is likely missing some key aspects of CDK2

ligand binding, and could be over-weighting some local atomic interactions while failing to realize that the ligand geometry is not optimal for binding. In fact, when the RMSD of all Vina top-scoring ligand poses was compared to the native pose, the results were surprising:

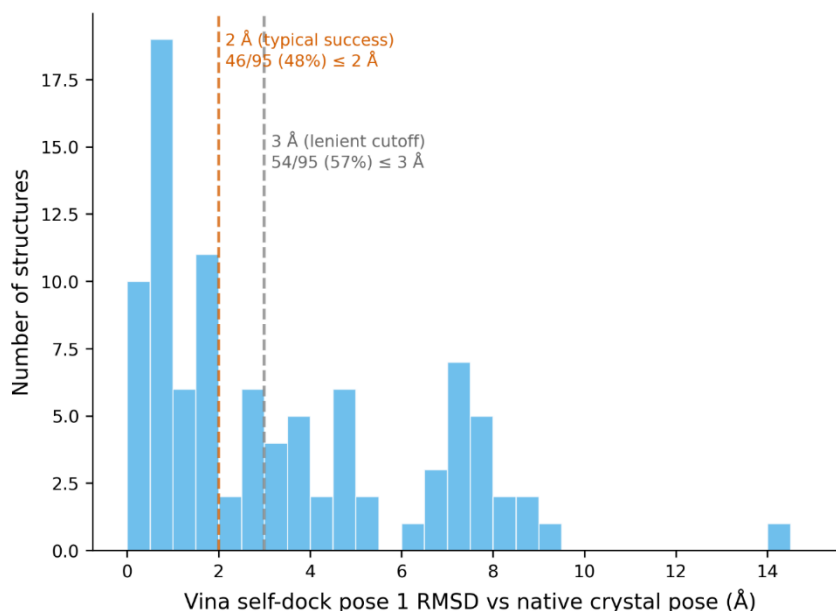


Figure 7. Bar graph representing the population of Vina self-docked ligands for which the self-docked ligand is able to re-capture the native ligand pose based on RMSD between the two. An RMSD of $\leq 2\text{\AA}$ is typically considered a docking success in docking studies.³⁷

Vina was only able to re-capture the native ligand pose with its top-scoring pose to an RMSD score of $\leq 2\text{\AA}$ 48% of the time (Figure 7), which is the industry standard considered to be docking “success”, likely because the resolution of deposited crystal structures being about $2\text{\AA}$.^{30,37} When the threshold is relaxed to $\leq 3\text{\AA}$, the performance of Vina is not significantly improved: 57%. The fact that Vina is only able to re-capture the native pose of CDK2 about half of the time with any degree of confidence indicates that the conformational search-score energy minimization method often fails to locate the crystallographic binding mode. As such, any docking score generated by the molecular docking scoring functions must take this into account – it is likely that Vina is not capturing the true binding pose of a novel ligand when docked into a flexible receptor such as CDK2, and thus the observed binding score may not reflect the true biological activity of that ligand.

The RMSD results shown in Figures 6c and 7 likely account for some of the loss in Pearson $|r|$ that is observed between GNINA's CNN scoring of the native ligand-receptor complex in comparison to the self-docked ligand, decreasing from 0.588 to 0.387. Because GNINA uses Autodock Vina's scoring function to predict binding poses, then scores the resulting ligand pose using its active-learning trained scoring functions, it can only evaluate the ligand pose produced using Vina. This reduction in GNINA's ability to score ligand-protein interactions in accordance with biological is almost certainly partially caused by the limitations of the Vina scoring functions. Figure 6d shows the RMSD of top-ranked self-docked ligand score using GNINA scoring functions, and the graph shows a distinct RMSD trend – ligand top scoring poses which have large RMSD differences from the native ligand pose appear closer to the origin with lower scores (red data points, $\leq$ CNN affinity 6), while smaller RMSD differences (green data

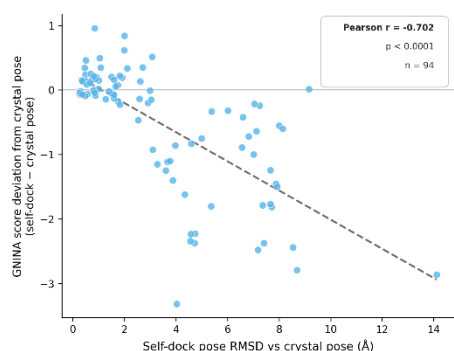


Figure 8. Comparison of the correlation between RMSD deviation from native crystal pose and the GNINA score deviation between its self-docked and crystal pose score.

points) tend to give higher scores and cluster at scores of $\geq$ CNN affinity 6 after re-scoring using GNINA. This demonstrates that GNINA is, in fact, detecting and penalizing these ‘bad’ poses, and the limitations of the pose-search is likely adding a lot of error to the subsequent scores, resulting in the observed decrease in Pearson $|r|$

correlation to biological activity. Figure 8 confirms that GNINA is consistently producing self-docking scores

lower than the crystal pose in direct correlation with RMSD deviation between the two ligand poses, which strongly suggests that any decrease in SAR correlation for the CDK2 self-docked set is due to pose error introduced by the Vina posing search.

3.3.2 Δ Molecular Docking Score and Δ Activity Comparison for RCSB Ligands

The pairwise Pearson r correlations between docking pairs and ΔpIC_{50} for this dataset were also compared, to directly assess the performance of the docking scoring functions in comparison to ligand-based similarity metrics (Figure 9). The secondary purpose was to evaluate the ability of these scoring functions to capture the SAR between ligand pairs and account for the scaffold modifications which should cause a similar biological activity change in CDK2 ligands.

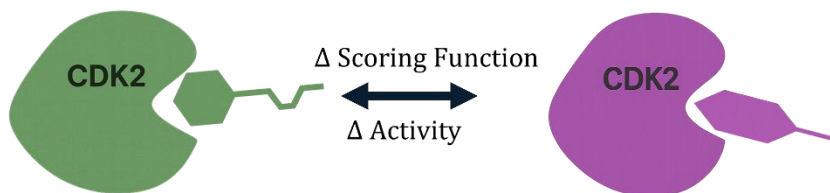


Figure 9. Δ docking score against Δ biological activity statistical analysis comparison performed on crystal CDK2 ligand-protein complexes and self-docked CDK2 ligand-protein complexes.

Based on the moderate Pearson r score observed for the crystal poses scored by GNINA, it was a reasonable expectation that GNINA would handle potency differences in ligand pairs well, and reflect these differences in both the magnitude and direction of the produced binding scores.

The statistical analysis was approached from two directions. First, the $|\Delta pIC_{50}|$ was compared to the $|\Delta \text{docking score}|$ for both Vina and GNINA crystal-pose-complex score and self-docking score, to evaluate which (if any) of these scoring functions can accurately represent the magnitude of the difference in $|\Delta pIC_{50}|$ observed in a ligand pair. Second, the ΔpIC_{50} was compared to the $\Delta \text{docking score}$ for both Vina and GNINA crystal-pose-complex score and self-docking score, to evaluate whether either of these scoring functions can accurately determine the directionality or ranking of the ligand pair; that is to say, accurately predict which is the higher activity ligand and the lower activity ligand in the pair via docking score. Note that any ligand pairs with a ΔpIC_{50} or $\Delta \text{docking score}$ of zero were disregarded for this particular analysis because they fail to give information on directionality in this case. From this metric the pairwise

ranking accuracy of the scoring functions were evaluated (n ligand pairs which correctly rank both ligands / n total ligand pairs).

Table 3. Pairwise relationships between docking score and $\Delta pIC50$ for Vina and GNINA, evaluated on both crystal poses and self-docked poses. Both magnitude ($|\Delta \text{docking score}|$ vs $|\Delta pIC50|$) and directionality ($\Delta \text{docking score}$ vs $\Delta pIC50$) of the relationship are examined.

Method	Pose type	Absolute Pearson r	Magnitude-based Pearson r ($ A $)	Directionality-based Pearson r (A)	n ligand pairs	Pairwise Ranking Accuracy
Vina	Crystal pose	0.404	+0.193	+0.241	4,465	63.8%
Vina	Self-docked pose 1	0.400	+0.229	+0.242	4,465	60.5%
GNINA	Crystal pose	0.588	+0.304	+0.377	4,465	71.0%
GNINA	Self-docked pose 1	0.387	+0.188	+0.223	4,465	63.4%

Magnitude-based Pearson r is calculated between $|\Delta \text{score}|$ for a ligand pair and $|\Delta pIC50|$ across all unique compound pairs to assess whether molecular docking scoring functions GNINA and Vina reflect the magnitude of $|\Delta pIC50|$ in their $|\Delta \text{score}|$. Directionality-based Pearson r is calculated between Δscore for a ligand pair and $\Delta pIC50$ across all unique compound pairs (95 structures, 29 pairs with identical docking scores or $\Delta pIC50$ were dropped because irrelevant to directionality) to assess whether molecular docking scoring functions GNINA and Vina reflect the directionality of $\Delta pIC50$ in their Δscore (correctly determine the highest and lowest activity ligand of the pair by score). "Pairwise ranking accuracy" reflects the % of pairs which correctly capture this directionality – 50% is considered "expected by chance". Absolute Pearson r is the Pearson r from the non-pairwise analysis.

Based on these results, both Vina and GNINA seem to perform worse upon pairwise comparisons of structures (Table 3). GNINA was able to somewhat accurately absolute-rank the crystal-complex dataset in accordance with ligand potency ($r = 0.588$), but when asked to determine the effects of specific structural changes in ligands upon their potency, the correlation drops to $r = 0.304$. Both scoring functions performed worse in predicting differences in biological activity within the self-docked poses compared to the crystal-poses, indicating that the native ligand pose does provide a slight advantage to predictive power. The loss of correlation

when comparing structural ligand changes and their effects on scoring, to differences in biological activity resulting from those changes, has weight implications for the field of VS. Most of these campaigns implicitly assume that small modifications to ligand scaffold will be directly reflected within changes in the scoring function³⁸, which is clearly not the case observed here.

The directionality Pearson r score (Δ docking score vs Δ pIC50) shows an improvement across nearly all four docking methods in comparison to the magnitude correlation scores, with a GNINA correlation of $r = 0.377$ (Table 3). This shows that the scoring functions have some success in ranking the top and bottom potency ligands in a pair, which is relevant to the use of docking to guide pharmacophoric replacement in drug design. Additionally, the pairwise ranking accuracy was above the 50% “correlation-by-chance” baseline for all four scoring function metrics, suggesting that both Vina and GNINA have moderate success in directionally ranking ligand pairs. GNINA is again the top performer in this category, with a pairwise ranking accuracy of 71% when the native ligand-receptor complex is known. Both scoring functions maintain a moderate ability above random chance to identify which of two CDK2 ligands is more potent, but both are unable to accurately estimate the magnitude of this potency difference.

3.4 Discussion and Conclusions

In this study it was shown that across four ligand-based similarity metrics and two structure-based scoring functions, evaluated on both a curated CDK2 RCSB crystal database and three sets of random ChEMBL CDK2 ligand validation sets, no cheminformatics method used to represent ligand similarity was able to predict pairwise differences in biological activity for CDK2 inhibitors. This was reflected in the Pearson r correlation coefficients for SAR comparisons upon the databases:

Table 4. Summary of pairwise Pearson r correlations between computational scores and pIC50 for 95 CDK2 crystal structures.

Method	Type	Absolute r	Pairwise Δr	n
<i>Ligand-based similarity metrics – Pairwise r ($\Delta pIC50$ vs similarity, 4,465 pairs)</i>				
Morgan ECFP4	2D fingerprint	—	−0.094	4,465
ROCS Shape	3D shape	—	−0.159	4,465
ROCS Color	Pharmacophore	—	−0.128	4,465
EON ESP	Electrostatic	—	−0.075	4,465
<i>Structure-based docking scoring functions – Absolute r (pIC50 vs docking score), Pairwise r - ($\Delta pIC50$ vs $\Delta docking\ score$)</i>				
Vina (crystal pose)	Empirical FF	0.404	+0.193	95 / 4,465
Vina (self-dock pose 1)	Empirical FF	0.400	+0.224	95 / 4,465
GNINA (crystal pose)	CNN	+0.588	+0.377	95 / 4,465
GNINA (rescore self-dock)	CNN	+0.387	+0.233	95 / 4,465

Note. Absolute r = Pearson r between score and pIC50 (per ligand). Pairwise Δr = Pearson r between $|\Delta Score|$ and $|\Delta pIC50|$ across all pairs. Directionality Pearson r from Table 3 not reported in this summary. FF = force field. CNN = convolutional neural network (GNINA CNNaffinity). Dashes indicate analysis not applicable for that protocol. Bold value = strongest result in the dataset. Similarity metrics computed across all 4,465 pairwise comparisons (n pairs). Docking correlations computed per ligand (n structures) for absolute pIC50 correlation, and across all pairs for pairwise $\Delta Score$ vs $\Delta pIC50$ correlation.

As explored in section 3.1 and 3.2, molecular similarity, whether evaluated using shape, pharmacophoric, chemical fingerprint, or electrostatic Tanimotos, no structural similarity correlation could be made with differences in biological activity for CDK2 ligands (Table 4). Not even in regions of high similarity-low potency difference could trends in ligand similarity be observed – if anything, these regions yielded worse statistical relationships. It was noted that breaking a native crystal pose dataset into groupings based on active site conformation provided a slight increase in similarity-activity correlation, but this difference was essentially negligible.

Furthermore, when these ligand-based similarity metrics were applied to three random ChEMBL datasets with no ligand pose information, the resulting similarity-activity distribution plots and correlation coefficients appeared nearly identical to those produced by the native-ligand-pose dataset (Table 4), indicating ligand pose does not meaningfully improve similarity-activity relationships. When the native ligand-receptor complexes were scored using molecular docking platforms Vina and GNINA, GNINA was able to produce a Pearson r correlation score between docking scores and ligand pIC50 of 0.588, the most significant SAR result observed in this study. This indicates that it is possible to relate structural similarity to biological activity within the CDK2 ligand database. When the ligands were self-docked back into their native receptors, however, this correlation was decreased, in part due to the inability of the Vina conformational search algorithm to re-capture the native ligand pose with any accuracy. Likewise, when the same ligand pairwise analysis approach was applied to docking scores (Δ docking score vs Δ pIC50), r was degraded to 0.387 – clearly while GNINA can directly rank ligands in agreement with their potency, it fails to capture structural differences in ligands which result in potency magnitude differences with any success. It was, however, able to rank the directionality of ligand pairs correctly 70% of the time (correctly scores which ligand of the pair has the most biological activity as the top scorer of the pair).

The failure of similarity metrics and scoring functions to predict ligand biological activity in CDK2 indicates that the problem is much more complex than can be captured by ligand-based similarity metrics or structure-based scoring functions. Both similarity metrics and scoring functions evaluate a single pose of a ligand, rather than the energetic landscape of a ligand binding event³⁹. Similarly, both fail to account for the biological conditions of binding such as solvent-network reorganization, entropic considerations, and protein active site flexibility, which

is of particular significance in a flexible protein such as CDK2.^{40,41} These conclusions are relevant to the field, given these tools are used extensively during early-stage screening steps of drug design protocols to propose potential scaffolds and screen the performance of new leads. Many of the data produced by similarity metrics and scoring functions are also used to train ML drug discovery models, bringing into question the quality of the data being used to train those models and how it affects performance. The results of this study show that the use of cheminformatic metrics to screen or compare libraries of molecules should be approached with caution, as these metrics are very clearly not robust enough to applied on a global scale.

Sources

- (1) Maggiora, G.; Vogt, M.; Stumpfe, D.; Bajorath, J. Molecular Similarity in Medicinal Chemistry. *J. Med. Chem.* **2014**, 57 (8), 3186–3204. <https://doi.org/10.1021/jm401411z>.
- (2) Martin, Y. C.; Kofron, J. L.; Traphagen, L. M. Do Structurally Similar Molecules Have Similar Biological Activity? *J. Med. Chem.* **2002**, 45 (19), 4350–4358. <https://doi.org/10.1021/jm020155c>.
- (3) Li, S.; Liu, T.; Zhang, X.; Chen, M.; Yang, Y.; Zhao, J.; Wang, M. Machine Learning-Based Virtual Screening Integrating Pharmacophores, Docking and Molecular Descriptors for Discovery of CDK2 Inhibitors. *Results Chem.* **2026**, 27. <https://doi.org/10.1016/j.rechem.2026.103437>.
- (4) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. The Protein Data Bank. *Nucleic Acids Res.* **2000**, 28 (1), 235–242.
- (5) Wang, R.; Fang, X.; Lu, Y.; Wang, S. The PDBbind Database: Collection of Binding Affinities for Protein-Ligand Complexes with Known Three-Dimensional Structures. *J. Med. Chem.* **2004**, 47 (12), 2977–2980. <https://doi.org/10.1021/jm030580l>.
- (6) Tanimoto, T. T. An Elementary Mathematical Theory of Classification and Prediction; International Business Machines Corporation: New York, 1958; pp 1–10.
- (7) Rogers, D.; Hahn, M. Extended-Connectivity Fingerprints. *J. Chem. Inf. Model.* **2010**, 50 (5), 742–754. <https://doi.org/10.1021/ci100050t>.
- (8) Morgan, H. L. The Generation of a Unique Machine Description for Chemical Structures. *J. Chem. Doc* **1965**, 5 (2), 107–113.
- (9) OpenEye, C. M. S. S. F. N. *ROCS 3.5.1.1*. <http://www.eyesopen.com>.
- (10) Grant, J. A.; Gallardo, M. A.; Pickup, B. T. A Fast Method of Molecular Shape Comparison: A Simple Application of a Gaussian Description of Molecular Shape. *J. Comput. Chem.* **1996**, 17 (14), 1653–1666.
- (11) Hawkins, P. C. D.; Skillman, A. G.; Nicholls, A. Comparison of Shape-Matching and Docking as Virtual Screening Tools. *J. Med. Chem.* **2007**, 50 (1), 74–82. <https://doi.org/10.1021/jm0603365>.
- (12) Grant, J. A.; Pickup, B. T. A Gaussian Description of Molecular Shape. *Journal of Physical Chemistry A* **1995**, 99, 3503–3510.
- (13) Ehrt, C.; Krause, B.; Schmidt, R.; Ehmki, E. S. R.; Rarey, M. SMARTS.plus – A Toolbox for Chemical Pattern Design. *Mol. Inform.* **2020**, 39 (12), 1–4. <https://doi.org/10.1002/minf.202000216>.
- (14) OpenEye Scientific Software, S. F. N. *EON 2.4.0.1*. <http://www.eyesopen.com>.
- (15) Honig, B.; Nicholls, A. Classical Electrostatics in Biology and Chemistry. *Classical Electrostatics in Biology and Chemistry* **1995**, 268 (5214), 1144–1149.
- (16) Openeye. *EON 3.1.2.3*; Cadence Molecular Sciences: Santa Fe, NM.

- (17) Thorner, D. A.; Wild, D. J.; Willett, P.; Wright, P. M. *Calculation of Structural Similarity by the Alignment of Molecular Electrostatic Potentials*; 1998.
- (18) Bajusz, D.; Rácz, A.; Héberger, K. Why Is Tanimoto Index an Appropriate Choice for Fingerprint-Based Similarity Calculations? *J. Cheminform.* **2015**, *7* (1). <https://doi.org/10.1186/s13321-015-0069-3>.
- (19) Schonbrunn, E.; Betzi, S.; Alam, R.; Martin, M. P.; Becker, A.; Han, H.; Francis, R.; Chakrasali, R.; Jakkaraj, S.; Kazi, A.; Sebt, S. M.; Cubitt, C. L.; Gebhard, A. W.; Hazlehurst, L. A.; Tash, J. S.; Georg, G. I. Development of Highly Potent and Selective Diaminothiazole Inhibitors of Cyclin-Dependent Kinases. *J. Med. Chem.* **2013**, *56* (10), 3768–3782. <https://doi.org/10.1021/jm301234k>.
- (20) Vogt, M.; Iyer, P.; Maggiora, G. M.; Bajorath, J. Conditional Probabilities of Activity Landscape Features for Individual Compounds. *J. Chem. Inf. Model.* **2013**, *53* (7), 1602–1612. <https://doi.org/10.1021/ci400288r>.
- (21) Bártová, I.; Koča, J.; Otyepka, M. Functional Flexibility of Human Cyclin-dependent Kinase-2 and Its Evolutionary Conservation. *Protein Science* **2008**, *17* (1), 22–33. <https://doi.org/10.1110/ps.072951208>.
- (22) Coxon, C. R.; Anscombe, E.; Harnor, S. J.; Martin, M. P.; Carbain, B.; Golding, B. T.; Hardcastle, I. R.; Harlow, L. K.; Korolchuk, S.; Matheson, C. J.; Newell, D. R.; Noble, M. E. M.; Sivaprakasam, M.; Tudhope, S. J.; Turner, D. M.; Wang, L. Z.; Wedge, S. R.; Wong, C.; Griffin, R. J.; Endicott, J. A.; Cano, C. Cyclin-Dependent Kinase (CDK) Inhibitors: Structure-Activity Relationships and Insights into the CDK-2 Selectivity of 6-Substituted 2-Arylamino-purines. *J. Med. Chem.* **2017**, *60* (5), 1746–1767. <https://doi.org/10.1021/acs.jmedchem.6b01254>.
- (23) Chen, Y.; Mathai, N.; Kirchmair, J.; Kirchmair, J.; Kirchmair, J. Scope of 3D Shape-Based Approaches in Predicting the Macromolecular Targets of Structurally Complex Small Molecules Including Natural Products and Macrocyclic Ligands. *J. Chem. Inf. Model.* **2020**, *60* (6), 2858–2875. <https://doi.org/10.1021/acs.jcim.0c00161>.
- (24) Zdrazil, B.; Felix, E.; Hunter, F.; Manners, E. J.; Blackshaw, J.; Corbett, S.; de Veij, M.; Ioannidis, H.; Lopez, D. M.; Mosquera, J. F.; Magarinos, M. P.; Bosc, N.; Arcila, R.; Kizilören, T.; Gaulton, A.; Bento, A. P.; Adasme, M. F.; Monecke, P.; Landrum, G. A.; Leach, A. R. The ChEMBL Database in 2023: A Drug Discovery Platform Spanning Multiple Bioactivity Data Types and Time Periods. *Nucleic Acids Res.* **2024**, *52* (D1), 1180–1192. <https://doi.org/10.1093/nar/gkad1004>.
- (25) Hunter, F. M. I.; Ioannidis, H.; Bento, A. P.; Bosc, N.; Corbett, S.; Felix, E.; Magarinos, M. P.; Manners, E.; Smit, I. A.; de Veij, M.; O’Boyle, N. M.; Zdrazil, B.; Leach, A. R. Drug and Clinical Candidate Drug Data in ChEMBL. *J. Med. Chem.* **2025**, *68* (19), 19800–19827. <https://doi.org/10.1021/acs.jmedchem.5c00920>.
- (26) OpenEye, C. M. S. S. F. N. *OMEGA 4.2.1.1*. <http://www.eyesopen.com>.

- (27) Hawkins, P. C. D.; Skillman, A. G.; Warren, G. L.; Ellingson, B. A.; Stahl, M. T. Conformer Generation with OMEGA: Algorithm and Validation Using High Quality Structures from the Protein Databank and Cambridge Structural Database. *J. Chem. Inf. Model.* **2010**, *50* (4), 572–584. <https://doi.org/10.1021/ci100031x>.
- (28) Ma, Z.; Ajibade, A.; Zou, X. Docking Strategies for Predicting Protein-Ligand Interactions and Their Application to Structure-Based Drug Design. *Commun. Inf. Syst.* **2024**, *24* (3), 199–230. <https://doi.org/10.4310/CIS.241021221101>.
- (29) Meng, X.-Y.; Zhang, H.-X.; Mezei, M.; Cui, M. Molecular Docking: A Powerful Approach for Structure-Based Drug Discovery. *Curr Comput Aided Drug Des* **2011**, *7* (2), 146–157.
- (30) Boittier, E. D.; Tang, Y. Y.; Buckley, M. E.; Schuurs, Z. P.; Richard, D. J.; Gandhi, N. S. Assessing Molecular Docking Tools to Guide Targeted Drug Discovery of Cd38 Inhibitors. *Int. J. Mol. Sci.* **2020**, *21* (15), 1–19. <https://doi.org/10.3390/ijms21155183>.
- (31) Oshima, H.; Re, S.; Sugita, Y. Prediction of Protein-Ligand Binding Pose and Affinity Using the GREST+FEP Method. *J. Chem. Inf. Model.* **2020**, *60* (11), 5382–5394. <https://doi.org/10.1021/acs.jcim.0c00338>.
- (32) Eberhardt, J.; Santos-Martins, D.; Tillack, A. F.; Forli, S. AutoDock Vina 1.2.0: New Docking Methods, Expanded Force Field, and Python Bindings. *J. Chem. Inf. Model.* **2021**, *61* (8), 3891–3898. <https://doi.org/10.1021/acs.jcim.1c00203>.
- (33) Morris, G. M.; Ruth, H.; Lindstrom, W.; Sanner, M. F.; Belew, R. K.; Goodsell, D. S.; Olson, A. J. Software News and Updates AutoDock4 and AutoDockTools4: Automated Docking with Selective Receptor Flexibility. *J. Comput. Chem.* **2009**, *30* (16), 2785–2791. <https://doi.org/10.1002/jcc.21256>.
- (34) Trott, O.; Olson, A. J. AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading. *J. Comput. Chem.* **2010**, *31* (2), 455–461. <https://doi.org/10.1002/jcc.21334>.
- (35) Quiroga, R.; Villarreal, M. A. Vinardo: A Scoring Function Based on Autodock Vina Improves Scoring, Docking, and Virtual Screening. *PLoS One* **2016**, *11* (5), 1–18. <https://doi.org/10.1371/journal.pone.0155183>.
- (36) McNutt, A. T.; Francoeur, P.; Aggarwal, R.; Masuda, T.; Meli, R.; Ragoza, M.; Sunseri, J.; Koes, D. R. GNINA 1.0: Molecular Docking with Deep Learning. *J. Cheminform.* **2021**, *13* (1). <https://doi.org/10.1186/s13321-021-00522-2>.
- (37) Onodera, K.; Satou, K.; Hirota, H. Evaluations of Molecular Docking Programs for Virtual Screening. *J. Chem. Inf. Model.* **2007**, *47* (4), 1609–1618. <https://doi.org/10.1021/ci7000378>.
- (38) Jorda, R.; Havlíček, L.; McNae, I. W.; Walkinshaw, M. D.; Voller, J.; Šturm, A.; Navrátilová, J.; Kuzma, M.; Mistrik, M.; Bártek, J.; Strnad, M.; Kryštof, V. Pyrazolo[4,3- d]Pyrimidine Bioisostere of Roscovitine: Evaluation of a Novel Selective Inhibitor of Cyclin-Dependent Kinases with Antiproliferative Activity. *J. Med. Chem.* **2011**, *54* (8), 2980–2993. <https://doi.org/10.1021/jm200064p>.

- (39) Zhou, H. X.; Gilson, M. K. Theory of Free Energy and Entropy in Noncovalent Binding. *Chem. Rev.* **2009**, *109* (9), 4092–4107. <https://doi.org/10.1021/cr800551w>.
- (40) Martin, M. P.; Endicott, J. A.; Noble, M. E. M. Structure-Based Discovery of Cyclin-Dependent Protein Kinase Inhibitors. *Essays in Biochemistry*. Portland Press Ltd November 8, 2017, pp 439–452. <https://doi.org/10.1042/EBC20170040>.
- (41) Huang, S. Y.; Zou, X. Advances and Challenges in Protein-Ligand Docking. *Int. J. Mol. Sci.* **2010**, *11* (8), 3016–3034. <https://doi.org/10.3390/ijms11083016>.

Chapter 4 – Ligand Cross-Docking and Scoring Function Evaluation

4.1 Scoring-Function Analysis on a Cross-Docked CDK2 Dataset

The goal of this chapter is to evaluate whether empirical and CNN scoring functions can preserve structure-activity relationships (SAR) under cross-docking conditions and to determine how these results can be used to optimize virtual screening (VS) protocols. While there is some literature precedent which shows empirical scoring functions like Vina breaking down with the conformational noise associated with cross-docking,¹ it is important to determine whether empirical scoring functions retain predictive ability under cross-docking conditions and to identify where they begin to fail. Chapter 3 demonstrated that GNINA's ability to correlate docking scores with biological activity decreases when ligand conformational freedom is introduced. It is therefore important to determine whether these scoring functions can still preserve meaningful structure-activity relationships under cross-docking conditions. Cross-docking more closely mimics prospective virtual screening experiments, making it important to determine whether either scoring function can preserve meaningful SAR under these conditions.

The effects of protein conformational flexibility and the utility of ensemble docking in docking score-potency correlation is another topic which is explored in this chapter. Protein flexibility and active site conformation have significant influence on ligand binding¹. For some proteins, the active site can adopt conformations so divergent that they can bind multiple different native ligands.² Rigid-receptor docking assumes that a single receptor pose can represent a target binding site, but many flexible proteins, including CDK2, bind to ligands in more than one conformation³. This has the potential to generate misleading docking scores in a drug screen, which is a serious problem that needs to be addressed in any VS screen performed

on a flexible protein. As such, many studies have attempted to characterize the number of representative receptor conformations needed to form a crystal-posed ensemble that accurately describes the protein-ligand binding complex, such that the potency of lead compounds can be approximated from docking scores.^{2,4,5} This is a valuable question which has produced a variety of answers, from “one, as long as it is promiscuous enough”⁶ to “a curated ensemble”⁵. This question is explored in the following chapter in greater detail, with the aim of providing additional evidence regarding the optimal size and composition of receptor ensembles for CDK2.

4.1.1 Mean-Ligand Cross-Docking Scoring Function Analysis on CDK2 Dataset

Although both Vina and GNINA have been benchmarked using CDK2 and kinase datasets on several occasions, it was still relevant to explore for the scope of this study. One of the main goals of this work is to identify the cheminformatics tools which can accurately represent CDK2 SAR correlation under cross-docking conditions, using the results to optimize VS protocols⁷⁻⁹. Many of the studies performed using these molecular docking programs and ligand-receptor datasets have been done for the purpose of improving docking scoring functions and ligand pose prediction¹⁰, rather than addressing the broad issue cheminformatics struggles with in defining and characterizing SARs in protein-ligand complexes, and possible explanations for the failure of this key principle. Chapter 3 showed that the GNINA CNN and Vina empirical scoring functions exhibited only moderate score–potency correlations for crystal poses and failed to accurately rank pairwise activity differences. It was therefore important to determine whether either scoring function could preserve meaningful structure–activity relationships under more representative virtual screening conditions, where the native ligand pose is unknown. Cross-docking, which involves removing the native ligand from its receptor computationally and docking into all other available crystal-posed receptor templates (Figure 1),¹¹ was performed on

the ligand-receptor crystal-posed dataset to simulate a condition where the native pose was unknown, although there is less uncertainty in this dataset than would be in a large-scale VS screen.

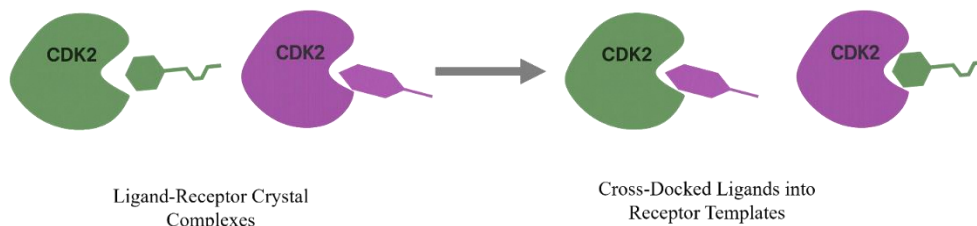


Figure 1. Cross-docking analysis performed on crystal-posed ligand dataset – native ligand was removed from active site and inserted into a different ligand’s receptor.

GNINA and Vina molecular docking platforms were used to perform the conformational search and subsequent scoring of all ligands. It is worth noting that CNN scoring functions have

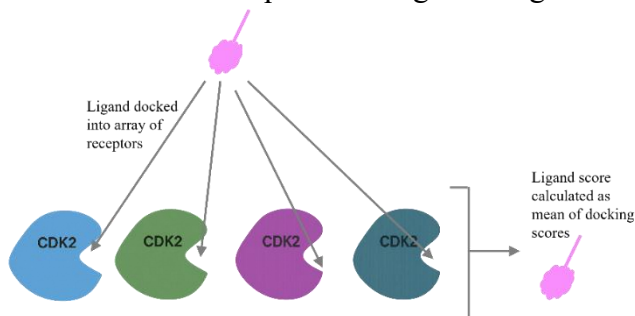


Figure 2. Showing how the ligand cross-docked into every receptor was condensed into one mean ligand score.

been trained on some cross-docked datasets, so they may be more robust to receptor-ligand mismatch scenarios than Vina’s simple approximations¹²

ability to form potency-binding score relationships, the mean score for each ligand was calculated by averaging its scores upon docking into each receptor (Figure 2). This method was used to keep the dataset manageable and interpretable and to retain the ability to analyze each

To analyze both scoring functions'

ligand as a single independent observation rather than a series of data points, which is consistent with the pIC50 being a single ligand property.

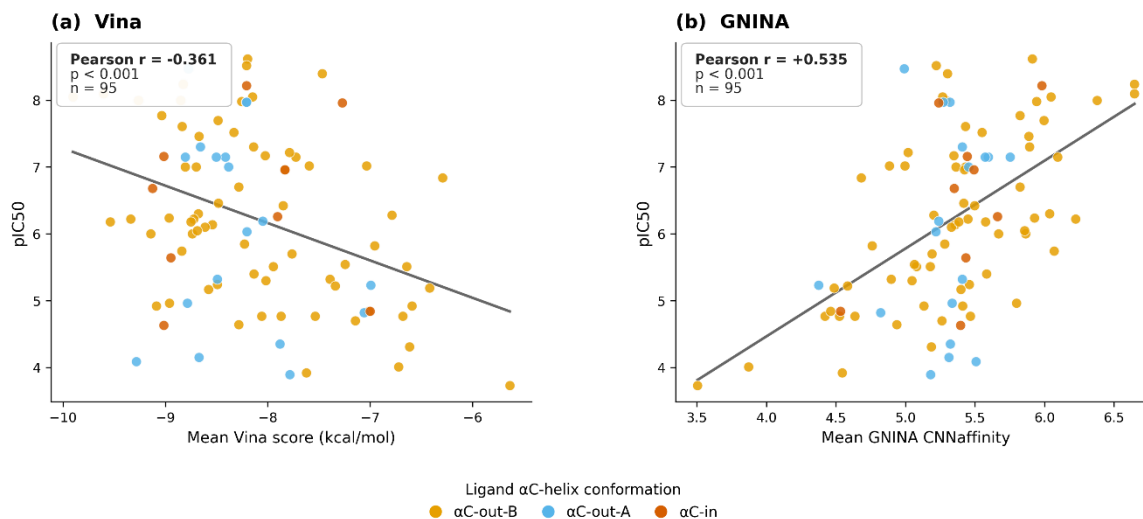


Figure 3. Results for cross-docking 95 pairs of ligands using Autodock Vina (a) and GNINA (b). Each data point represents the ligand mean score from docking that ligand into all receptors. Colours correspond to active site conformational groupings established in Chapter 2.

Table 1. Comparison table of docking procedures performed on the CDK2 RCSB dataset and their SAR correlation

Method	Comparison Condition (X vs Y)	Pearson r values (X/Y)	Stieger Z	p-value	Significant?
Vina	Cross-docked vs crystal-posed	-0.361/-0.404	0.552	0.581	No
Vina	Cross-docked vs self-docked	-0.361/-0.405	0.880	0.379	No
Vina	Crystal-posed vs self-docked	-0.404/-0.405	0.013	0.990	No
GNINA	Cross-docked vs crystal-posed	0.535/0.588	-0.939	0.348	No
GNINA	Cross-docked vs self-docked	0.535/0.387	2.006	0.045	Yes
GNINA	Crystal-posed vs self-docked	0.588/0.387	2.610	0.009	Yes

Comparison condition: indicates the two docking procedures being compared. The Steiger z-test is necessary to directly compare Pearson r correlations because the datasets across all six conditions are not independent. p-value of <0.05 indicates that the difference in Pearson r correlations observed between the two conditions is statistically significant.

The results in Figure 3 show that GNINA maintains a moderate correlation between docking score and biological activity ($r = 0.535$), despite the additional conformational noise introduced by docking each ligand into multiple non-native receptor conformations. A p-value of <0.0001 indicates that this relationship is statistically significant. This correlation did not differ significantly from that observed for the crystal-posed dataset ($r = 0.588$; Table 1) and was

stronger than that observed for the self-docked dataset ($r = 0.387$). This is a surprising result, but it likely ties to the RMSD deviation issues presented in Chapter 3. The correlation between Δ GNINA score (self-docked - crystal pose) and RMSD was found to have a Pearson r of -0.702, indicating that as the self-docked pose RMSD deviates from the original crystal pose, GNINA penalizes this difference substantially. The improved potency-docking score correlation observed in the cross-docked set is likely due to the mean score of the ligand accounting for pose error – essentially, many samples of the ligand pose within the active site “smooth over” individual poor pose estimates, creating a better activity prediction through docking score. A Steiger z -test showed that the Pearson correlation coefficients for the GNINA self-docked dataset differed significantly from those of both the crystal-posed and cross-docked datasets (Table 1). This finding suggests that averaging ligand scores across multiple receptor conformations may partially compensate for pose-dependent scoring errors, resulting in a stronger score-potency relationship than self-docking alone.

Vina’s results were somewhat expected; in Chapter 3, Vina’s Pearson r score was -0.400 for the dataset of CDK2 ligands when self-docked. Vina’s docking score-potency correlation for the cross-docked dataset is -0.361, which is somewhat similar to the self-docked score and only shows a small decrease in Pearson r value, which is somewhat surprising given the cross-docking introduces significant noise. The difference between the two is +0.039, a somewhat small gap, indicating that despite the increased noise associated with cross-docking a ligand into a non-native receptor, Vina does not perform markedly worse at relating docking score to biological potency. This is likely for similar reasons as observed with the GNINA scoring function – averaging ligand pose over many samples smooths out outliers, creating a better docking score prediction.

To determine whether this was a result of several badly docked ligands heavily skewing the Vina correlation score, the potency-docking score was recalculated using the maximum docking score for each ligand after docking into the ensemble of receptors, rather than the mean score obtained by docking each ligand into the receptor ensemble, to represent each ligand in the dataset (Table 1). Given that VS drug design experiments do not have native ligand poses to compare to, this analysis seemed relevant despite the fact that all of the ligands in the CDK2 dataset do have native poses.

The use of maximum ligand docking score improved Vina somewhat, but not a large difference ($\Delta r = -0.056$). That the use of maximum score for a ligand docked into an ensemble somewhat improved the Vina correlation indicates that Vina likely struggles significantly to score off-target ligands and is producing outliers which pull the mean off-target (Table 1). However, this difference was not found to be statistically significant ($z = 1.55$, $p = 0.122$), thus choosing

Table 2. Comparison of the effects of cross-docked ligand dataset representation selection on SAR correlation

Method	Analysis method	Pearson r / p-value	Steiger z / p-value
GNINA	Mean	0.535 / <0.0001	0.195 / 0.8451
GNINA	Max	0.525 / <0.0001	0.195 / 0.8451
Vina	Mean	-0.360 / 0.0003	1.55 / 0.1221
Vina	Max	-0.416 / <0.0001	1.55 / 0.1221

Pearson r calculated using the different analysis methods listed: "Mean" indicates that mean ligand score after docking into receptor array was trended against that ligands pIC_{50} value. "Max" means the maximum docking score after docking into receptor array was the score trended against that ligands pIC_{50} value. Steiger z -value calculated between "mean" and "max" Pearson r -values within each scoring function methodology.

the mean or maximum ligand score does not meaningfully affect docking score-potency correlation results (Table 1). GNINA loses a negligible amount of SAR correlation when the maximum ligand value is chosen, indicating it is robust to statistical analysis type and is scoring each ligand similarly when the ligand is

cross-docked into receptors with moderately different active site conformations. A Steiger z -test showed the difference in using mean or maximum ligand score when cross-docking ligands into an ensemble using GNINA does not significantly affect SAR correlation ($z = 0.195$, $p = 0.845$),

thus the choice of ligand scoring method was found to be arbitrary within both molecular docking methods (Table 1).

4.1.2 Effects of Active Site Conformation on Mean-Ligand CDK2 Cross-Docking

Based on the results of Chapter 3, which showed a slight increase in similarity metrics for the CDK2 ligand dataset divided into subsets by the active site conformation RMSD clustering, it made sense to perform a similar analysis using the cross-docking. The mean score of a matched ligand conformational subset docked only into its native receptor conformation versus docking the entire mean ligand dataset into that same receptor conformational grouping was analyzed (Figure 3).

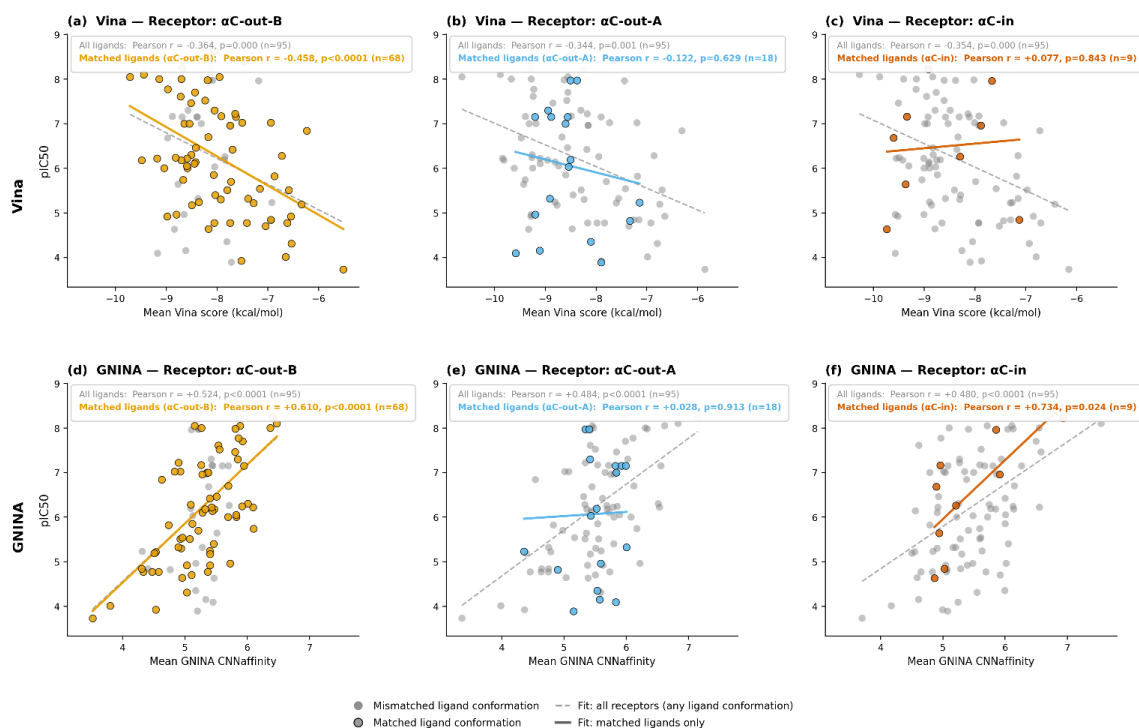


Figure 4. Results of Autodock Vina and GNINA cross-docking on the 95-ligand-receptor complex RCSB crystal posed database. Ligands are cross-docked, and the mean of the score of one ligand docked into all receptors is assigned as that ligand's score value, which is then compared with pIC50 activity. a, b, c) Vina cross-dock showing all ligands cross-docked into each of the three assigned active site conformation receptors. Coloured data points correspond to the mean ligand score when ligands were docked into their native conformation pose. Grey data points correspond to all ligands docked into that conformational pose. d, e, f) GNINA cross-dock showing all ligands cross-docked into each of the three assigned active site conformation receptors. Coloured data points correspond to mean ligand score when ligands were docked into their native conformation pose. Grey data points correspond to all ligands docked into that conformational pose. Active site conformations grouped based on methods described in Chapter 2.

The Vina results show little consistent improvement in docking score-potency SAR when clustered based on receptor conformation, with the exception of the α C-out-B group, which appears to largely carry the correlation observed between ligand mean docking score and potency (Figure 4a). All three conformational groupings showed somewhat similar correlation values when all ligands were docked into one conformational grouping (Pearson r range = -0.364 to -0.344), meaning Vina does not favour any one receptor conformation for docking score potency correlation over the others. Conformational matching does not seem to substantially rescue Vina's ability to rank ligands based on potency for either of the other two groups; in fact, both α C-out-A and α C-in show decreased Pearson r scores and p -values of $p < 0.05$, indicating these correlations are both weak and are not statistically significant. The grouping results are somewhat surprising given that Vina does not show a strong correlation between cross-docking mean ligand score and potency within the full dataset ($r = -0.361$), but once the dataset is split it becomes evident that the two conformational subsets, α C-out-A and α C-in are negatively affecting the stronger correlation established in α C-out-B. This suggests there may be a benefit to maximizing structural diversity within ligand subsets to establish SAR trends, as in α C-out-B is by far the most structurally diverse and well-populated group ($n=68$).

These results are somewhat reflected in the split GNINA conformational subgroups, with α C-out-B's mean-ligand docking score improving when grouped by conformational subset, (Figure 3d). This mirrors the trend observed in the Vina result, with the α C-out-B group ($n=68$) largely carrying the observed trend within the entire dataset, although in the GNINA results this score-potency relationship is further supported by α C-in within the full dataset. All three conformational groupings showed somewhat similar correlation values when all ligands were docked into one conformational grouping (Pearson r range = 0.480 to 0.524), meaning GNINA

does not favour any one receptor conformation for docking score potency correlation over the others. Where the α C-in group had a Pearson r score of 0.077 in Vina and the correlation was not significant ($p=0.843$), indicating a negative relationship between score and potency, α C-in has a strong positive correlation in GNINA (0.743). Though this subset is small, a p -value of 0.024 implies this correlation is significant. Most interesting, group α C-out-A shows a marked decrease in correlation with potency in comparison to the full dataset in GNINA ($r = 0.028$) – this subgroup shows almost no correlation with biological activity (Figure 4e). Further, the p -value of 0.913 shows the correlation is close to the null hypothesis, reinforcing the lack of mean ligand docking score-potency correlation within this subgroup.

Overall, the results of the analysis in Figure 4 show that docking ligands into their native receptor conformations does not improve correlation consistently within those receptor groupings. In fact, the results of this analysis suggest that ligand-receptor docking score-potency correlation is a ligand-based property, as ligands which bind to specific active-site conformations appear to have largely divergent correlations when separated into their respective active-site RMSD-assigned groupings. This implies that the utility of identifying different active-site conformations may be useful in establishing SAR continuity simply because the ligand groups which possess SAR continuity and those which do not may be separated from each other.

Based on these results, the effects of dataset splitting via conformational subset on the results of GNINA and Vina cross-docking with CDK2 ligands were further explored to determine the impact of performing this pre-screening step as a VS protocol. For this analysis, a mean score was calculated for each ligand by docking into all receptors of the designated conformational subset and taking the mean of the resulting scores as the representative score of the ligand. These

were then correlated with each ligand's corresponding pIC₅₀ value to assess correlation. This approach allows a ligand to be treated as a single data point, consistent with the pIC₅₀ being a single-ligand property (Figure 5).

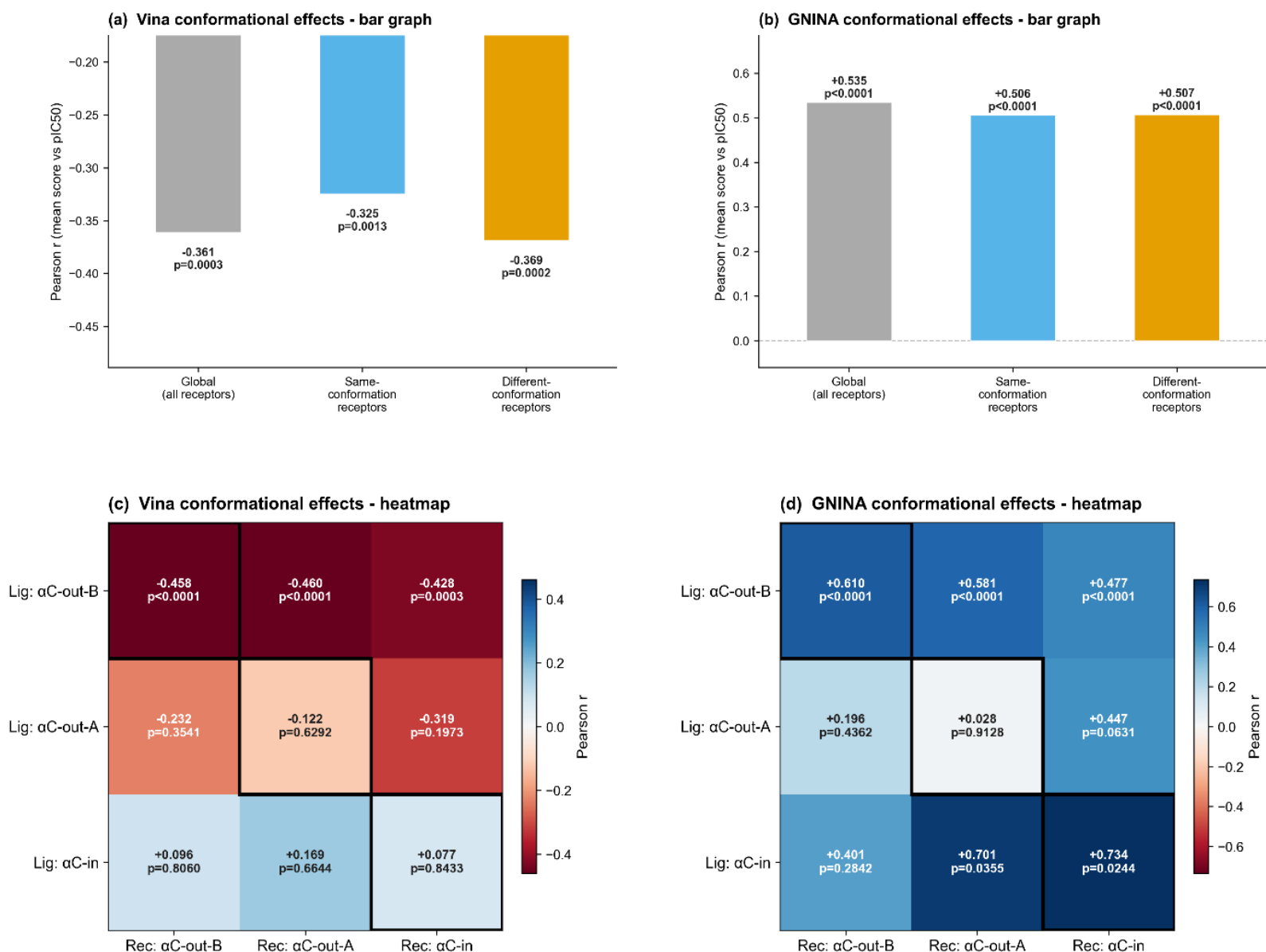


Figure 5. Assessment of the effects of docking into different active site conformations in Vina and GNINA scoring functions. Bar graphs a) and b): Pearson r calculated using the correlation between pIC50 and ligand docking score in Vina and GNINA for all 3 conditions: global = ligand mean score taken after docking into all receptors; same conformation = ligand mean score calculated after docking only into native conformation receptors; different conformation = ligand mean score calculated after docking only into non-native conformation. Heatmaps c) and d): Pearson r calculated using correlation between pIC50 and ligand docking score in Vina and GNINA for each conformational group. Score is represented for each ligand by docking into all the receptors in the designated conformational group (x-axis). These are then grouped into the designated ligand conformational group (y-axis).

Figures 5a and 5b show a near-identical correlation between docking score and potency for all three conditions: ligand mean score after docking into all receptors (grey), after docking

into only native conformation (blue), and after docking into only non-native conformation (red). A slight decrease in correlation between pIC50 and docking score below the correlation score for the entire dataset ($r = 0.535$ in GNINA, $r = -0.361$ in Vina) is observed when ligands are only docked into native active site configurations. This is counter-intuitive from the expected result – the implication is that docking score accuracy is actually decreased by excluding non-native active site receptors from a drug screen. This indicates that docking only into the native receptor active site conformation would weaken docking score-potency prediction within a VS experiment.

Figure 5c and 5d explores these further and characterizes the correlation differences between each ligand active site preference docked into each active site configuration. This is important, as it allows the analysis of some trends which are too detailed for the rough analysis in the previously explored graphs to show. Note that for this correlation analysis each ligand in the y-axis groups is represented by their mean score after docking only into the designated receptor conformation specified along the x-axis. In Figures 5c and 5d, it becomes evident that the α C-out-B group consistently outperforms at docking into any active site conformation across both docking scoring functions. What's more, the CDK2 α C-out-A group performs the worst across both docking scoring functions, and none of its mean-ligand scores in any receptor conformation have a significant correlation, that is to say, $p > 0.05$ for all Pearson r values belonging to the three distinct receptor conformations that this ligand group is docked into. Finally, the α C-in group behaves rather curiously: in Vina, docking this ligand group into all three receptor conformations results in significant decreases in mean docking score-potency Pearson r correlations (these values are so poor they are positive) with little to no statistical significance beyond the null hypothesis ($p >> 0.05$). In contrast, this ligand set has consistently

high correlations between mean docking score and ligand potency when docked into all three receptor conformations, and two out of three of these values are found to be statistically significant.

These observations may be explained using the results of the RMSD clustering analysis performed in Chapter 2, Section 2.1.2 – the CDK2 α C-out-B set is the largest cluster (n=68) with a tight spread in α C-helix RMSD (standard deviation of 0.19Å), while the α C-out-A set has the largest α C-helix RMSD spread (0.25Å), a smaller number of crystal-posed structures from which to establish a trend (n=18) and the most ambiguity within the dataset for conformational assignment, as shown in Chapter 2. This could explain its lack of docking-score-potency correlation – the active site α C-helix positioning is more flexible within this subgroup, leading to larger differences in potency for smaller changes in ligand similarity, confounding the SAR trends that GNINA and Vina may be able to detect. The α C-in set is slightly harder to understand due to its opposing behaviour within the two scoring functions. The α C-helix RMSD shows that the α C-in conformation forms a distinct cluster, with its center located at 6.11Å (almost 5Å) away from the other two conformational groups (Chapter 2, Figure 2). Although it contains the fewest structures (n = 9), it also exhibits the smallest cluster spread (0.13Å), suggesting that this active-site configuration is structurally well-defined. It is possible that the consistent geometry of this active site conformation is handled well by the GNINA scoring function, which uses pattern-based recognition of binding interactions, but some aspects of this conformational subset binding mode are heavily penalized by the Vina scoring functions, perhaps sterics. It is difficult to understand this particular grouping's behaviour within the two scoring functions without performing a more in-depth structural analysis of the ligand binding, however.

4.1.3 Cross-Docking Analysis of Scoring Functions for Ligand Ranking Accuracy

Based on the pairwise ranking analysis performed on Vina and GNINA in Chapter 3, it made sense to assess pairwise ranking accuracy for the cross-docked ligands. For this analysis, all cross-docked ligand scores ($n = 4,465$) were examined in a pair-wise manner against each receptor ($n=95$) to determine whether the GNINA top- and bottom- scoring ligand when cross-docked into each receptor reflects the native binding pose pIC50 potency ranking of the ligands. The analysis was deliberately limited to pairs docked into the same receptor to simulate VS screening conditions, where non-native ligands will be docked into the same target receptor and ranked.

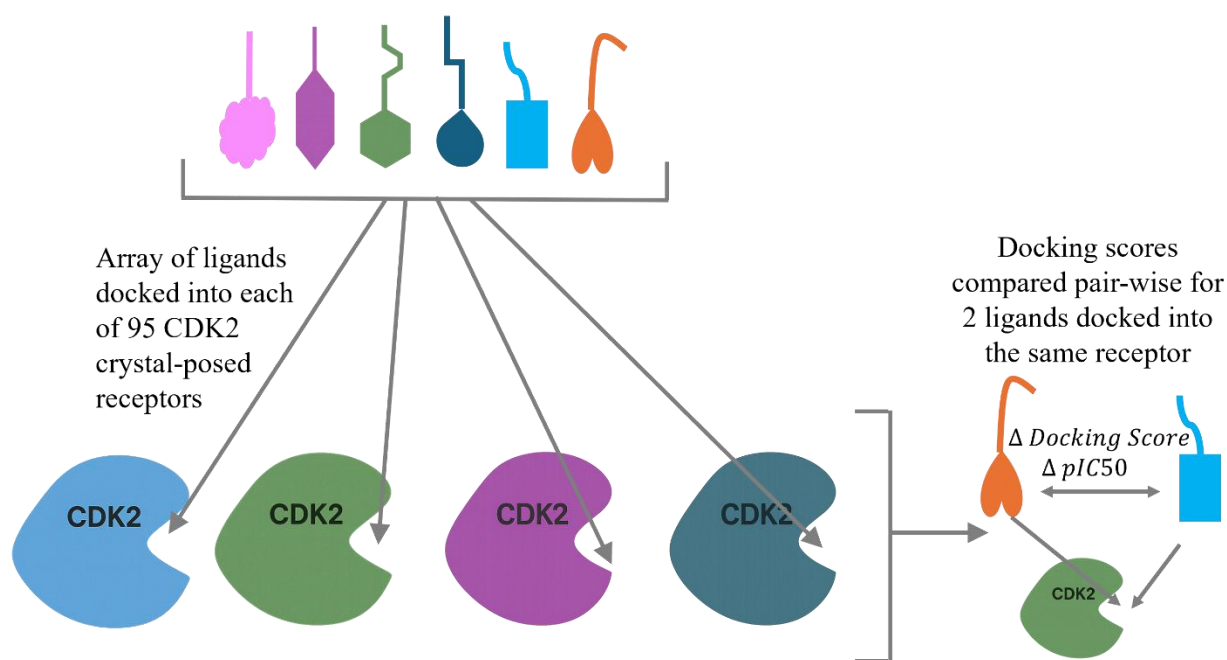


Figure 6. Diagram showing how the Pairwise Ranking Rate study was conducted. Only scenarios where both ligands in the pair were docked into the same receptor were considered, to simulate VS protocols.

Table 3. Summary of ability of scoring functions Vina and GNINA to predict pairwise ranking of cross-docked ligands in accordance with pIC50, and how receptor active site conformation affects prediction

Method	Receptor Scope	Signed Pearson r	Pairwise Ranking Accuracy	Δ vs non-native (%)	n pairs
Vina	All receptors	0.181	59.7%	—	421,306
Vina	Native conformation	0.245	62.8%	+3.6%	156,856
Vina	Non-native conformation	0.167	59.2%	—	108,576

Method	Receptor Scope	Signed Pearson r	Pairwise Ranking Accuracy	Δ vs non-native (%)	n pairs
GNINA	All receptors	0.166	59.2%	—	421,419
GNINA	Native conformation	0.205	61.2%	+2.1%	156,889
GNINA	Non-native conformation	0.169	59.1%	—	108,610

Signed Pearson r calculated using pairwise comparison of two cross-docked ligand scores within the same receptor and both their pIC50 values (Figure 6). ‘Signed’ means the lower pIC50 value is subtracted from the higher, with the docking score difference calculated in correspondence (Vina scores are multiplied by -1 to standardize). Pearson r therefore measures the pairwise ranking accuracy and the magnitude accuracy within each scoring functions docking scoring. “Native receptors” means the dataset was limited to pairs where both ligands were from receptor poses of the same active site conformation. “Non-native conformation” means either one or both ligands in the pair were from a different active-site conformation. “All receptors” means all cross-docked score pairs were calculated within all 95 receptor conformations. All pairs with the same pIC50 score or docking score were dropped since they do not contribute to the analysis. “Pairwise ranking accuracy” reflects the % of pairs which correctly capture this directionality (n pairs within category/ n total pairs $\times$ 100%) – 50% is considered “expected by chance”.

Table 3 shows that both Vina and GNINA are able to rank the highest and lowest potency ligands of a pair with moderate success under pseudo-drug-screening conditions. The pairwise ranking accuracies for the entire cross-docked set within all three receptor conditions are meaningfully above the chance cutoff (50%).¹³ Vina’s ability to predict concordance of docking score to pIC50 score actually out-performs GNINA, which is curious given that GNINA performed better than Vina with pairwise ranking on CDK2 native crystal poses and self-docked poses. Vina also gains a small advantage (+3.6%) in identifying ligand ranking when these ligands are compared in their homologous active site conformation, and loses a small portion of ability to properly rank ligands when they are compared in non-native active site conformations (-0.5%). GNINA also gains a small advantage when docking into the native receptor active site conformation in comparison with the whole receptor set. These results are curious, as they disagree with the mean-ligand cross-docking analysis performed in Section 4.1.2, where ligand mean docking score is found to be largely unaffected by which receptor active site conformation it is docked into. This could be due to fact that calculating the mean-score of the ligand as was done in the previous section overpowers the small, localized benefit of docking into a native

receptor. When pairwise docking scores are compared, the ligand pair is ranked more accurately when it interacts with its native active site binding conformation as its score belongs to a single ligand, not many scores amalgamated into one. When the mean ligand binding score is calculated across many binding events, this local accuracy is lost due to many ligand binding poses being wrapped into one single mean value. Additionally, the small size of the α C-out-A and α C-in groupings make the conclusions that can be drawn about mean-ligand docking into native conformation difficult to support without further investigation and larger subsets of ligand-receptor complexes for cross-docking.

4.2 Discussion and Conclusions

In this chapter, it was established that Vina and GNINA are moderately capable of establishing SAR relationships in CDK2 datasets. For a cross-docked dataset of co-crystallized protein-receptor complexes, it was found that GNINA can correlate the mean score for a ligand docked into an ensemble of receptors with its accompanying pIC50 data with moderate accuracy. GNINA was found to perform better on cross-docked datasets than self-docked sets, likely due to the normalizing effect of multiple ligand pose samples being considered rather than just one. Vina showed a decrease in accuracy when scoring cross-docked mean ligand poses in accordance with biological activity in comparison to its capability to score self-docked poses, which is in agreement with other literature studies.¹⁴ It was also discovered that docking a non-native ligand into CDK2 receptors with its native active site conformation results in a slight decrease in scoring-potency correlation from baseline (all receptors) in both GNINA and Vina; and similarly, docking a non-native ligand into CDK2 receptors with a different active site conformation than its native receptor results in a slight penalization to the scoring-potency correlation from baseline (all receptors) in GNINA, but an increase in accuracy in Vina (Figure 4). Further, it was observed

that ligands clustered based on the active-site RMSD they preferentially bind with were found to have very different docking score correlations with pIC50 across different groups – some had very high correlation when isolated by group, and some had very low correlation. These results were fairly independent of receptor conformation which the ligands were docked into, indicating that SAR correlation may be a property of found within some groups of ligand binders, and absent in others. Finally, both GNINA and Vina correctly ranked pairs of cross-docked CDK2 ligands according to their biological activity in approximately 60% of cases. Pairwise ranking accuracy increased slightly when both ligands were docked into their native receptor conformations and decreased slightly when one or both ligands were docked into non-native receptor conformations. This is an increase in accuracy from the concordance observed in the self-docked set, likely owing to the smoothing effect that sampling many different ligand poses has on scoring accuracy. It also disagrees with the results presented above, which indicate no improvement in ligand docking when docked into the native receptor conformation over other active site conformations, indicating that this may be a localized effect which is lost if the mean score of the ligand docked into many receptors is calculated.

These results have several implications for establishing VS protocols to optimize drug discovery in CDK2 experiments. They indicate that ligands clustered based on receptor conformation may have inherent SAR correlation properties, either positive or negatively affecting the overall SAR landscape correlation. It may be useful to separate ligand scaffold groups to identify whether some tend towards SAR continuity or discontinuity. Even more curious, the receptor conformation being docked into seems to have no bearing upon SAR correlation. Also, GNINA scoring-potency correlation ability benefits from multiple ligand poses being generated, as occurs in cross-docking, and calculating the mean score over these ligand

poses. As such, it is worth identifying a small number of representative receptor active site poses for flexible proteins like CDK2 as a pre-screening step, to ensemble dock the ligands of interest into, given that it was shown that docking into native receptor conformations improved the ability to correctly rank ligands within both scoring functions. Though GNINA uses the Autodock Vina conformational search,^{12,15} generating multiple ligand poses for each ligand alleviates some of the scoring penalties associated with poor ligand posing.

This chapter explored the role of scoring functions in establishing SAR correlations while simulating pseudo-VS conditions through cross-docking of the CDK2 dataset. Because cross-docking allows the docking of a non-native ligand into a receptor of interest, it somewhat replicates the circumstances of a drug screen, where various ligand scaffolds are tested against a target of interest. The goal of this chapter was to evaluate the ability of scoring functions to correlate docking and potency under sub-optimal conditions, and to assess the effects of conformational splitting via active site RMSD on the cross-docked dataset. While there have been one or two similar literature-documented studies on optimizing VS workflows using scoring functions to produce better accuracy in CDK2 drug-screening such as the one conducted by Duca et al.,^{5,16} these typically focus on constructing an ensemble for scoring, not testing the value a representative ensemble can add to SAR correlation, and using this to guide VS development protocols. Moreover, none of these studies question the ability of these scoring functions to rank ligand pairs accurately. As such, this work has addressed several points of interest in the field, and provided a thorough analysis of the ability of empirical and CNN scoring functions to establish SARs in CDK2 datasets.

Sources

- (1) Totrov, M.; Abagyan, R. Flexible Ligand Docking to Multiple Receptor Conformations: A Practical Alternative. *Curr. Opin. Struct. Biol.* **2008**, *18* (2), 178–184. <https://doi.org/10.1016/j.sbi.2008.01.004>.
- (2) Thomas, M. P.; McInnes, C.; Fischer, P. M. Protein Structures in Virtual Screening: A Case Study with CDK2. *J. Med. Chem.* **2006**, *49* (1), 92–104. <https://doi.org/10.1021/jm050554i>.
- (3) Martin, M. P.; Endicott, J. A.; Noble, M. E. M. Structure-Based Discovery of Cyclin-Dependent Protein Kinase Inhibitors. *Essays in Biochemistry*. Portland Press Ltd November 8, 2017, pp 439–452. <https://doi.org/10.1042/EBC20170040>.
- (4) Ricci-Lopez, J.; Aguila, S. A.; Gilson, M. K.; Brizuela, C. A. Improving Structure-Based Virtual Screening with Ensemble Docking and Machine Learning. *J. Chem. Inf. Model.* **2021**, *61* (11), 5362–5376. <https://doi.org/10.1021/acs.jcim.1c00511>.
- (5) Voigt, J. H.; Elkin, C.; Madison, V. S.; Duca, J. S. Cross-Docking of Inhibitors into CDK2 Structures. 2. *J. Chem. Inf. Model.* **2008**, *48* (3), 669–678. <https://doi.org/10.1021/ci700428d>.
- (6) Birch, L.; Murray, C. W.; Hartshorn, M. J.; Tickle, I. J.; Verdonk, M. L. Sensitivity of Molecular Docking to Induced Fit Effects in Influenza Virus Neuraminidase. *J. Comput. Aided. Mol. Des.* **2002**, *16*, 855–869.
- (7) Schaller, D. A.; Christ, C. D.; Chodera, J. D.; Volkamer, A. Benchmarking Cross-Docking Strategies in Kinase Drug Discovery. *J. Chem. Inf. Model.* **2024**, *64* (23), 8848–8858. <https://doi.org/10.1021/acs.jcim.4c00905>.
- (8) Ricci-Lopez, J.; Aguila, S. A.; Gilson, M. K.; Brizuela, C. A. Improving Structure-Based Virtual Screening with Ensemble Docking and Machine Learning. *J. Chem. Inf. Model.* **2021**, *61* (11), 5362–5376. <https://doi.org/10.1021/acs.jcim.1c00511>.
- (9) Buccheri, R.; Rescifina, A. High-Throughput, High-Quality: Benchmarking GNINA and AutoDock Vina for Precision Virtual Screening Workflo. *Molecules* **2025**, *30* (16), 1–16. <https://doi.org/10.3390/molecules30163361>.
- (10) Dong, C.; Huang, Y. P.; Lin, X.; Zhang, H.; Gao, Y. Q. DSDPFlex: Flexible-Receptor Docking with GPU Acceleration. *J. Chem. Inf. Model.* **2024**, *64* (22), 8537–8548. <https://doi.org/10.1021/acs.jcim.4c01715>.
- (11) Sutherland, J. J.; Nandigam, R. K.; Erickson, J. A.; Vieth, M. Lessons in Molecular Recognition. 2. Assessing and Improving Cross-Docking Accuracy. *J. Chem. Inf. Model.* **2007**, *47* (6), 2293–2302. <https://doi.org/10.1021/ci700253h>.
- (12) McNutt, A. T.; Francoeur, P.; Aggarwal, R.; Masuda, T.; Meli, R.; Ragoza, M.; Sunseri, J.; Koes, D. R. GNINA 1.0: Molecular Docking with Deep Learning. *J. Cheminform.* **2021**, *13* (1). <https://doi.org/10.1186/s13321-021-00522-2>.
- (13) Heller, G.; Mo, Q. Estimating the Concordance Probability in a Survival Analysis with a Discrete Number of Risk Groups. *Lifetime Data Anal.* **2016**, *22* (2), 263–279. <https://doi.org/10.1007/s10985-015-9330-3>.

- (14) Ravindranath, P. A.; Forli, S.; Goodsell, D. S.; Olson, A. J.; Sanner, M. F. AutoDockFR: Advances in Protein-Ligand Docking with Explicitly Specified Binding Site Flexibility. *PLoS Comput. Biol.* **2015**, *11* (12), 1–28. <https://doi.org/10.1371/journal.pcbi.1004586>.
- (15) Trott, O.; Olson, A. J. AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading. *J. Comput. Chem.* **2010**, *31* (2), 455–461. <https://doi.org/10.1002/jcc.21334>.
- (16) Duca, J. S.; Madison, V. S.; Voigt, J. H. Cross-Docking of Inhibitors into CDK2 Structures.1. *J. Chem. Inf. Model.* **2008**, *48* (3), 659–668. <https://doi.org/10.1021/ci7004274>.

Chapter 5 – Identification and Structural Analysis of Activity Cliffs

5.1 Similarity- and Conformational-based Definition of Activity Cliffs

Activity cliffs represent a major challenge in quantitative structure–activity relationship (QSAR) analysis and virtual screening because they violate the fundamental assumption that structurally similar molecules exhibit similar biological activity.^{1,2} Structure-activity relationships are based on the principle that similar molecules exhibit similar activity. Activity cliffs are exceptions to this principle, consisting of highly similar compounds that display large differences in biological activity.³ These serve as instances of confoundment for any similarity-based VS model, which predicts ligand leads based on an SAR landscape of “smooth, rolling hills”, not a jagged series of cliffs.¹ As such, the exploration of activity cliffs and their causes are of great interest in medicinal chemistry. If the origins of these cliffs can be characterized, they can be properly handled by VS algorithms so their presence does not interfere with drug activity prediction. Activity cliffs also represent a promising area for medicinal chemists, given that small changes to drug scaffolds that result in large increases in drug potency are highly beneficial. Therefore, it is also of interest to characterize the origins of these in order to replicate their effects on novel drug compounds.

Activity cliffs remain largely difficult to resolve in the field of cheminformatics. Numerous machine learning approaches have attempted to predict activity cliffs, but all have met with limited success. Dablander et al. investigated whether machine learning models trained on known activity cliff examples could predict activity cliffs among structurally similar ligand pairs lacking experimental activity data. The models showed limited predictive performance, highlighting the inherent difficulty of identifying activity cliffs from molecular structure alone.⁴

Szostek and Szulczyk showed that successful machine learning prediction of activity cliffs depends critically on the QSAR metrics used for model training, as each metric captures different features of the structure–activity landscape and has distinct limitations.² Furthermore, machine learning models designed to identify activity cliffs must be trained specifically on activity cliff examples rather than complete QSAR datasets, as the abundance of non-cliff compounds reduces model sensitivity. Improving predictive models therefore depends not only on appropriate training data, but also on understanding the molecular features responsible for activity cliffs. To address this challenge, the activity cliff pairs identified from the QSAR analyses in Chapters 3 and 4 are examined to determine their structural origins.

The discontinuity of the QSAR landscape for CDK2 observed in Chapter 3 is almost certainly caused in part by the presence of activity cliffs within the dataset. The presence of activity cliffs within the CDK2 protein SAR landscape is well-known, and several works have attempted to characterize them.^{5,6} It is likely that the conformational flexibility of the CDK2 active site plays a significant role in the presence of activity cliffs,⁶ as the active site alone can bind several different inhibitor types with distinct active site binding modes, all of which induce different conformational changes in the active site to the DFG loop, α C helix, hinge region, and the activation loop.^{7,8} Two studies have attempted to capture the structural basis of activity cliffs using CDK2. Méndez-Lucio et al. identified activity cliffs using similarity-potency thresholds in ligand-based similarity metrics before analyzing the resulting pairs with protein–ligand interaction fingerprints. However, they found that differences in protein–ligand interactions explained only 44.6% of the identified activity cliffs, suggesting that additional structural factors contribute to their formation.⁶ Furtmann et al. assessed the ability of ensemble and ligand template-based docking predict activity cliffs in CDK2. Prediction accuracy was highest when

cross-docked ligands were positioned using the native binding pose of a structurally similar ligand, suggesting that accurate representation of ligand binding geometry is critical for reproducing activity cliffs.⁵ Building on the QSAR analyses presented in Chapters 3 and 4 and the active-site conformational classification developed in Chapter 2, this chapter investigates whether protein conformation contributes to the formation and prediction of activity cliffs in CDK2. Activity cliff pairs are first identified using a standardized thresholding protocol to address inconsistencies in activity cliff definition across the literature. Their distribution across the active-site conformational groups defined in Chapter 2 is then examined to determine whether particular receptor conformations are associated with activity cliff formation. The ability of GNINA to correctly rank activity cliff pairs is subsequently evaluated, followed by structural analyses using KLIFS interaction fingerprints and molecular modelling to identify the molecular features responsible for representative activity cliffs.

5.1.1 Similarity Metric-based Classification of Activity Cliff Thresholds

A major challenge in activity cliff analysis is the lack of a standardized definition for what similarity and potency differences constitute a cliff pair. Activity cliffs are commonly identified using similarity–potency thresholds, corresponding to the region of high structural similarity and large activity difference within the SAR landscape. However, many studies select these thresholds arbitrarily, making comparisons between studies difficult and potentially negatively influencing the quality of the resulting activity cliff.

One approach proposed by Guha and Van Drie, in which each ligand pairs similarity-potency dissension is captured by the Structure-Activity Ligand Index (SALI) value, which is calculated: $SALI_{i,j} = \frac{|A_i - A_j|}{1 - sim(i,j)}$ where A_i and A_j corresponds to the activities of the i^{th} and j^{th} molecules, and $sim(i,j)$ is the similarity coefficient between the two molecules.⁹ A large SALI

value corresponds to a significant activity cliff (large difference in potency and small difference in similarity coefficient). Thus, this metric was thought to be suitable for analysis of the ligand-based similarity metric data presented in Chapter 3, given that similarity was evaluated in the form of 2D and 3D similarity-based Tanimoto coefficients.

Within this study, an attempt to classify thresholds manually was made first (Table 2), but upon review, this was not useful for the purpose of the study, as it gives ambiguous results and does not maximize metric agreement for cliff pairs. Because the data distribution across the four metrics is very different, there was no cross-metric standardization option using this threshold definition methodology, making this system of cliff threshold classification unsuitable for further study. Next, an attempt was made to use SALI as a threshold cutoff, which is the ideal method in combination with the metric overlap analysis used further in this process, as it simplifies the threshold definition to a singular combined similarity-activity value. Given that EON ESP Tanimoto does not follow conventional Tanimoto bounds (it ranges -0.333 to 1), however, it became necessary to use a different threshold to describe the top activity cliffs across the similarity metrics used in this work. Based on the literature, $|\Delta pIC_{50}| \geq 2.0$ was chosen as an activity threshold – 2 orders of magnitude is a robust activity difference measure; some studies use only one.⁵ The 95th percentile was used as a beginning measure for similarity coefficient, as it offers a method of standardization within the differing data distributions found in the ligand similarity metric datasets. The results of this analysis were evaluated on the basis of SALI values (excluding EON ESP Tanimoto) at the minimum value to evaluate the significance of the minimum SALI-value similarity pair of structures. Using a 95th percentile threshold cutoff, the minimum similarity score for ROCS Color Tanimoto was 0.52, and for Morgan Fingerprint Tanimoto 0.38 (Table 2), which are very low similarity scores and appear to be structurally

divergent ligands upon visual analysis of the chemical structures. Due to the low structural similarity at the minima, the threshold was tightened to the 98th percentile (Figure 1), which gave a similarity minimum score for ROCS Color Tanimoto of 0.68, and a Morgan Fingerprint Tanimoto similarity minimum value of 0.54 (Table 2) – significant improvements to both these metrics. Across all four metrics, substantial agreement was observed in the activity cliff pairs identified, despite each metric capturing different aspects of ligand similarity. Overall, 42% of activity cliff pairs satisfied the activity cliff criteria in two or more similarity metrics (Figure 2),

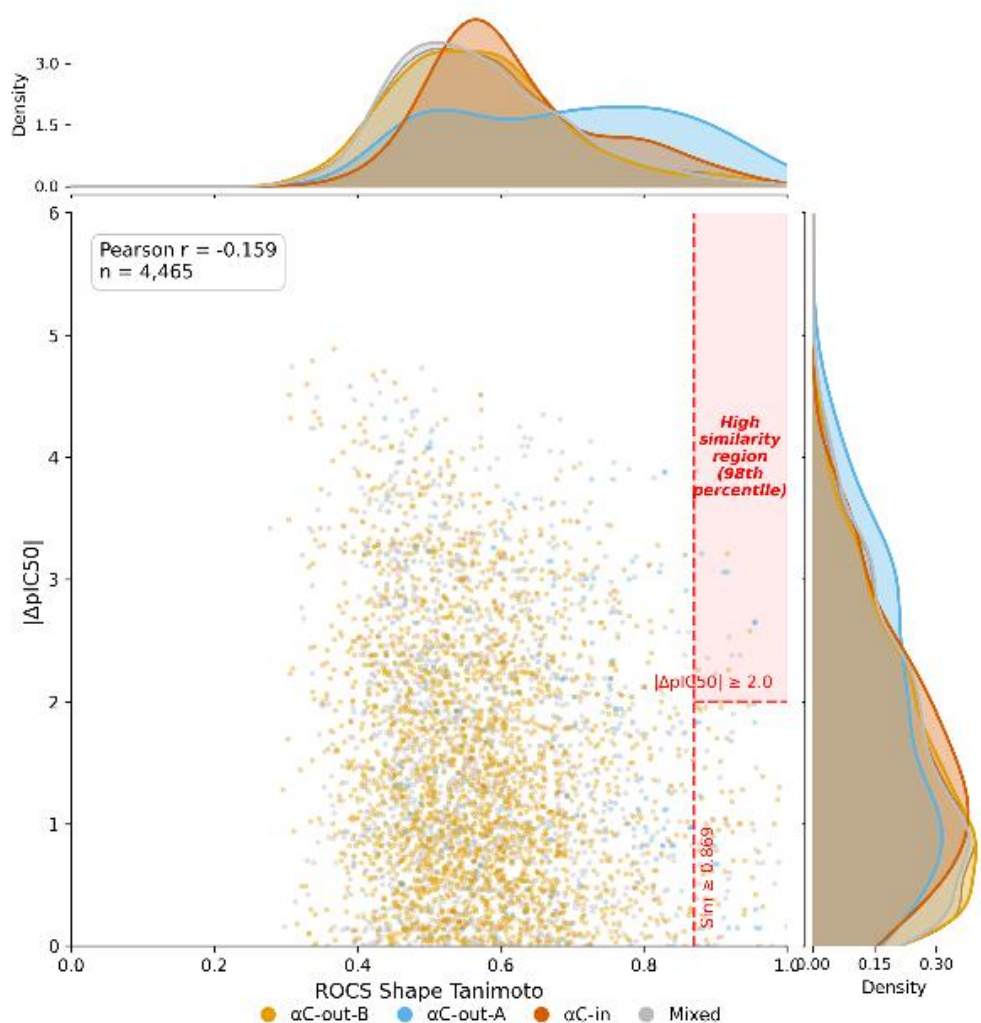


Figure 1. ROCS Tanimoto plot displaying the activity cliff threshold classification applied (98th percentile in similarity, $|\Delta pIC_{50}| \geq 2.0$).

indicating that the thresholds identify a common subset of ligand pairs recognized as activity cliffs across multiple representations of molecular similarity.

5.1.2 Active-Site Conformation-based Analysis of Activity Cliff Distribution

Using the ligand pairs which appear above the 98th percentile similarity and $|\Delta pIC_{50}| \geq 2.0$ potency thresholds as the designated “activity cliff set” for this study, the distribution of each conformational subset which appeared within the activity cliff set was evaluated. Because a modest preference for the native active-site conformation was observed in both the ligand-based

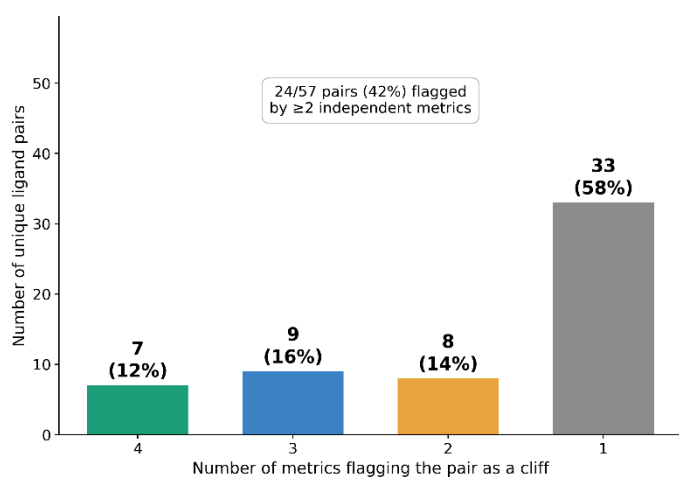


Figure 2. Bar chart showing the cross-metric agreement on assigned activity cliff pairs, both as an n-pairs (n = 57) and a percentage.

similarity analyses (Chapter 3) and the ligand–receptor docking analyses (Chapter 4), it was hypothesized that certain CDK2 active-site conformations may be associated with an increased propensity for activity cliff formation. The idea that activity cliffs will form with higher frequency within the

conformational groupings, and the

distribution of such groupings may show a preference for a specific active site conformation, and by proxy, subset of CDK2 ligand scaffolds, was explored next.

The distribution of activity cliffs across the CDK2 active-site conformational groups was evaluated using three complementary statistics, which were designated with the titles “Cliff Prevalence”, “Cliff Contribution”, and “Cliff Load”. “Cliff Prevalence” is the cliff rate within

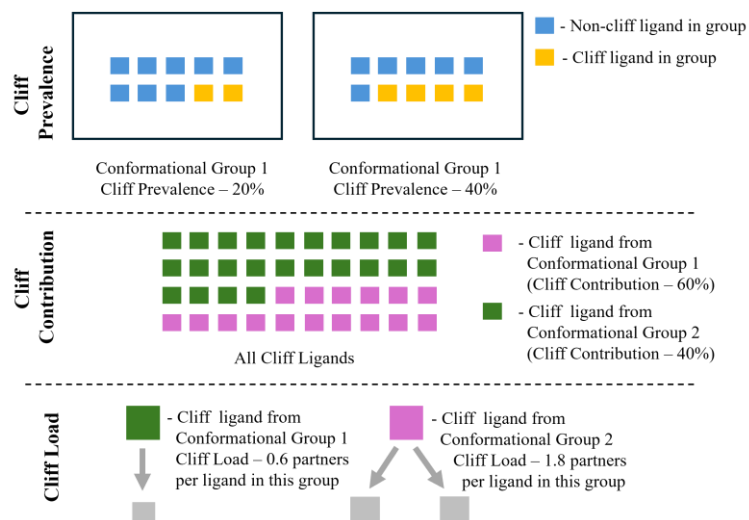


Figure 3. Image representing the statistics used to analyze the activity cliff distribution within the CDK2 dataset and its designated conformations.

each group – the number of cliff ligands in each group (α C-helix-out-A, α C-helix-out-B, α C-helix-in) divided by the total number of ligands in each group and expressed as a percentage. This answers the question: of this group’s own ligands, what fraction appears in at least one cliff pair? “Cliff

Contribution” is the percentage of cliff ligands which belongs to each set – the number of cliff ligands in the group divided by the total number of cliff ligands across all groups, expressed as a percentage. This answers the question: of all the ligands appearing in at least one cliff pair (any group), what fraction came from this group? “Cliff Load” is the cliff-pair load per group ligand – the total number of cliff-pair membership pairs contributed by the group divided by the total number of ligands in the group (all ligands in the group). This answers the question: how many cliff-pair relationships does a typical ligand from this group carry?

Table 1. Activity cliff active-site conformation-based membership for ligands, by similarity metric (98th-percentile threshold).

Metric	Conformation	n in group	n cliff ligands	Cliff Prevalence (%)	Cliff Contribution (%)	Cliff Load
ROCS Shape	α C-out-B	69	8	11.6	47.1	0.28
	αC-out-A	18	8	44.4	47.1	0.89
	α C-in	9	1	11.1	5.9	0.11
ROCS Color	α C-out-B	69	8	11.6	44.4	0.32
	αC-out-A	18	10	55.6	55.6	1.89
	α C-in	9	0	0.0	0.0	0.00
EON ESP	α C-out-B	69	16	23.2	55.2	0.49
	αC-out-A	18	12	66.7	41.4	1.83
	α C-in	9	1	11.1	3.4	0.11
Morgan ECFP4	α C-out-B	69	7	10.1	41.2	0.26
	αC-out-A	18	10	55.6	58.8	1.67
	α C-in	9	0	0.0	0.0	0.00

“Conformation” is the active-site conformational grouping of ligand-receptor complexes in Chapter 2 based on active site α C-helix RMSD. “Cliff Prevalence” measures cliff rate between conformational group (number of cliff ligands in group/total number of ligands in group $\times$ 100%). “Cliff Contribution” measures what % of the cliff set belongs to each group (number of cliff ligands in group/total number of cliff ligands in dataset $\times$ 100%). “Cliff Load” measures cliff-pair instances for each ligand within a conformational group (total number of cliff-pair members in group/total number of ligands in group), e.g. how many activity cliff pairs each ligand in the entire conformational group participates in.

Table 1 displays that the “Cliff Prevalence” and “Cliff Load” show a disproportionate favourability for α C-helix-out-A activity cliff pairs, which is rather surprising given that the group makes up only about 20% of the dataset, yet it accounts for about half of the activity cliffs across all metric activity cliff pair groups (Cliff Contribution). Across nearly all similarity metrics, more than 50% of α C-helix-out-A ligands participating in at least one activity cliff pair (Cliff Prevalence). Additionally, each ligand within the entire the α C-helix-out-A conformational set (not limited to activity cliff ligands) is shown to participate in between one (ROCS Shape Tanimoto, 0.87) and two (Color, ESP, Morgan FP) activity cliff pairs within the threshold cutoff.

There are several plausible explanations for this distribution. The most likely explanation is that this conformational subset is very small, which holds (n=18), and from a highly-related scaffold series from the same drug study.¹⁰ As such, the set would have high scaffold similarity and a wide range of activity.

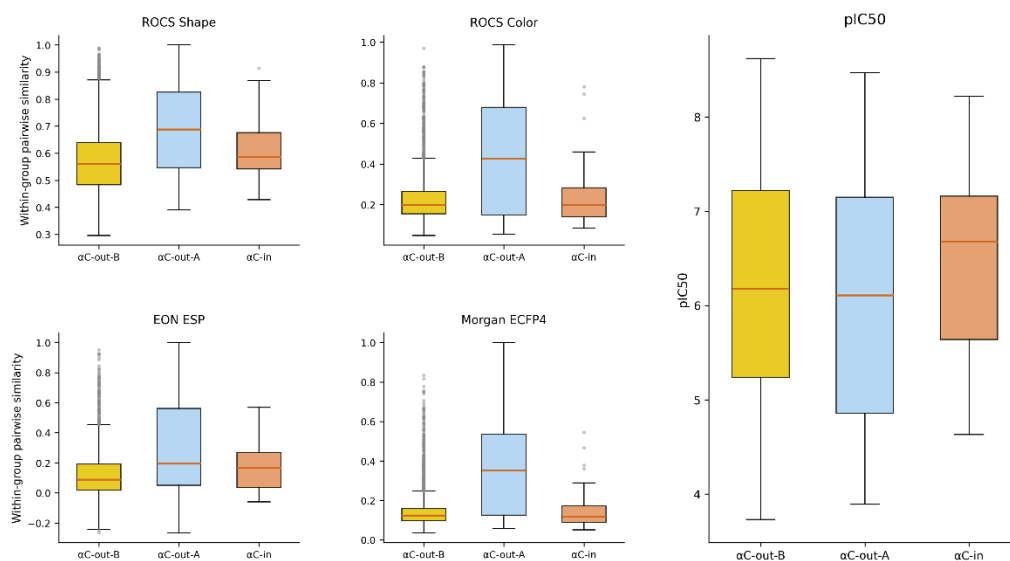


Figure 4. Distribution box plots for pairwise similarity metrics (ROCS Shape, ROCS Color, EON ESP, Morgan ECFP Tanimotos) and potency (pIC50) grouped by crystal-posed active-site conformational grouping (α C-helix-out-A, α C-helix-out-B, α C-helix-in).

As shown in Figure 4, the α C-helix-out-A potency distribution is comparable to the other groupings, while the pairwise structural similarity distribution is much larger in comparison to the other groups across all metrics, with a higher mean score. This indicates that the α C-helix-out-A has a higher mean pairwise similarity than the other groups (homogeneity), and it also has a much more diverse range of pairwise similarities (heterogeneity). The α C-out-A group occupies a higher region of chemical, shape, and electrostatic space overall, which significantly increases the rate at which its ligand pairs clear the similarity threshold used to define a cliff.

Because the α C-helix-out-A conformational group contained relatively few ligands, a Fisher's exact test was performed to determine whether the observed enrichment of activity cliff

ligands exceeded that expected by chance. Ligands identified by at least one similarity metric as participating in an α C-helix-out-A activity cliff were classified as cliff members. A 2×2 contingency table was then constructed comparing conformational group (α C-helix-out-A versus all other conformations combined) with activity cliff membership. Fisher's exact test, implemented in SciPy, was used to evaluate whether activity cliff membership was significantly associated with the α C-helix-out-A conformational state.

Table 2. Sensitivity of cliff detection, and probability of α C-out-A overrepresentation at various thresholds tested within the study (Fisher's exact test)

Threshold Scheme	ROCS Shape (sim min, <i>n</i> pairs, <i>p</i> -value)	ROCS Color (sim min, <i>n</i> pairs, <i>p</i> -value)	ESP EON (sim min, <i>n</i> pairs, <i>p</i> -value)	Morgan ECFP (sim min, <i>n</i> pairs, <i>p</i> -value)
Visually determined/ arbitrary	0.8, 42, <i>p</i> =0.1020	0.6, 73, <i>p</i> =0.0644	0.6, 40, <i>p</i> =0.0012	0.5, 34, <i>p</i> <0.0001
95 th percentile	0.784, 48, <i>p</i> =0.0608	0.521, 68, <i>p</i> =0.0346	0.402, 80, <i>p</i> =0.1168	0.379, 71, <i>p</i> =0.0059
98 th percentile	0.869, 18, <i>p</i> =0.0032	0.682, 28, <i>p</i> <0.0001	0.623, 34, <i>p</i> =0.0005	0.537, 24, <i>p</i> <0.0001

*Summary of tested thresholds for evaluating similarity metric definition of activity cliffs. All three tested threshold definitions listed here. 98th percentile determined to be optimal based on similarity score of lowest-ranked SALI ligand pairs. *p*-value corresponds to the results from the Fisher's exact test, which reports the probability of α C-out-A activity cliff overrepresentation occurring due to random chance as opposed true correlation.*

Table 2 shows that for all tested activity cliff thresholds, at least two metrics showed a statistically significant *p*-value (*p*<0.05) for the overrepresentation of α C-out-A within activity cliff ligand populations when tested via Fisher's exact test. The chosen "best" percentile (98th) for activity cliff analysis notably gave highly statistically significant *p*-values in all four metrics when tested via Fisher's exact test, which makes sense given the threshold is much tighter than the 95th percentile. The included activity pairs will therefore be of higher similarity and larger difference in potency, an area which Figure 3 showed to have a higher population density of α C-out-A ligands.

5.2 Examination of Scoring Function's Activity Cliff Prediction Capabilities

5.2.1 Analysis of Vina and GNINA's Prediction of Activity Cliffs

In the previous section, it was established that the CDK2 active site conformational grouping α C-out-A is significantly overrepresented within the activity cliff pairs population of this dataset at a 98th percentile similarity threshold cutoff and a $|\Delta pIC_{50}|$ threshold cutoff of 2.0. This observation is particularly interesting because the α C-out-A conformational group also exhibited the weakest docking score–potency correlations in Chapter 4 (Pearson $r = 0.22$ and 0.21 for Vina and GNINA, respectively), whereas substantially stronger correlations were observed for the α C-out-B group ($r = 0.46$ and 0.61) – essentially, the conformational group which accounts for much of the cross-docking dataset noise is the most cliff-dense region of the dataset. This observation suggests that molecular docking scoring functions have moderate success with predicting activity through docking score, but activity cliffs within the dataset are confounding this prediction. The observed association between activity cliff enrichment and poor docking performance suggests cliff-dense regions may reduce scoring function accuracy over the dataset. It is possible that identifying cliff-dense regions within a dataset and separating these different cases from a large dataset could be beneficial for improving SARs within CDK2 drug studies, although this association requires further exploration.

These findings emphasize that activity cliffs are not uniformly distributed throughout the CDK2 ligand space but are concentrated within specific conformational subsets. Consequently, docking performance is influenced not only by ligand diversity but also by the distribution of activity cliffs within the dataset. Identifying conformational subsets enriched in activity cliffs may therefore improve the interpretation of docking-based SAR analyses. More broadly, the RMSD-based active-site classification developed in Chapter 2 may provide a practical

framework for identifying ligand subsets that are inherently prone to activity cliff formation and, consequently, more difficult to model using conventional docking scoring functions.

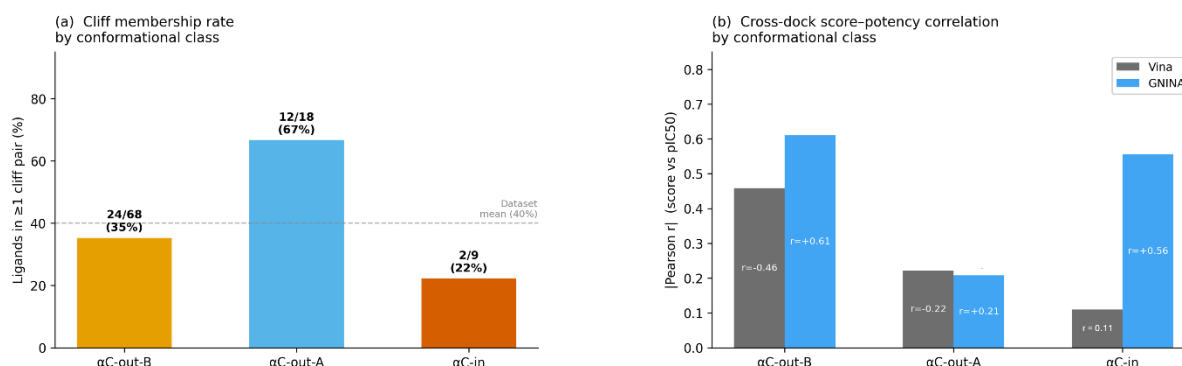


Figure 5. a) Shows the percent of ligands within each conformational subset which has at least one ligand participating in an activity cliff pair. b) shows the Vina and GNINA Pearson r values for mean ligand score when ligands were docked into every receptor within that conformational subset.

In Chapter 5, Vina and GNINA achieved pairwise ranking accuracies of approximately 60% when comparing cross-docked ligand pairs within the same receptor. However, the previous section demonstrated that activity cliffs are enriched within the α C-out-A conformational group, which also exhibited the weakest docking score–potency correlations (Figure 5b). The central question addressed in this section was whether the ability of molecular docking scoring functions to correctly rank activity cliff pairs depends on the receptor conformation used for docking. If docking performance deteriorates when ligands are evaluated in non-native conformations, it would suggest that active-site geometry plays an important role in the accurate prediction of activity cliffs. Cross-docking provides a particularly stringent test of this hypothesis because it introduces receptor conformational variability that more closely reflects the conditions encountered during prospective virtual screening. Because activity cliff pairs had already been identified using the similarity-based analyses and classified according to the active-site conformational groups defined in Chapter 2, the effect of receptor conformation on activity cliff ranking could be evaluated directly.

For each activity cliff pair defined by the similarity thresholds, the more potent (stronger binding) and less potent (weaker binding) ligand were determined. The cross-docked results were used, dividing the ligands by the receptor they were docked into based on the three conformational groups: α C-out-A, α C-out-B, α C-in. Each ligand pair examined was docked into the same target receptor structure (as in Chapter 4 Section 4.1.3). Then the ability of GNINA and Vina to correctly rank the more potent CDK2 ligand within the entire dataset in comparison with the classified activity cliff pairs at the 98th percentile threshold was assessed.

Table 3. Vina and GNINA Activity cliff pairwise ranking prediction in activity cliffs compared to the remainder of the CDK2 crystal pose dataset

Scoring Function	n activity cliff pairs	Pairwise Ranking Prediction (cliffs)	n non-cliff pairs	Pairwise Ranking Prediction (non-cliffs)
Vina	57 (1.3%)	46.5%	4,408 (98.7%)	59.9%
GNINA	57 (1.3%)	53.2%	4,408 (98.7%)	59.3%

“Pairwise ranking accuracy” reflects the % of pairs which correctly capture the directionality of a ligand potency pairwise comparison with their molecular docking score. The largest potency value is assigned the top ligand position and smallest the bottom within the pair. If docking score agrees with this, it is a positive result. If it does not, its negative. Pairwise ranking prediction is calculated as n pairs within category/n total pairs x 100% – 50% is considered “expected by chance”, anything above is significant.

The results of this study show that GNINA and Vina struggle to rank most activity cliffs in accordance with their potency (Table 3). Both scoring functions only successfully rank ligands according to potency about 50% of the time, which indicates the ranking is insignificant as this is the probability which is observed if ranking is assigned by chance. In contrast, ranking the CDK2 dataset ligand pairs via both scoring functions with the identified cliff pairs removed shows a moderate increase in predictive ranking power (about 10%).

Vina achieved a pairwise ranking accuracy of 46.5% for activity cliff pairs, while GNINA achieved 53.2%. Both values are close to the 50% accuracy expected under random ranking, indicating little or no predictive ability for activity cliff pairs. These results suggest it is possible to increase the ability of GNINA and Vina to correctly rank CDK2 ligand pairs by excluding

activity cliff pairs from the dataset. Conversely, GNINA and Vina are incapable of ranking activity cliff pairs with any predictive confidence over random chance. These statistics imply a significant problem to address within a VS pipeline which would be intended to predict new drug ligands for CDK2 studies – if activity cliffs are present, they cannot rank them in accordance with potency with any confidence, although GNINA performs slightly better.

5.3 Structural Analysis of α C-helix-out-A Activity Cliff Pairs

5.3.1 Structural Analysis of CDK2 Ligands using KLIFS Interaction Fingerprints

CDK2 and other proteins within the kinase family perform somewhat similar functions within biological systems (phosphorylation), and, as such, have moderately conserved structures and protein sequences. Because of this, a comprehensive database of kinase protein structural features called KLIFS (**K**inase **L**igand **I**nteraction **F**ingerprints and **S**tructures) has been created and curated, which contains structural kinase-ligand information derived from human and mouse kinase catalytic domains deposited on the PDB.^{11,12} These kinase-ligand complexes have been simplified into molecular interaction fingerprints (IFPs) which display the interactions of 85 key binding residues in the kinase active site, focusing on the front cleft, hinge, and DFG-loop areas, and their interaction population and types with the bound ligand.^{11,12} Information about kinase active-site conformation can also be found within this database, although the conformational classification is broad. KLIFS IFPs are encoded as an 85 x 7-bit string which corresponds to the 85 residues of interest and the seven interaction types which can be characterized at each residue, such as hydrophobic contacts, hydrogen bond donor/acceptor, or ionic interactions.¹³ These bit-string IFPs can then be used to compare ligand binding profiles within proteins such as CDK2 and potentially relate differences in ligand-protein binding to biological activity.

KLIFS Interaction Frequency Comparison by Conformational Group
(colour = proportion of structures (0-1); // = residues with no interactions omitted)

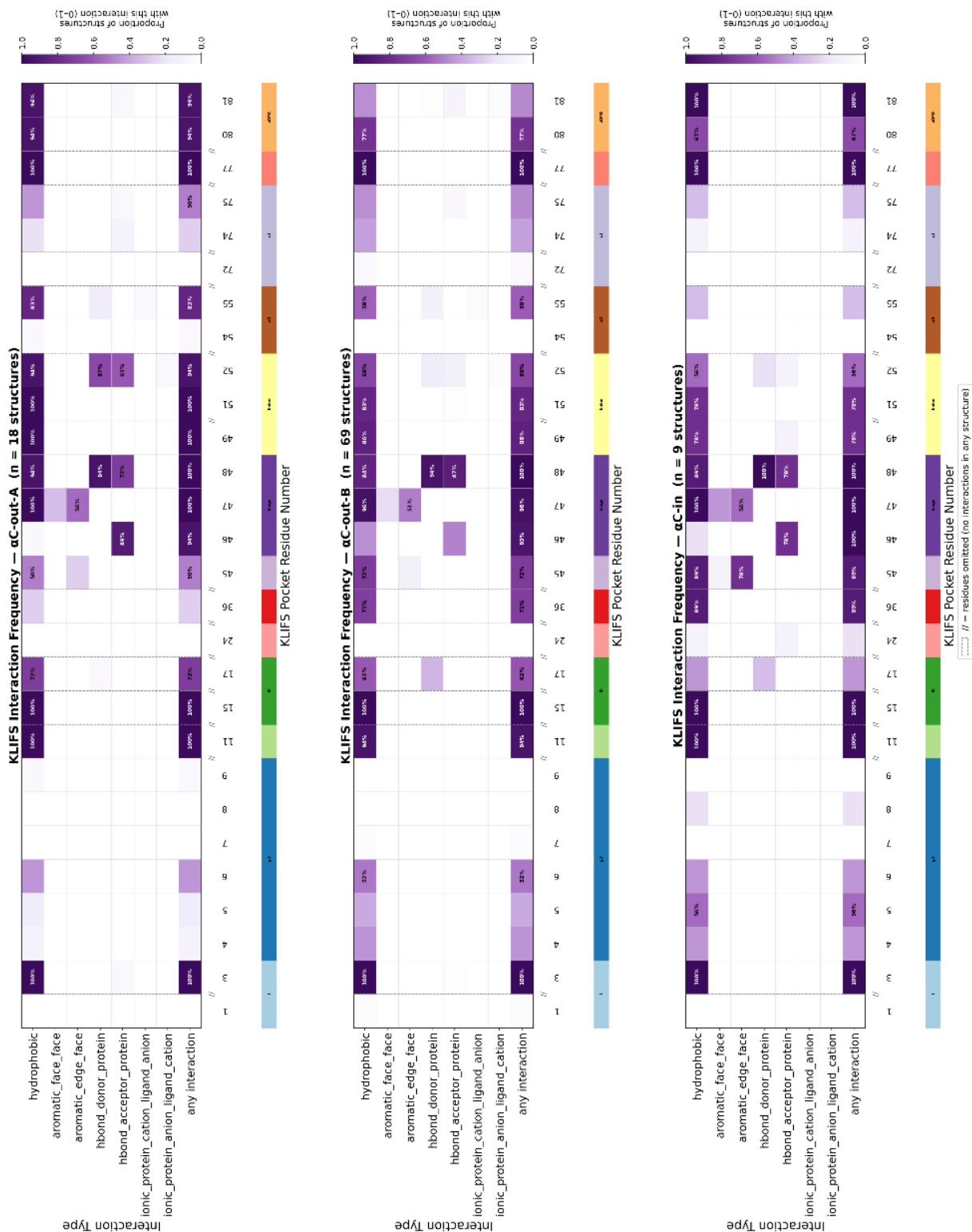


Figure 6. KLIFS database IFP heatmaps for all CDK2 ligands within the crystal dataset. Grouped by conformational grouping. Color gradient in heatmaps corresponds to frequency of interaction type at each residue (%) based on n structures in each conformational group. Blank spaces and // denote gaps in the KLIFS residue sequence due to no occupancy – dropped due to whitespace management.

The analysis performed by Méndez-Lucio et al. found that about 46% of all activity cliffs could be explained by interaction fingerprint differences, also known as interaction cliffs.⁶ Given that α C-out-A ligands showed a propensity to form activity cliff pairs, it made sense to examine the KLIFS interaction fingerprints (IFPs) belonging to the CDK2 dataset to determine whether there is a receptor-ligand interactional basis for the activity cliffs in α C-out-A.

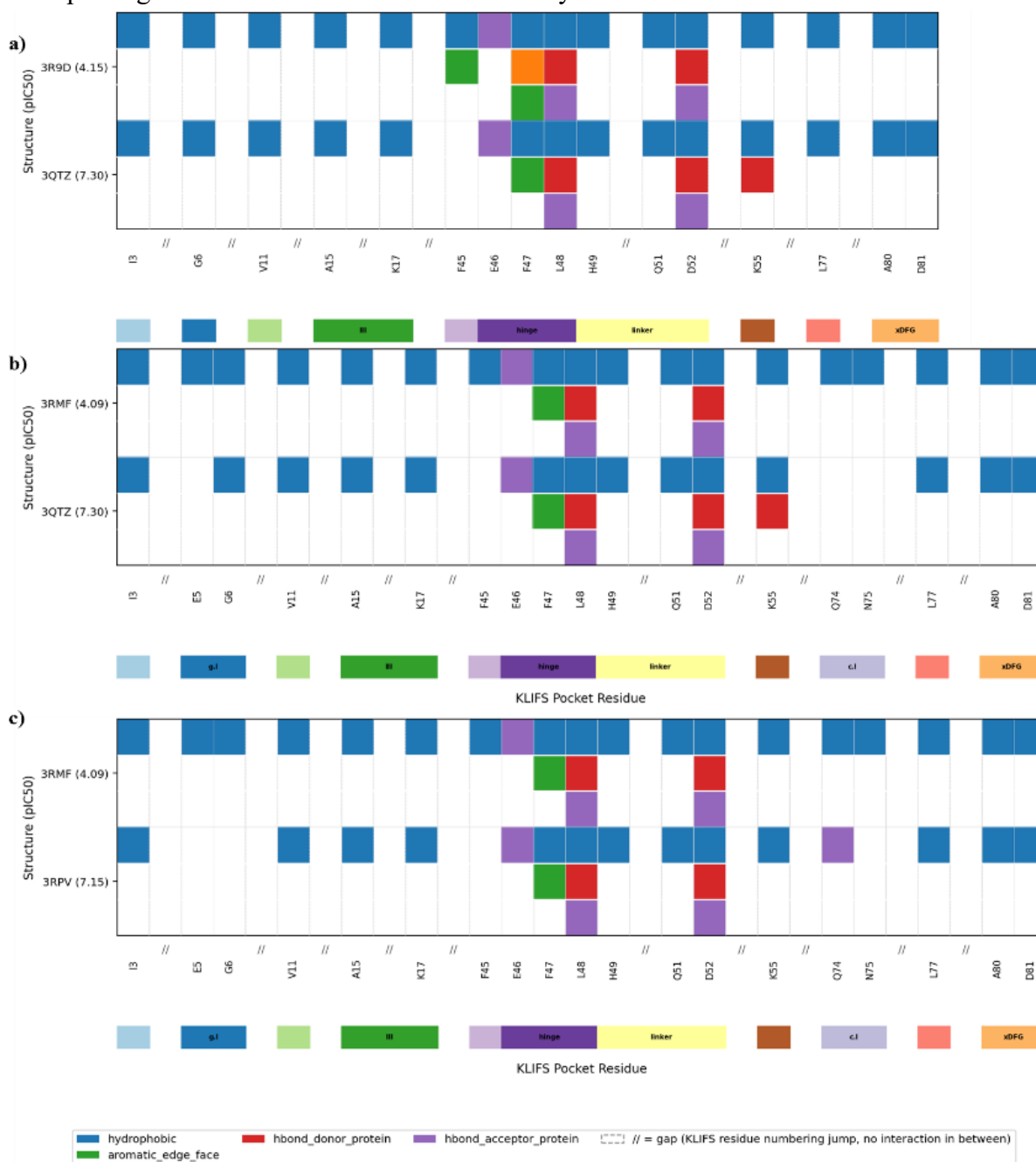


Figure 7. Pairwise comparison of KLIFS IFPs for α C-out-A pairs 3RPV/3RMF (top), 3QTZ/3RMF (middle), 3QTZ/3R9D (bottom). Lower ligand has higher pIC50 value. // denotes gap in KLIFS sequence due to unoccupancy.

A rough analysis of the heatmap in Figure 6 provides possible basis for the frequency of activity cliff formation observed within the α C-out-A conformation. The heatmap shows that the CDK2 binding is largely governed by hydrophobic contacts. A quick comparison shows that α C-out-A has 11 residues with >90% frequency within ligand-receptor complexes for hydrophobic contacts occurring at that residue. In comparison, α C-out-B only has 5 residues which have hydrophobic contacts above this threshold frequency, and α C-in only has 6 residues which meet this condition (3 more at 89% frequency, though). Based on this data, it could be reasoned that α C-out-A has a highly conserved hydrophobic binding pattern within this region, and the presence or absence of one of these contacts leads to a large change in potency for these ligands. These conclusions must be taken lightly because the small group size of α C-out-A compared to α C-out-B may result in more conserved interactions by nature.

To further investigate these findings, the interactional basis for several α C-out-A activity cliff pairs was examined. The ligand pairs chosen (3RPV/3RMF, 3QTZ/3RMF, 3QTZ/3R9D) were chosen because they were all flagged across multiple similarity metrics (3-4), and had high SALI values. It was additionally observed that several activity cliff pairs had common anchors (weaker ligand, 3RMF and 3R9D). The properties of these activity cliff nodes were of interest, so a set of pairs with one of these comparisons was selected.

Table 4. PDB ID references for selected activity cliff pair ligands

PDB ID	Source
3RPV	https://doi.org/10.2210/pdb3RPV/pdb
3RMF	https://doi.org/10.2210/pdb3RMF/pdb
3QTZ	https://doi.org/10.2210/pdb3QTZ/pdb
3R9D	https://doi.org/10.2210/pdb3R9D/pdb

All activity cliff ligands from the same paper (Schonbrunn et al.).¹⁰

First, pairwise KLIFS IFPs were generated for each pair (Figure 7). The KLIFS heatmaps showed completely opposite behaviour than what was expected based on the conclusions made about the conformationally-sorted global

heatmap from the CDK2 dataset (Figure 6). In all

three cases, the lower-potency ligand was shown to have more protein-ligand hydrophobic contacts than the higher-potency ligand. The 3RPV/3RMF pair alone showed that the more potent ligand also formed a hydrogen bond with the protein backbone which the lower potency ligand did not. Because of these counterintuitive results, the ligand-protein structural binding complex was investigated using PyMOL.¹⁴

5.3.2 Structural Analysis of CDK2 α C-out-A Ligand Activity Cliff Pairs in PyMOL

All three ligand pairs of interest were visualized using PyMOL.¹⁴ The discordant residues noted between them were flagged and the R-group to ligand interactions visualized and labelled. The solvent accessible surface area (SASA, calculated as the percent of the unbound ligand which is inaccessible to solvent upon binding) and active site cleft packing (percent of atoms in close contact, calculated as the percent of heavy atoms in the ligand which are within 3.6 Å of a protein atom) were calculated to assess active site binding properties. Hydrogen bonds between protein and ligand were assigned using a ≤ 3.2 Å cutoff between candidate ligand atoms and potential donor or acceptor protein atoms (3.2 Å used because crystal structures did not have resolved hydrogens and thus bond angle could not be assessed). Potential hydrogen bond donors

or acceptors within the ligand that do not form a hydrogen bond with the protein based on the threshold cutoff are designated as “unsatisfied polar atoms”.

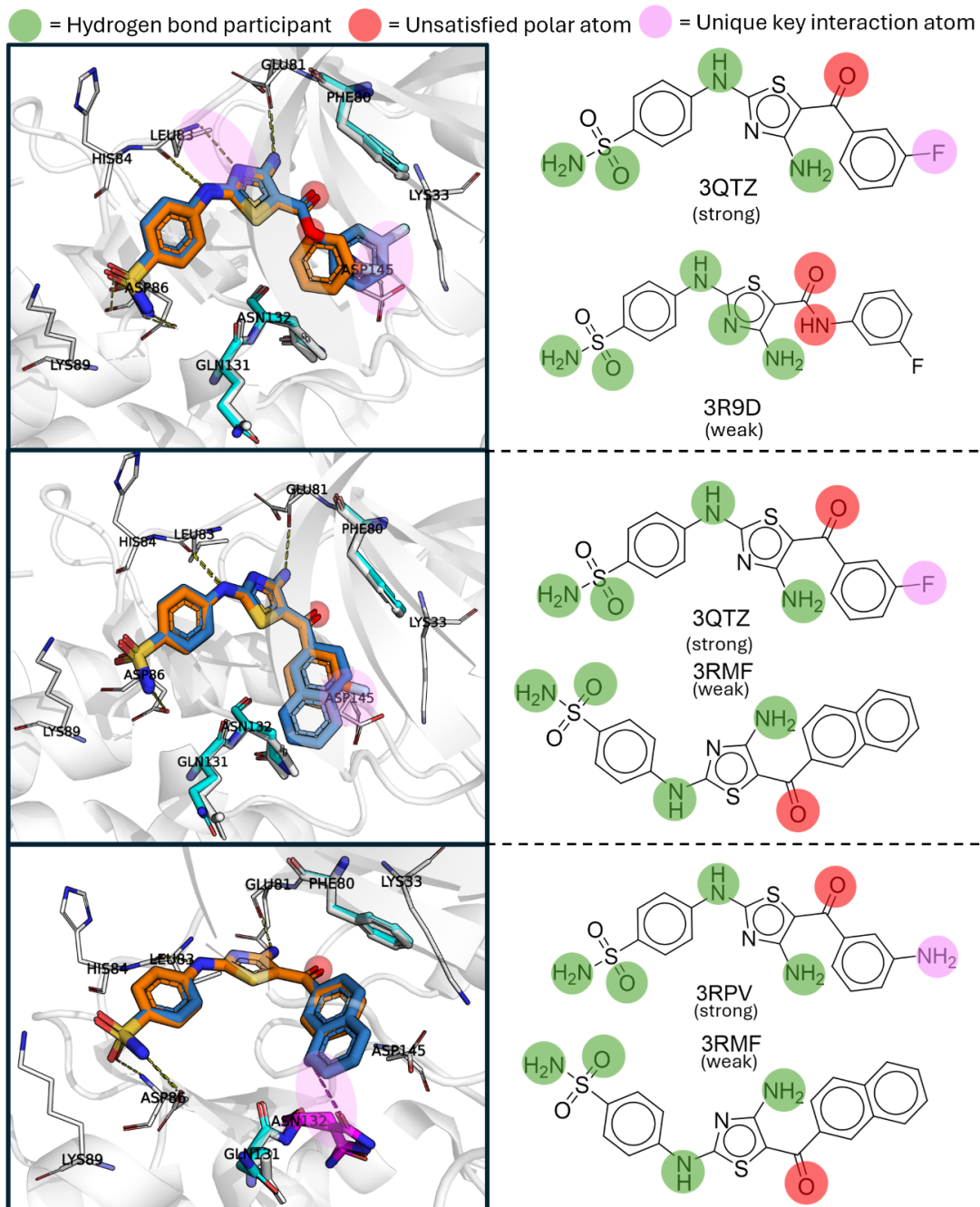


Figure 8. Visualization of α C-out-A activity cliff pair ligands (<https://doi.org/10.1021/jm301234k>) within their aligned protein receptors using PyMOL.^{10,14} On the ligand scaffold, green circles denote hydrogen bond participants, red circles denote “unsatisfied atoms” (atoms which are not hydrogen bonded within the active and are energetically penalized for desolvation) and purple circles denote atoms which participate in unique interactions which contribute to the molecule having a higher potency.

The active site visualization and evaluation of the three selected ligand pairs using PyMOL yielded very surprising results.¹⁴ All six ligands retained the KLIFS-documented hinge contact: hydrogen bonding at the GLU81 (46E in KLIFS), LEU83 (48L in KLIFS), and both donor and acceptor bonds with ASP86 (52D in KLIFS). Upon hydrogen bond analysis using the ≤ 3.2 Å threshold in PyMOL, the hinge binding network was found to agree with the KLIFS definition of the hydrogen bonding system. Below, divergences between KLIFS IFPs and their ability to predict activity cliffs will be explored. Similarly, several ligand structure-dependent effects which may lead to the large increase in potency across similar ligand pairs which were identified within the crystal complexes using PyMOL will be explored, and their capacity to cause activity cliffs will be investigated.

3QTZ (strong) vs. 3R9D (weak). The KLIFS heatmap comparisons in Figure 7 show that these two ligands differ only in one place significantly – a hydrogen-bonding interaction at LYS89 (55K in KLIFS) which is present in 3QTZ and not in 3R9D. This difference alone may be an adequate description for the pIC₅₀ difference of 3.21. Upon structural analysis of this pair using PyMOL, however, it was found that the difference between the participating ligand sulfonamide group and the LYS89 amino group was shown to be 3.45Å in 3QTZ and 3.43Å in 3R9D, respectively. The interaction distance between these two groups is negligible; additionally, they imply that 3R9D should be slightly more favoured to bind than 3QTZ. Based on this analysis, both will form a weak hydrogen bond with LYS89. Clearly then, this distinction is not enough to describe the pIC₅₀ difference upon structural examination. Further structural analysis showed that 3R9D engages LEU83 in a bidentate interaction where its thiazole ring nitrogen (an aromatic N) accepts a hydrogen bond from LEU83's backbone amine. This second interaction was determined due to protein-ligand proximity (3.13Å). This interaction was not observed in

3QTZ's structure, where this interaction was predicted to occur outside the threshold range (3.36Å), which is the range for a looser hydrogen bond but does not pass the more rigorous structural analysis performed here. This disagrees with the KLIFS IFP, which shows both ligands having the same bidentate interactions. According to these PyMOL results, the weaker ligand forms an extra bond beyond the four formed in the strong ligand, which seems counterintuitive – however, it is possible that the quantity of interactions is not the defining predictor of potency, rather the magnitude - or that other factors within this ligand binding event beyond observed hydrogen bonding are more significantly influencing binding affinity.

Two large structural differences separated the ligands. One difference observed between the 3QTZ and 3R9N was in the location of the fluorophenyl rings – the 3QTZ fluorine was rotated clockwise relative to the 3R9N ring due to the amide connectivity present in 3R9N. This led to a curious dipole-dipole orthogonal interaction taking place between the 3QTZ fluorine and the ASP145 sidechain (C – F----C=O interaction), which is supported by the distance between the two: 3.42Å, which is within the optimal range for this type of interaction (3.6-3.7Å, Figure 8a).¹⁵ This interaction is sometimes called an orthogonal fluoro-multipolar interaction and is a known targeted interaction within medicinal chemistry to improve potency; in a study by Xing et. Al, replacements of hydrogen with fluorine led to a 2.2 fold potency increase within matched molecular pairs on average.^{15,16} The same interaction in 3R9N was at a distance 5.01Å – too far to be a viable interaction. If this interaction is truly taking place at ASP145 this would also account for some of the gained potency of 3QTZ: ASP145 is a key residue of the main catalytic “DFG” loop within the CDK2 active site, and this residue has been shown to significantly contribute energetically to CDK2 ligand binding.¹⁷ While both ligands have the potential to form this interaction, both having the same fluorophenyl ring in their scaffolds, the amide linker in

3R9N places the fluorine too far away within the active site from ASP145, likely resulting in the observed sharp decrease in potency. The second difference between the two is the unsatisfied, or unbonded, polar amide nitrogen. All the ligands within this case study have an unbonded carbonyl oxygen which will incur a desolvation penalty upon binding, but 3R9N is the only ligand which has an additional unsatisfied polar atom in its amide nitrogen. This will likely result in a higher desolvation penalty upon binding, significantly contributing to the overall decreased potency observed for 3R9N. Both the ligand solvent-accessible surface area and the active site ligand packing were found to be relatively similar within both ligand-receptor complexes, so these do not contribute meaningfully to the observed potency difference (Table 5).

3QTZ (strong) vs. 3RMF (weak). According to the KLIFS IFP pairwise comparison in Figure 7, the weaker ligand (3RMF) makes additional hydrophobic contacts at residues GLU12, PHE80, GLN131, and ASN132. Using the PyMOL threshold hydrogen bond definition, the four conserved hinge hydrogen bonds described previously were again observed within the protein-ligand complex. The KLIFS IFPs showed a distinct hydrogen bond with LYS89 in 3QTZ and not present in 3RMF, which could cause the activity cliff if it were a significant interaction.

However, when the distances between the ligand's participating sulfonamide oxygen and the LYS89 nitrogen were measured, it was found that the distances between the two were 3.34Å for 3RMF and 3.45Å for 3QTZ, respectively. Based on this structural analysis, KLIFS IFP has mislabelled the hydrogen bond of this pair, as it is also being made in 3RMF, and appears to be stronger. KLIFS also stated the same bidentate interaction with LEU83 with the thiazole ring nitrogens – this was verified by distance and found to be a loose definition of hydrogen bonding, with the thiazole ring nitrogen in 3RMF 3.27Å away from the LEU backbone nitrogen, and this atom 3.36Å away in 3QTZ.

There were again two major differences between the two ligands which serve as potential causes of the observed large pIC₅₀ difference. First, the orthogonal fluoro-multipolar interaction takes place between 3QTZ fluorine and ASN145 of the DFG loop in the CDK2 active site (Table 5), an energetically favourable interaction which can result in potency differences similar to what is observed between these two ligands given that 3RMF has no fluorine within its structure.^{15,16} Not only that, but the bulkier bicyclic substituent in 3RMF will likely be sterically excluded completely from the ASP145 “pocket” and forced to sit within a looser region of the active site (Figure 8b). The second major difference is observed within the active site packing efficiencies of the two ligands. The percentage of heavy atoms in close contact within the active site for 3RMF was 48.3%, while the percentage of packing efficiency for 3QTZ was found to be 69.2% (Table 5). This is a major difference in active site packing, and supports the hypothesis that 3QTZ is forming more tight-packing favourable interactions within the active site than 3RMF, contributing to an overall much-larger potency. This does not necessarily support the KLIFS increase in hydrophobic contacts for 3RMF but neither does it disprove this idea – it is possible that 3RMF forms very weak and loose hydrophobic interactions within the active site which are then over-weighted as true hydrophobic interactions which contribute to binding.

3RPV (strong) vs. 3RMF (weak). According to the KLIFS IFP pairwise comparison in Figure 6, the weaker ligand (3RMF) has additional hydrophobic contacts at residues GLU12, GLY13, PHE80, GLN131 and ASN132. The stronger ligand is distinguished with a hydrogen bonding interaction occurring at GLN131 (Q74 in KLIFS). This is only time within the case studies observed where a change in a KLIFS IFP reflects the directionality of potency in the activity cliff. Using the PyMOL threshold hydrogen bond definition, the four conserved hinge hydrogen bonds described previously were again observed within the protein-ligand complex.

There were again two differences in the ligand structures which could potentially cause the observed pIC50 difference between the pair; the first was a gained hydrogen bond in 3RPV to the ASN132 residue within the CDK2 active site due to the aniline NH₂ hydrogen bond-donor in 3RPV which can access the ASN132 carbonyl hydrogen-bonding acceptors (Figure 8c). This is the only activity cliff mechanism within the case studies where an isosteric replacement directly results in the introduction of a new bonding interaction – the scenario which is typically the most intuitively expected. It should be noted that the KLIFS IFP for this interaction showed the hydrogen bond taking place between the GLN131 rather than the ASN132, but the ASN132 was in much closer proximity to the ligand within the crystal pose: 2.93Å, than GLN131. 3RMF cannot form a hydrogen bond at this location within the active site because it has no polar groups at this location to do so. The second significant difference between the cliff pair was the ligand active site packing once more, with 3RPV having a heavy-atom packing percentage of 61.5% in comparison to 3RMF's 48.3%, meaning 3RPV is likely forming more close-packing contacts within the binding site which lowers the energy of binding significantly.

Table 5. Structural comparison chart for α C-out-A ligand activity cliff pair properties.

Protein-Ligand Properties	3QTZ vs. 3R9D (higher vs. lower)	3QTZ vs. 3RMF (higher vs. lower)	3RPV vs. 3RMF (higher vs. lower)
pIC50 (higher ligand/lower ligand)	7.30 / 4.09	7.30 / 4.09	7.15 / 4.09
PyMOL Hinge contact (LEU83/ GLU81/ ASP86)	Identical in both but gains an extra bidentate LEU83 H-bond as an N-acceptor	Identical in both	Identical in both
Discordant residues (defined in KLIFS)	PHE80, PHE82, LYS89	12, PHE90, LYS89, GLN131, ASN132	12, 13, PHE80, GLN131, ASN132
KLIFS-defined H-bonds	LEU83 (x2), GLU81, ASP86 (x2), LYS89	LEU83 (x2), GLU81, ASP86 (x2), LYS89	LEU83 (x2), GLU81, ASP86 (x2), GLN131
PyMOL H-bonds (higher ligand/lower ligand)	5 / 4	4 / 4	5 / 4
Unsatisfied polar atoms (higher ligand/lower ligand)	1 / 2	1 / 1	1 / 1
F-Asp145 distance (higher ligand/lower ligand)	3.42Å / 5.01Å	3.42Å / 7.99Å	N/A – no fluorine in either ligand
Interaction contact changes	Low-potency ligand gains extra H-bond bidentate LEU83 interaction, high potency gains F-ASP145 interaction	Low-potency ligand gains hydrophobic contacts, high potency gains F-ASP145 interaction	Low-potency ligand gains hydrophobic contacts; high potency ligand gains H-bond at ASN132
% atoms in close contact (higher ligand/lower ligand)	69.7% / 63.0%	69.2% / 48.3%	61.5% / 48.3%
% SASA buried (higher ligand/lower ligand)	87.6% / 85.6 %	87.6% / 88.6%	87.2% / 88.7%
Dominant mechanism for higher potency ligand	F-ASP145 contact gain and one less unsatisfied atom	Better packing efficiency within active site and gain of F-ASP145 contact/pocket interactions	Gain of extra ASN132 H-bond, better active site packing efficiency

All ligand structural features (atom distances/bond lengths) determined using PyMOL. Note that none of the crystal-posed proteins had hydrogens, so hydrogen bonds were inferred by distance threshold (≤ 3.2 Å). Discordant residues are residues in KLIFS IFPS which are flagged to have an interaction at a residue in one ligand-receptor complex but not the other within the pair. Unsatisfied polar atoms are atoms which have the ability to form hydrogen bonds but do not appear to be in close proximity to a potential partner; thus incurring a desolvation penalty upon binding. % atoms in close contact refers to the packing efficiency of the ligand within the active site based on the heavy atoms in the ligand – the more they are contained within, the better the packing efficiency. % SASA (solvent accessible surface area) buried refers to the percentage of the ligand that is buried within the active site and cannot make solvent contacts.

5.3.3 Conclusions about CDK2 α C-out-A Activity Cliff Pairs

From the three activity cliff case studies discussed within this section, several broad conclusions can be made. First, dataset-level KLIFS IFPs are unable to predict the mechanism of an activity cliff with consistent accuracy, for several reasons. It was shown that several of the ligand-receptor interactions observed within the IFP were not observed upon in-depth structural analysis of the ligand, or were shown to be stronger within the ligand of the pair which did not have these interactions present in its IFP. As well, KLIFS IFPs mis-numbered one of the key binding interactions occurring that actually defines an activity cliff between a pair – clearly there are inconsistencies within the IFPs which lead to dataset noise in comparing activity cliffs as performed in Méndez-Lucio's study.⁶ Beyond this, KLIFS IFPs cannot provide the magnitude of these interactions, only their presence or absence, which could be an important distinction within the activity cliff pairs, given that the weaker of the two was shown to make more hydrophobic contacts within the active site in all cases. These were likely weak and not as favourable as the close pocket packing observed in the stronger potency ligands – differences in magnitude which KLIFS IFPs fail to capture. Finally, a conclusion about the activity cliffs themselves: the same CDK2 ligand does not lose or gain potency over its paired ligand by the same mechanism across its different pair partners. This adds a significant layer of complexity to the prediction of activity cliffs within CDK2. In the ligands studied, 3RMF was found to be weaker than 3QTZ due to poor close-packing, and an inability to form a key orthogonal multipolar interaction between fluorine and ASP145. But this is not the same reason for 3RMF's weaker potency in comparison to 3RPV – the close-packing penalty is the same, but 3RMF does not make a key ASN132 hydrogen bond that 3RPV can. Similarly, while 3QTZ is the more potent ligand compared to 3RMF because of efficient active site packing and favourable electrostatic interactions within the active site, this is not the full explanation for how it is stronger than 3R9D. While the orthogonal

multipolar explanation holds true, 3QTZ also incurs less energetic desolvation penalty upon binding due to only having one unsatisfied polar atom, while 3R9D has two. Furthermore, the additional hydrogen bond in 3R9D over 3QTZ shows that more protein-ligand contacts in the IFP do not necessarily indicate an overall stronger ligand potency.

While this is a very anecdotal dataset due to size limitations, the findings are consistent with the idea that activity cliffs are resistant to a global scale SAR relationship definition, and instead must be treated on a case-by-case basis, or at least within small structurally restricted ligand series. Given that these activity cliff pairings were all taken from CDK2 crystal complexes with highly similar active site conformation based on RMSD clustering and still display significant variation in binding mode and activity cliff contributing mechanisms, it is clear that there are few common descriptors which might be used to predict SAR of these activity cliffs. It is obvious that the α C-out-A activity cliff pair group is not caused by a series of singular pharmacophoric swaps at one position in the ligand scaffold, but rather a series of nodes within a complex activity network where large potency differences may be gained and lost in many different ways in the same ligand scaffold, due to the highly flexible nature of the CDK2 active site. Thus, the idea of predicting the biological activity of such structures based on similarity becomes significantly more challenging.

5.4 Discussion and Conclusions

In this chapter, a standard protocol for defining CDK2 activity cliff thresholds was proposed and implemented. Applying this 98th percentile threshold was found to have enriched the activity cliff set with pairs which were agreed upon within multiple ligand similarity metrics, proving the surviving activity cliff set to be robustly defined and representative of the CDK2 activity cliff population. Furthermore, it was identified that within the CDK2 activity cliff

population, ligands which were bound within the α C-out-A showed a markedly increased participation in activity cliffs beyond the other two conformational groups, accounting for about 50% of the activity cliff pairs identified by all of the similarity metrics, though the group contains less than a third of the ligand population of α C-out-B ($n = 18$ vs. $n=68$, respectively). This was attributed to the increased mean similarity within the group above the other two groups, which naturally leads to a higher activity cliff pair occurrence within the subset. A comparison of the results of the cross-docking of the CDK2 crystal-posed dataset in Chapter 4 against the activity cliffs identified using ligand similarity metrics and 98th percentile similarity threshold found that the cross-docking results are reinforced by the conclusions drawn from the activity cliff definition. The cross-docking score-potency comparison shows almost no correlation within the α C-out-A subset, while maintaining moderate correlations within the other subsets. This behaviour is expected from activity cliff pairs, but it is notable that Vina and GNINA capture this SAR discontinuity with their scoring functions. Another noteworthy observation is that Vina and GNINA are able to rank non-cliff ligand pairs about 10% more accurately than they are able to rank activity cliff pairs, and their capability to rank activity cliff pairs is not statistically significant beyond random chance.

To examine the basis of the activity cliffs found within this set and gain insight about the high frequency of activity cliff pairs within the subset, structural analysis of the CDK2 protein-ligand complex dataset was performed using the KLIFS interaction fingerprints and molecular modelling program PyMOL.^{11,14} Upon first glance, KLIFS IFPs, grouped by active-site RMSD-established conformational grouping, appeared to have more conserved hydrophobic contacts than the other two conformational subsets. However, a pairwise analysis of three activity cliff pairs yielded the conclusion that within the KLIFS IFP pairs, the weaker ligand was shown to be

making more protein-ligand interactions than the stronger – a finding that is rather counterintuitive. Upon further examination of these pairs using PyMOL and defining hydrogen bonding interactions to be proximity between potential donors and acceptors on the protein and ligand below a threshold of 3.2Å, it was observed that several of the hydrogen bonds defined by the KLIFS IFP were beyond this threshold. It was also found that a key hydrogen bond within one of the ligand-receptor complexes which contributes significantly to an activity cliff was misattributed to GLN131 rather than ASN132, meaning these IFPs cannot be used as a guide for scaffold design with confidence, either. This proves that a granular analysis of KLIFS IFPs with the intention of identifying activity cliff-causing interaction differences like the study performed by Méndez-Lucio is subject to significant noise due to problems with the IFP bit-wise representation.⁶ It is important to note that most of the proposed activity cliff-causing properties of these ligands were only determined upon intensive structural analysis, rather than a scan of the protein-ligand interaction map.

The inconsistencies and imperfections arising in the KLIFS IFPs originate from several different sources. First, the KLIFS definition of the kinase active site is a broad categorization of 85 residues which make key interactions in the active site with a ligand, but this definition is kinase-specific, not CDK2-specific.¹¹ Thus small deviations in active site sequences in different branches of the kinase protein family can lead to misassignment of interactions such as the one observed within ASN132. Another major source of error is within the definitions of the interaction types themselves – they are fairly loose, with the hydrogen bond threshold being defined as $\leq 3.5\text{\AA}$ and hydrophobic contact threshold being defined as $\leq 4.5\text{\AA}$.¹⁸ This likely led to some of the issues observed with assigning a hydrogen bond in LYS89 within 3QTZ and not in 3RMF or 3R9D – whatever the cause, this discrepancy would lead to an incorrect assignment of

an additional hydrogen bond in one member of the pair. A notable observation originating from the structural analysis of several activity cliff pairs was that the identified root cause for the increase in potency between the weaker and stronger ligand was not typically derived using basic interaction features like the gain or loss of a specific bond. Rather, these cliffs were governed by specific non-intuitive electronic interactions and packing of the active site, with one pair's potency increase also resulting from the gain of a hydrogen bond. The other key observation was that CDK2 activity cliffs are not linear sets of pharmacophoric swaps which result in jumps in potency – the CDK2 ligands investigated in this study had multiple mechanisms of activity cliff formation, often with two or more contributing to the observed magnitudes of difference in biological activity. The cliff pairs form a web where each member of a pair is a node and participates as a top or bottom cliff member in multiple different cliffs. It is important to note that these observations about cliff mapping are not necessarily applicable to the entire dataset, but they provide interesting starting points for further investigation.

The goal of this chapter was to propose a method to standardize activity cliff threshold definition by similarity metrics, and explore the underlying mechanisms which cause activity cliffs within the CDK2 ligand dataset, using conformational analysis, interaction fingerprints, and structural analysis. While it needs to be further investigated whether there is a conformational basis for activity cliff formation, the protein-ligand interaction and structural basis for several α C-out-A ligands investigated. The mechanism for several CDK2 activity cliffs was proposed, and the results of this inquiry suggest that CDK2 activity cliff formation is difficult to define except on a case-by-case basis. A future direction of work in this field might be the application of CNN scoring functions to this problem, and try to treat the problem with pattern-oriented machine learning networks to identify activity cliff-forming patterns.

Sources

- (1) Maggiora, G. M. On Outliers and Activity Cliffs - Why QSAR Often Disappoints. *J. Chem. Inf. Model.* **2006**, *46* (4), 1535. <https://doi.org/10.1021/ci060117s>.
- (2) Szostek, T.; Szulczyk, D. From Obstacle to Design Advantage: Activity Cliff Aware Modeling for Small-Molecule Drug Discovery. *Drug Discov. Today* **2026**, *31* (1), 1–14. <https://doi.org/10.1016/j.drudis.2025.104589>.
- (3) López-Pérez, K.; Miranda-Quintana, R. A. Extended Activity Cliffs-Driven Approaches on Data Splitting for the Study of Bioactivity Machine Learning Predictions. *Molecular Informatics* **2025**, *44* (1), 1–17. <https://doi.org/10.1002/minf.202400054>.
- (4) Dablander, M.; Hanser, T.; Lambiotte, R.; Morris, G. M. Exploring QSAR Models for Activity-Cliff Prediction. *J. Cheminform.* **2023**, *15* (1). <https://doi.org/10.1186/s13321-023-00708-w>.
- (5) Furtmann, N.; Hu, Y.; Bajorath, J. Comprehensive Analysis of Three-Dimensional Activity Cliffs Formed by Kinase Inhibitors with Different Binding Modes and Cliff Mapping of Structural Analogues. *J. Med. Chem.* **2015**, *58* (1), 252–264. <https://doi.org/10.1021/jm5009264>.
- (6) Méndez-Lucio, O.; Kooistra, A. J.; Graaf, C. De; Bender, A.; Medina-Franco, J. L. Analyzing Multitarget Activity Landscapes Using Protein-Ligand Interaction Fingerprints: Interaction Cliffs. *J. Chem. Inf. Model.* **2015**, *55* (2), 251–262. <https://doi.org/10.1021/ci500721x>.
- (7) Łukasik, P.; Baranowska-bosiacka, I.; Kulczycka, K.; Gutowska, I. Inhibitors of Cyclin-dependent Kinases: Types and Their Mechanism of Action. *Int. J. Mol. Sci.* **2021**, *22* (6), 1–24. <https://doi.org/10.3390/ijms22062806>.
- (8) Bártová, I.; Koča, J.; Otyepka, M. Functional Flexibility of Human Cyclin-dependent Kinase-2 and Its Evolutionary Conservation. *Protein Science* **2008**, *17* (1), 22–33. <https://doi.org/10.1110/ps.072951208>.
- (9) Guha, R.; Van Drie, J. H. Structure - Activity Landscape Index: Identifying and Quantifying Activity Cliffs. *J. Chem. Inf. Model.* **2008**, *48* (3), 646–658. <https://doi.org/10.1021/ci7004093>.
- (10) Schonbrunn, E.; Betzi, S.; Alam, R.; Martin, M. P.; Becker, A.; Han, H.; Francis, R.; Chakrasali, R.; Jakkaraj, S.; Kazi, A.; Sebt, S. M.; Cubitt, C. L.; Gebhard, A. W.; Hazlehurst, L. A.; Tash, J. S.; Georg, G. I. Development of Highly Potent and Selective Diaminothiazole Inhibitors of Cyclin-Dependent Kinases. *J. Med. Chem.* **2013**, *56* (10), 3768–3782. <https://doi.org/10.1021/jm301234k>.
- (11) Van Linden, O. P. J.; Kooistra, A. J.; Leurs, R.; De Esch, I. J. P.; De Graaf, C. KLIFS: A Knowledge-Based Structural Database to Navigate Kinase-Ligand Interaction Space. *J. Med. Chem.* **2014**, *57* (2), 249–277. <https://doi.org/10.1021/jm400378w>.
- (12) Kanev, G. K.; de Graaf, C.; Westerman, B. A.; de Esch, I. J. P.; Kooistra, A. J. KLIFS: An Overhaul after the First 5 Years of Supporting Kinase Research. *Nucleic Acids Res.* **2021**, *49* (D1), D562–D569. <https://doi.org/10.1093/nar/gkaa895>.

- (13) Rodríguez-Pérez, R.; Miljković, F.; Bajorath, J. Assessing the Information Content of Structural and Protein-Ligand Interaction Representations for the Classification of Kinase Inhibitor Binding Modes via Machine Learning and Active Learning. *J. Cheminform.* **2020**, *12* (1). <https://doi.org/10.1186/s13321-020-00434-7>.
- (14) Schrödinger, L. PyMOL - The PyMOL Molecular Graphics System.
- (15) Xing, L.; Keefer, C.; Brown, M. F. Fluorine Multipolar Interaction: Toward Elucidating Its Energetics in Binding Recognition. *J. Fluor. Chem.* **2017**, *198*, 47–53. <https://doi.org/10.1016/j.jfluchem.2016.12.013>.
- (16) Pollock, J.; Borkin, D.; Lund, G.; Purohit, T.; Dyguda-Kazimierowicz, E.; Grembecka, J.; Cierpicki, T. Rational Design of Orthogonal Multipolar Interactions with Fluorine in Protein-Ligand Complexes. *J. Med. Chem.* **2015**, *58* (18), 7465–7474. <https://doi.org/10.1021/acs.jmedchem.5b00975>.
- (17) Bravo-Moraga, F.; Bedoya, M.; Zinovjev, K.; Tuñón, I.; Alzate-Morales, J. Computational Estimation of Residence Time on Roniciclib and Its Derivatives against CDK2: Extending the Use of Classical and Enhanced Molecular Dynamics Simulations. *ACS Omega* **2025**, *10* (16), 16731–16747. <https://doi.org/10.1021/acsomega.5c00555>.
- (18) Marcou, G.; Rognan, D. Optimizing Fragment and Scaffold Docking by Use of Molecular Interaction Fingerprints. *J. Chem. Inf. Model.* **2007**, *47* (1), 195–207. <https://doi.org/10.1021/ci600342e>.

Chapter 6 – Conclusions and Future Work

6.1 Conclusions

This work sought to address gaps in several areas of the cheminformatics SAR prediction field. The capability of ligand quantitative similarity metrics to predict biological activity for use as ML training sets, and these were found to be sub-optimal in predictive power. The findings in this work suggest that the descriptors used do not encode the true determinants of binding affinity in this system. The role of protein flexibility, active site conformation, and ligand selectivity based on active site conformation was explored in this study, as many QSAR studies within the CDK2 system fail to account for these key ligand-receptor binding complex properties. The findings suggest that protein active site conformation may play a key role in the SAR discontinuity observed within this dataset, and that protein flexibility and active site ligand selectivity should be accounted for within drug studies. A study of optimizing protocols for early stage VS via molecular docking within drug studies was carried out, and it was found that both receptor ensemble selection and ligand grouping can improve the accuracy of molecular docking scoring functions. Finally, a study of the SAR discontinuity within the CDK2 set was conducted and potential ligand-receptor structural causes for activity cliffs were identified, providing several insights into the discontinuity observed in this set which can be leveraged for future activity cliff prediction strategies.

In Chapter 2, a novel method of conformationally clustering CDK2 ligand-receptor complexes was proposed and implemented to allow the application of ligand-based similarity metrics to study SAR landscapes of CDK2 ligands. Computational pipelines were developed which can be leveraged to compile and prepare representative datasets of ligand-receptor crystal poses, calculate ligand-based similarity metrics, as well as perform molecular docking

simulations.^{1,2} A workflow was also developed to sample known CDK2 ligands from the ChEMBL database which have accompanying activity data but no crystal pose coordinates. This chapter established a protocol for RCSB crystal structure active site conformational clustering: grouping via key active site residue RMSD using GMM-BIC clustering method allows the definition of conformational groups which can be used to create representative ensembles for ligand similarity analyses and molecular docking analyses.^{3,4} The chapter also saw the creation of several protein preparation pipelines which seek to retain the native pose 3D coordinates of a crystal-posed ligand while maintaining structural connectivity, an issue which poses a surprising methodological challenge within the molecular docking field.^{5,6}

In Chapter 3, four different aspects of ligand similarity were evaluated to establish structure-activity relationships within the CDK2 active site. It was found that even within high similarity, low potency difference regions, there was no strong correlation between structure and activity within the representative CDK2 ligands. In fact, comparisons between the curated RCSB crystal-posed ligand set and three sets of random non-crystallized, conformationally searched CDK2 ligands from the ChEMBL database showed no substantial difference between the two SAR distributions across all four ligand similarity metrics.^{7,8} Molecular docking was explored using Autodock Vina and GNINA scoring algorithms upon the RCSB crystal set, with both methods showing that the conformational search used in ligand docking drastically reduces the ability of molecular docking scoring functions to recover the correlation between docking score and biological activity, and similarly affects the ability of docking program to correctly rank ligands in concordance with their potencies.^{9,10} It was also shown that clustering the ligands via conformational RMSD as in Chapter 2 made negligible improvements to SAR correlations within the similarity metrics. In this chapter, the foundational assumption of SAR (that molecular

similarity predicts biological activity) across a large CDK2 dataset was tested, and it was observed that common ligand similarity metrics failed to provide strong correlations with biological activity, even when binding poses are known.

In Chapter 4, the ability of CDK2 ligand cross-docking to simulate VS conditions and thus inform on the predictive performance of molecular docking was evaluated.^{9,10} The capability of ligand mean score when cross-docked into an ensemble of receptors to recover potency values was shown to out-perform the potency-docking score correlation observed within the same CDK2 dataset when self-docked. Cross-docking ligands also showed an improvement in ranking capabilities in concordance with potency values over the self-docked ranking equivalent. Grouping the ligand-receptor complexes via active site conformational group (as in Chapter 2) did not improve cross-docking results; rather, this showed that improved SAR correlations may be a ligand-based property, with specific groups of ligands responsible for notably decreasing the overall SAR trend between docking score and potency (α C-out-A subset). It was, however, observed that cross-docking into a ligand's native receptor conformation increased the pairwise ranking accuracy for pairs of ligands within both scoring functions tested (Vina and GNINA). These results suggest that ensemble docking is a useful tool to improve docking prediction within VS screens. As well, grouping ligand-receptor complexes may help to improve SAR correlations by separating well-correlated groups from those which do not display SAR correlation at all. This grouping also improves ranking, with ligands ranked more accurately within their native receptor. The clustering of ligands and subsequent docking into a curated representative ensemble using the clustering method presented in Chapter 2 would allow for the best SAR prediction within a test set, improving the outcomes of VS screens within drug discovery pipelines.

In Chapter 5, the presence of activity cliffs (ligand pairs which have high similarity and large potency differences) within the dataset was explored.¹¹ A method of activity threshold definition was developed and used to characterize activity cliff pairs represented within the results of the similarity analysis performed on the CDK2 crystal-posed ligand set. These pairs were found to be disproportionately populated with α C-out-A pairs, indicating this active site conformation and ligand scaffold type may be overly susceptible to activity cliff formation in comparison to the rest of the dataset. Several activity cliff-forming pairs were structurally analyzed and their binding modes assessed, and among these, several potential different methods of cliff formation were observed – this indicates that no singular mechanism or bond is responsible for cliff formation within this group.¹² This chapter identified some useful observations about activity cliffs within the CDK2 dataset: first, cliff pairs revealed within the ligand similarity metrics using a universal threshold metric were found to also confound molecular docking scoring functions; second, conformational clustering via active site may reveal activity cliff-prone clusters within the dataset; third, even among similar ligand scaffolds within a subset of the full CDK2 dataset, cliff-forming mechanisms were variable, indicating these phenomena are highly complex, and prediction will be elusive without involved structural pose analysis.

6.2 Future Work

Perhaps the most immediate and accessible improvement which can be made within this work is the expansion of the CDK2 dataset. Currently, the two smaller subsets, α C-out-A (n=18) and α C-in (n=9), are both too small for conclusions to be drawn which are statistically robust. Whether the addition of more ligands or clusters resolves or further contributes to the overrepresentation of α C-out-A conformations within the activity cliff pair

population would be helpful insight for the CDK2 SAR landscape. Similarly, it would make sense to explore the presence of activity cliffs across the whole kinase family. If the ligands within this study have cross-kinase activity, as many of these ligands do,¹³ it would be curious to determine whether these ligands form activity cliffs within different proteins in the kinase family, and, if they do form these cliffs within other kinases, whether the mechanism for cliff formation is the same as observed in CDK2. Another direction for this project is to apply the workflows and procedures developed to other proteins of interest with large crystal pose databases, such as HIV protease or BACE1, to explore whether the results observed for CDK2 apply to other systems. Particularly in the case of Chapter 4, where VS screening protocols are simulated and explored through cross-docking, it would be intriguing to determine whether the conclusions about docking scoring functions and optimizing accuracy in VS screens obtained from analyzing the CDK2 system are reinforced when the same analysis is applied to a different protein system. This would allow more global conclusions to be drawn from these analyses, rather than local ones about a singular protein receptor.

An addition which can be made to further explore the results of Chapter 4 is to compare the RMSD between the maximum ligand score which is cross-docked into the ensemble of receptors, and the native ligand pose. Given that it was shown that GNINA penalized poses in direct correlation with their RMSD deviation from the native ligand pose, it would be interesting to determine the effects of pose upon the cross-docked ligand, and whether these are similarly penalized. Following this, it would be useful (though computationally expensive) to perform molecular docking using GNINA's conformational search algorithm, which is guided by the GNINA scoring function and thus may perform better than Vina when seeking to recover native ligand pose within the receptor. The RMSD of the GNINA-guided docking should be compared

with the native ligand pose, to determine the effects that pose recovery have on the potency-docking score correlation within the set.

Another addition to the VS optimization in Chapter 4 which would be highly relevant is the determination of the best representative ensemble for CDK2 ligand screening via molecular docking. Based on the results in Chapter 4, calculating the mean ligand docking score after cross-docking into the receptor ensemble recaptures some of the score-potency correlation which was lost in self-docking, likely because of the ligand pose and receptor flexibility sampling which occurs as a consequence of performing many docking events. This is supported by the literature, where studies have further shown that choosing the optimal receptor ensemble via ML may improve docking score accuracy.¹⁴ However, non-random selection of a representative ensemble is a difficult task. Receptor selection could be explored by clustering representative receptors by active site RMSD (as in Chapter 2), and ranking these by their structural distinction between groups, from most diverse to least. Then, the ensembles will be built by sampling from each group's ranked list and gathering n receptors from each group, then plotting docking as a saturation curve, where n can be determined by identifying the point where adding more receptors to the ensemble provides diminishing returns in the pIC50-docking score correlation. If it is possible to use fewer receptors to achieve the same docking score-potency correlation than the full receptor set, this can save computational resources in a VS screen.

Given the activity cliff exploration performed in Chapter 5, a molecular dynamics (MD) study upon several ligand pairs within the high-similarity region of the ligand similarity metric graphs would be insightful to characterize. This would be an exploration of both high similarity pairs with high potency difference (activity cliff pairs), and high similarity pairs with low potency difference as a baseline comparison. In this study, all computational methods primarily

measure enthalpy of ligand binding, with the exception of Autodock Vina, which has a rough entropic penalty built into its scoring function.⁵ As such, there is no consideration of entropic effects upon binding such as protein flexibility, ligand flexibility, desolvation entropy, or water network reorganization effects. Research in this area has shown that MD methods can predict activity cliff pair energy differentials with high accuracy.¹⁵ MD simulations allow the capture of the time-dependent fluctuations of ligand-receptor complexes that a crystal structure cannot represent, allowing the examination of key interactions present in the potent cliff partner but not the weaker partner of the pair.¹⁶ Explicit solvent modelling would allow the exploration of solvation and desolvation effects and the role of key active-site water molecules under biologically representative conditions.¹⁶ Where this does not fully account for the observed ΔpIC_{50} , conformational entropy could additionally be estimated using the interaction entropy method.¹⁷ Applying this pipeline to the high similarity region within the CDK2 ligand-receptor dataset would allow for a detailed analysis of several entropy-based factors which may be key in describing the underlying causal relationship between cliff pairs.

Sources

- (1) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. The Protein Data Bank. *Nucleic Acids Res.* **2000**, *28* (1), 235–242.
- (2) Maggiora, G.; Vogt, M.; Stumpfe, D.; Bajorath, J. Molecular Similarity in Medicinal Chemistry. *J. Med. Chem.* **2014**, *57* (8), 3186–3204. <https://doi.org/10.1021/jm401411z>.
- (3) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Cournapeau, D.; Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Thirion, B.; Grisel, O.; Dubourg, V.; Passos, A.; VanderPlas, J.; Brucher, M.; Perrot, M.; Duchesnay, É. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.
- (4) Schwarz, G. Estimating the Dimension of a Model. *The Annals of Statistics* **1978**, *6* (2), 461–464.
- (5) Eberhardt, J.; Santos-Martins, D.; Tillack, A. F.; Forli, S. AutoDock Vina 1.2.0: New Docking Methods, Expanded Force Field, and Python Bindings. *J. Chem. Inf. Model.* **2021**, *61* (8), 3891–3898. <https://doi.org/10.1021/acs.jcim.1c00203>.
- (6) Santos-Martins, D.; He, Y.; Eberhardt, J.; Sharma, P.; Bruciaferri, N.; Holcomb, M.; Llanos, M. A.; Hansel-Harris, A.; Barkdull, A. P.; Tillack, A. F.; Bianco, G.; Paulsen, M. L.; Mato, J.; Taneja, I.; Forli, S. Meeko: Molecule Parametrization and Software Interoperability for Docking and Beyond. *J. Chem. Inf. Model.* **2025**, *65* (24), 13045–13050. <https://doi.org/10.1021/acs.jcim.5c02271>.
- (7) OpenEye, C. M. S. S. F. N. *ROCS 3.5.1.1*. <http://www.eyesopen.com>.
- (8) OpenEye Scientific Software, S. F. N. *EON 2.4.0.1*. <http://www.eyesopen.com>.
- (9) Trott, O.; Olson, A. J. AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading. *J. Comput. Chem.* **2010**, *31* (2), 455–461. <https://doi.org/10.1002/jcc.21334>.
- (10) McNutt, A. T.; Francoeur, P.; Aggarwal, R.; Masuda, T.; Meli, R.; Ragoza, M.; Sunseri, J.; Koes, D. R. GNINA 1.0: Molecular Docking with Deep Learning. *J. Cheminform.* **2021**, *13* (1). <https://doi.org/10.1186/s13321-021-00522-2>.
- (11) Stumpfe, D.; Hu, H.; Bajorath, J. Evolving Concept of Activity Cliffs. *ACS Omega* **2019**, *4* (11), 14360–14368. <https://doi.org/10.1021/acsomega.9b02221>.
- (12) Schrödinger, L. PyMOL - The PyMOL Molecular Graphics System.
- (13) Fernández, A.; Maddipati, S. A Priori Inference of Cross Reactivity for Drug-Targeted Kinases. *J. Med. Chem.* **2006**, *49* (11), 3092–3100. <https://doi.org/10.1021/jm060163j>.
- (14) Ricci-Lopez, J.; Aguila, S. A.; Gilson, M. K.; Brizuela, C. A. Improving Structure-Based Virtual Screening with Ensemble Docking and Machine Learning. *J. Chem. Inf. Model.* **2021**, *61* (11), 5362–5376. <https://doi.org/10.1021/acs.jcim.1c00511>.

- (15) Pérez-Benito, L.; Casajuana-Martin, N.; Jiménez-Rosés, M.; Van Vlijmen, H.; Tresadern, G. Predicting Activity Cliffs with Free-Energy Perturbation. *J. Chem. Theory Comput.* **2019**, *15* (3), 1884–1895. <https://doi.org/10.1021/acs.jctc.8b01290>.
- (16) Karplus, M.; Kuriyan, J. Molecular Dynamics and Protein Function. *PNAS* **2005**, *102* (19), 6679–6685.
- (17) Duan, L.; Liu, X.; Zhang, J. Z. H. Interaction Entropy: A New Paradigm for Highly Efficient and Reliable Computation of Protein-Ligand Binding Free Energy. *J. Am. Chem. Soc.* **2016**, *138* (17), 5722–5728. <https://doi.org/10.1021/jacs.6b02682>.

Chapter 7 – Supplementary Information

7.1 Proteins from CDK2 RCSB Crystal-Posed Dataset

Table 1. List of proteins and ligands and accompanying IC50 values obtained from KLIFS and RCSB databases with primary sources.

PDB ID	Ligand ID	IC50 (uM)	Reference	PDB ID	Ligand ID	IC50 (uM)	Reference
1PXI	CK1	17000	Structure. 2003, 11, 399–410 ¹	1Y8Y	CT7	2000	Bioorg. Med. Chem. Lett. 2005, 15, 863–867 ²
2BTR	U73	95	Journal of Molecular Graphics and Modelling, 2006, 24, 341–348 ³	2C68	CT6	1800	Bioorg. Med. Chem. Lett. 2005, 15, 863–867 ²
2CLX	F18	14500	J. Med. Chem. 2006, 49, 22, 6500–6509 ⁴	2R3G	SC9	800	Biopolymers. 2008, 89, 372–379 ⁵
2R3G	SC9	800	Biopolymers. 2008, 89, 372–379 ⁵	2R3I	SCF	1000	Biopolymers. 2008, 89, 372–379 ⁵
2R3K	SCQ	100	Biopolymers. 2008, 89, 372–379 ⁵	2R3L	SCW	100	Biopolymers. 2008, 89, 372–379 ⁵
2R3M	SCX	10	Biopolymers. 2008, 89, 372–379 ⁵	2R3P	3SC	900	Biopolymers. 2008, 89, 372–379 ⁵
3LE6	2BZ	35	Bioorg. Med. Chem. Lett. 2010, 20, 5984–5987 ⁶	3R9D	X6B	71000	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3R9N	Z68	1400	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3RNI	21Z	6700	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3RAH	O1Z	12000	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	1H1R	6CP	2300	J. Med. Chem. 2004, 47, 15, 3710–3722 ⁸
3TNW	F18	14500	J. Med. Chem. 2006, 49, 22, 6500–6509 ⁹	2A0C	CK9	50	Eur. J. Med. Chem., 2013, 61, 61–72 ¹⁰
1OIU	N76	210	J. Med. Chem. 2004, 47, 15, 3710–3722 ⁸	2B53	D23	600	Bioorg. Med. Chem. Lett. 2001, 11, 1157–1160 ¹¹
2B52	D42	8	Bioorg. Med. Chem. Lett. 2004, 14, 343–346 ¹²	2R3F	SC8	500	Biopolymers. 2008, 89, 372–379 ⁵
2B54	D05	20	J. Med. Chem. 2004, 47, 24, 5894–5911 ¹³	2R3O	2SC	600	Biopolymers. 2008, 89, 372–379 ⁵
2R3J	SCJ	10	Biopolymers. 2008, 89, 372–379 ⁵	3FZ1	B98	146	Bioorg. Med. Chem. Lett. 2009, 19, 4878–4881
3BHT	MFR	11	J. Med. Chem. 2008, 51, 4, 737–751 ¹⁴	3QQK	X02	15000	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3PXY	JWS	5900	ACS Chem. Biol. 2011, 6, 5, 492–501 ¹⁵	3QTR	X36	930	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3QTQ	X35	3100	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3QTU	X44	70	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3QTS	X46	3100	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3QTX	X43	70	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3QTW	X3A	650	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3QU0	X40	10.5	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3QTZ	X42	50	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3R8U	Z31	5000	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3QXP	X64	10.8	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3R8Z	Z63	49000	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3R8V	Z62	2900	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3RJC	06Z	5700	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3RAL	04Z	100	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3RK7	08Z	200	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3RK5	07Z	4000	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3RKB	12Z	380	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3RK9	09Z	520	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3RPV	26Z	70	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3RMF	20Z	82000	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3S00	Z60	12000	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷
3RPY	27Z	4800	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3SQQ	99Z	4800	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷

PDB ID	Ligand ID	IC50 (uM)	Reference	PDB ID	Ligand ID	IC50 (uM)	Reference
3S1H	56Z	17000	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3TIY	TIY	17000	ChemBioChem, 2012, 13, 14, 2128–2136 ¹⁶
3TI1	B49	130000	ChemBioChem, 2012, 13, 14, 2128–2136 ¹⁶	4CFV	75X	23500	J. Med. Chem. 2014, 57, 1, 56–70 ¹⁷
3UNJ	0BX	11000	ACS Chem. Biol. 2012, 7, 4, 698–706 ¹⁷	4EZ3	0S0	45000	ChemBioChem, 2012, 13, 14, 2128–2136 ¹⁶
4EK6	10K	3	J. Med. Chem. 2004, 47, 13, 3367–3380 ¹⁸	4LYN	1YG	60	J. Med. Chem. 2002, 45, 18, 3905–3927 ¹⁹
4GCJ	X64	10.8	J. Med. Chem. 2013, 56, 10, 3768–3782 ⁷	3IG7	EFP	9	Bioorg. Med. Chem. Lett. 2009, 19, 5703–5707 ¹⁹
1PXO	CK7	10	Chemistry & Biology, 2006, 13, 201–211 ²⁰	4FKL	CK2	6500	Chemistry & Biology, 2006, 13, 201–211 ²⁰
3IGG	EFQ	95	Bioorg. Med. Chem. Lett. 2009, 19, 5703–5707 ¹⁹	1KE7	LS3	8.9	J. Med. Chem. 2001, 44, 25, 4339–4358 ²¹
6GUK	FC8	17	Bioorganic & Medicinal Chemistry, 2017 25, 6523–6535 ²²	1KE9	LS5	660	J. Med. Chem. 2001, 44, 25, 4339–4358 ²¹
1KE6	LS2	5.7	J. Med. Chem. 2001, 44, 25, 4339–4358 ²¹	3ULI	1N3	70	Bioorg. Med. Chem. Lett. 2012, 22, 2063–2069 ²³
1KE8	LS4	1000	J. Med. Chem. 2001, 44, 25, 4339–4358 ²¹	2C6K	DT2	730	Bioorg. Med. Chem. Lett. 2006, 16, 1353–1357 ²⁴
3WBL	PDY	23000	J. Med. Chem. 2012, 55, 15, 6700–6715 ²⁵	2VTL	LZ5	97000	J. Med. Chem. 2008, 51, 16, 4986–4999 ²⁶
2C6M	DT5	350	Bioorg. Med. Chem. Lett. 2006, 16, 1353–1357 ²⁴	2VTH	LZ2	120000	J. Med. Chem. 2008, 51, 16, 4986–4999 ²⁶
2C6I	DT1	11000	Bioorg. Med. Chem. Lett. 2006, 16, 1353–1357 ²⁴	4BGH	3I6	4	ChemMedChem 2013, 8, 1067–1085 ²⁷
2VTI	LZ3	660	J. Med. Chem. 2008, 51, 16, 4986–4999 ²⁶	1E1V	CMG	17000	Bioorg. Med. Chem. Lett. 2003, 13, 3079–3082 ²⁸
2VTR	LZB	1500	J. Med. Chem. 2008, 51, 16, 4986–4999 ²⁶	3DDQ	RRC	543.3	J. Med. Chem. 2019, 62, 4, 1989–1998 ²⁹
2VTS	LZC	30	J. Med. Chem. 2008, 51, 16, 4986–4999 ²⁶	6GUC	SU9	110	Bioorg. Med. Chem. Lett. 2003, 13, 1939–1942 ³⁰
3NS9	NS9	2.4	ChemMedChem 2017, 12, 372–380 ³¹	4EK4	1CK	6000	J. Med. Chem. 2004, 47, 3367–3380 ¹⁸
3PY0	SU9	110	Bioorg. Med. Chem. Lett. 2003, 13, 1939–1942 ³⁰	4ERW	STU	3.4	J. Med. Chem. 2009, 52, 7, 1853–1863 ³²
1H1S	4SP	69.7	Bioorg. Med. Chem. Lett. 2006, 16, 1353–1357 ²⁴	5JQ5	I74	578	ChemMedChem 2017, 12, 372–380 ³¹
4EK5	03K	68	J. Med. Chem. 2004, 47, 3367–3380 ¹⁸	2VTA	LZ1	185000	J. Med. Chem. 2008, 51, 16, 4986–4999 ²⁶
2R3R	6SC	24.5	Patent US8580782B2 ³³	6GUE	FB8	6	Bioorg. Med. Chem. Lett. 2008, 18, 6369–6373 ³⁴

7.2 ChEMBL Ligand Datasets with Accompanying pKi values

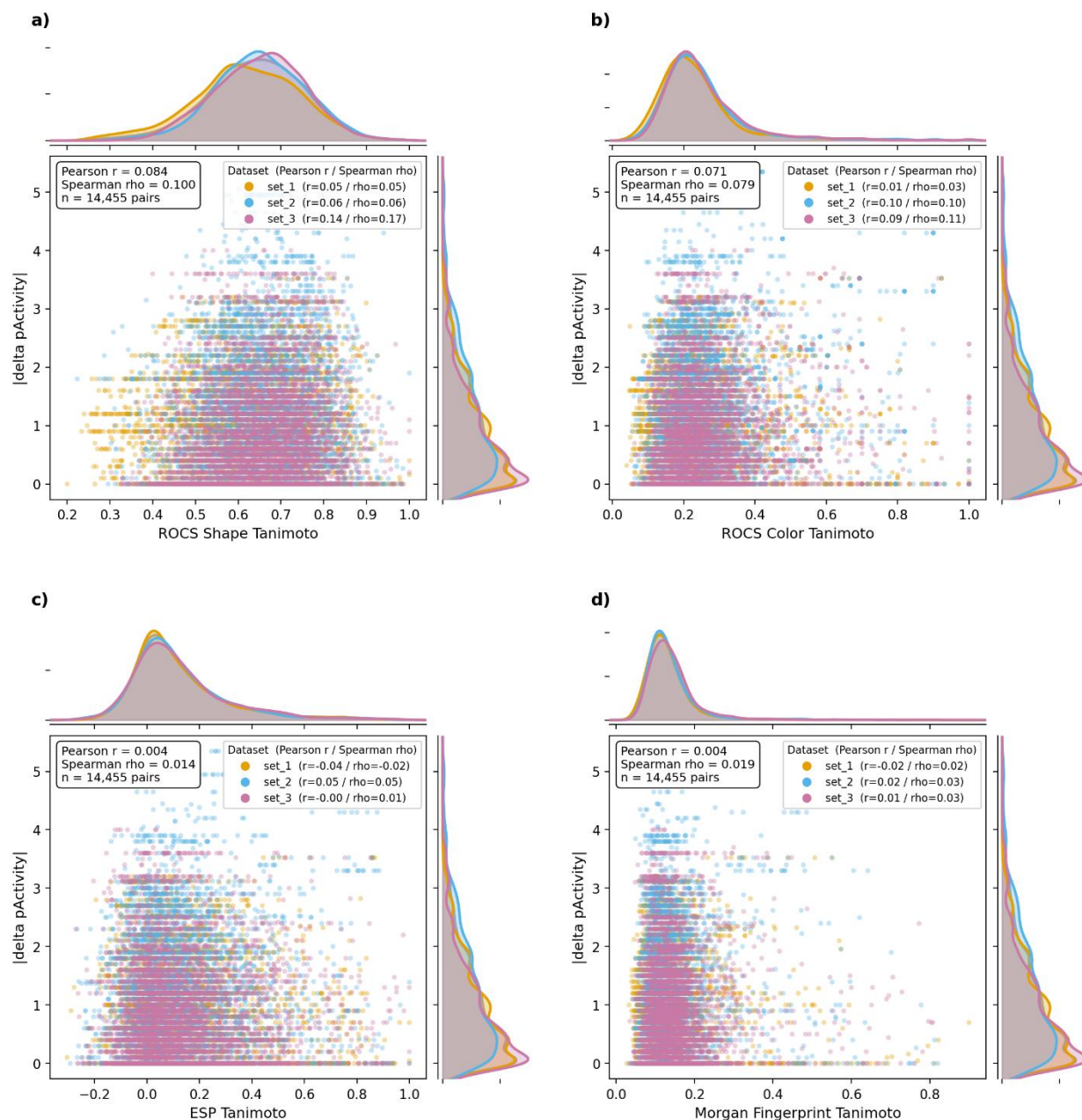


Figure 1. Scatter plots of pair-wise ligand Tanimoto similarity vs $|\Delta pK_i|$ for 3 datasets of ChEMBL CDK2 ligands ($n = 95$, creating $n = 4,465$ unique comparisons per set). Ligand poses were generated from SMILES strings using RDKit and conformationally sampled using Openeye OMEGA for top ligand conformations, then overlaid using ROCS to find a maximum structural overlap. Marginal kernel density estimates (KDE) display the density distribution of similarity values (top) and activity values (right). (a) Openeye ROCS Shape Tanimoto vs $|\Delta pK_i|$ for all ligands (b) Openeye ROCS Color Tanimoto vs $|\Delta pK_i|$ for all ligands (c) Openeye EON Electrostatic Potential

Tanimoto vs $|\Delta pK_i|$ for all ligands. Note that negative ESP Tanimoto similarity values correspond to an in-space overlap of opposing charges. (d) Morgan Fingerprint ECFP4 Tanimoto vs $|\Delta pK_i|$ for all ligands.

7.3 Cross-Docked CDK2 Database Pairwise into the Same Receptor

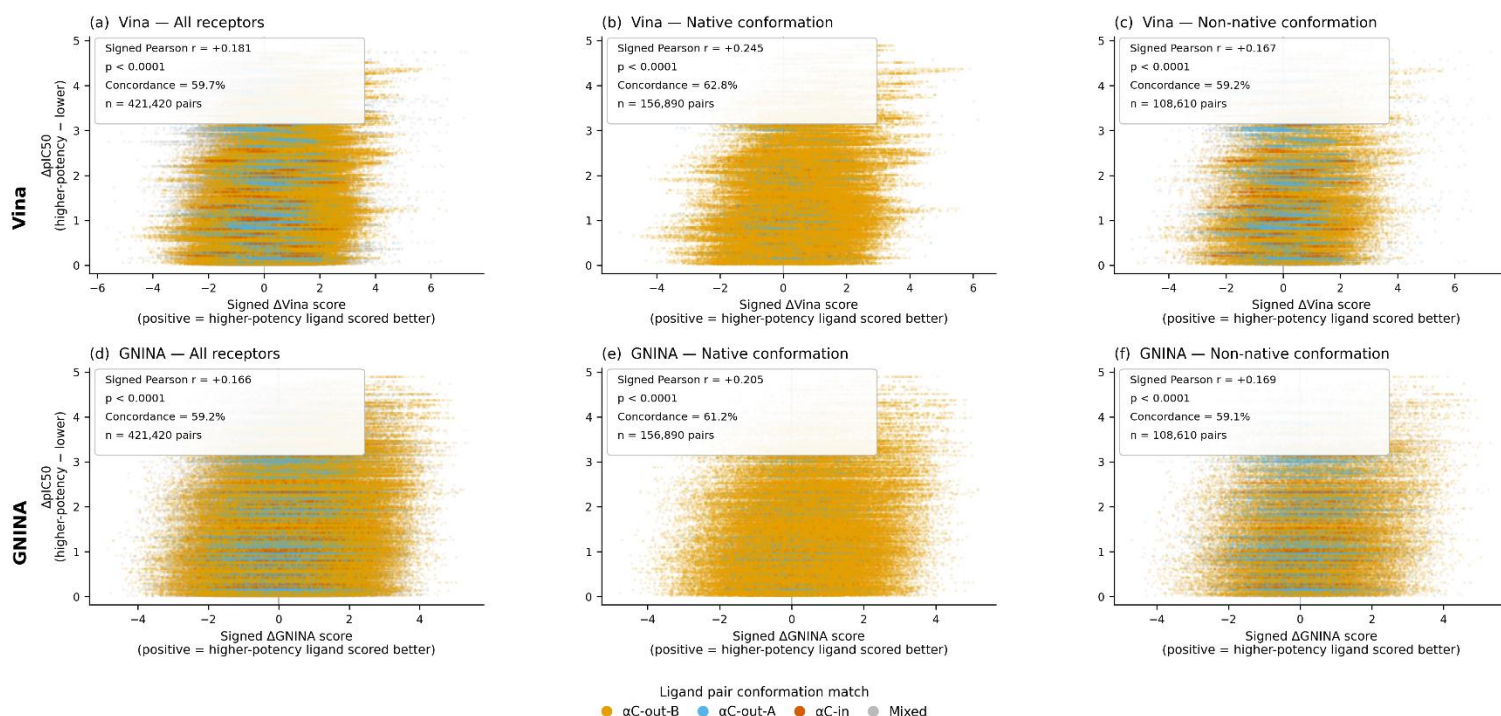


Figure 2. Each data point corresponds to a pair of ligands docked into the same receptor. Signed Pearson r calculated using pairwise comparison of two cross-docked ligand scores within the same receptor and both their pIC₅₀ values. ‘Signed’ means the lower pIC₅₀ value is subtracted from the higher, with the docking score difference calculated in correspondence (Vina scores are multiplied by -1 to standardize). Pearson r therefore measures the pairwise ranking accuracy and the magnitude accuracy within each scoring functions docking scoring. “Native receptors” means the dataset was limited to pairs where both ligands were from receptor poses of the same active site conformation. “Non-native conformation” means either one or both ligands in the pair were from a different active-site conformation. “All receptors” means all cross-docked score pairs were calculated within all 95 receptor conformations. All pairs with the same pIC₅₀ score or docking score were dropped since they do not contribute to the analysis. “Pairwise ranking accuracy” reflects the % of pairs which correctly capture this directionality (n pairs within category/n total pairs x 100%) – 50% is considered “expected by chance”.

7.4 RCSB CDK2 Database Protein-Ligand Complex Preparation, and ROCS and Vina/GNINA Workflow

7.4.1 Fetch Ligands, Align, Prepare and Convert Ligands for Docking

```
#!/usr/bin/env python3
"""
process_pdb_mustang.py

CDK2 docking preparation pipeline with MUSTANG structural alignment before conversion from pdb to
pdbqt
The goal of this script is to prepare pdb files for docking (convert to pdbqt) with preserving
the
connectivity and coordinates of the ligand as much as possible, which is a major issue with
docking
Because this script is based on the process highlighted in a long Jupyter notebook using
different methods
and uses many different modules and computational packages, there is troubleshooting throughout
to determine
which modules aren't working.

The workflow is based on the Jupyter notebook found here:
https://github.com/ljmartin/pdb_to_pdbqt

it is improved for large scale kinase database use by adding multi-structure handling and
automation with
a RCSB and KLIFS fetch, by aligning the receptors and ligands using MUSTANG, using Dimorphite for
protonation, and using Meeko for the conversion. Molecular dynamics stuff is removed from the
original.
Post-conversion, the script calculates a docking box based on the aligned ligand coordinates.

SCRIPT OVERVIEW
For a list of PDB files the script will:
- Fetch the raw PDB from RCSB
- Identify the drug-like ligand (chain A, largest HETATM residue)
- Extract chain A protein and write a clean PDB to a MUSTANG input
- Run MUSTANG pairwise against the chosen reference structure to get a
  rigid-body rotation matrix and translation vector
- Apply the transformation matrix to BOTH protein and ligand so that all
  structures land in the common reference frame
- Select the chain containing the most KLIFS pocket residues and preserve the sequence
  (no residues removed), the full native chain is used for docking so all
  binding-site contacts are preserved
- Clean the ligand with RDKit (removes non-drug HETATM atoms (metals, ions, solvents, extra
  ligands, assigns correct bond orders from RCSB SMILES, protonates at physiological pH 7.4
  (via Dimorphite-DL if available, otherwise RDKit, removes explicit Hs on heavy atoms
- Protonates the trimmed pocket protein with PDBFixer at pH 7.4, then relaxes only the
  H positions with OpenMM
- Convert the protonated ligand to PDBQT via Meeko (+ MGLTools fallback).
- Converts the protonated pocket protein to PDBQT via AutoDock4's
  "prepare_receptor4.py" (with OpenBabel fallback)
- After all structures are processed, compute a global docking box
  that encloses all aligned ligand poses with padding
- Compute pairwise and all-vs-reference Cα RMSD on the 85 KLIFS active-site
  residues across every aligned crystal pose and writes a CSV

MUSTANG is called as an external subprocess within this script. Install it at:
  http://lcb.infotech.monash.edu.au/mustang/
and place the binary on your PATH (or set MUSTANG_BIN below).

The pipeline extracts the MUSTANG output transformation (translation + rotation)
from the .pml / .html output, applies it with numpy, and writes the transformed
coordinates - so no intermediate superimposed PDB from MUSTANG is required
```

KLIFS ACTIVE-SITE RESIDUES (CDK2 numbering)

KLIFS defines 85 pocket residues across 8 structural regions. For CDK2 (UniProt P24941) the residue numbers that map to KLIFS positions are defined in KLIFS RESIDUES CDK2 below. These are used to identify which chain in the asymmetric unit contains the CDK2 active site – the chain with the most overlapping residues is selected and kept WHOLE. No residues are cut out.

Reference: van Linden et al. (2014) J. Med. Chem. 57(2):249-277.

DEPENDENCIES TO RUN THIS SCRIPT:

```
conda install -c conda-forge prody pdbfixer openmm numpy rdkit scipy -y
pip install meeko requests dimorphite-dl          # dimorphite optional but recommended given
that its the optimal for protonating
MUSTANG binary on PATH (or MUSTANG_BIN below)
MGLTools 1.5.7 (for receptor PDBQT conversion)
```

USAGE INSTRUCTIONS EXAMPLE

```
python process_pdb_mustang.py                # all IDs in pdb_ids.txt
python process_pdb_mustang.py 3ddq 4cfv 1oiu  # specific entries only
python process_pdb_mustang.py --rmsd-only     # rerun RMSD/box without re-preparing
```

"""

```
import sys
import os
import re
import json
import shutil
import subprocess
import tempfile
import warnings
import requests
from pathlib import Path
from io import StringIO
```

```
import numpy as np
import prody
from prody import writePDB, parsePDB
import pdbfixer
from openmm import app, unit, openmm
from rdkit import Chem
from rdkit.Chem import AllChem
import meeko
from scipy.spatial.distance import squareform
from scipy.cluster.hierarchy import linkage
```

USER CONFIG - edit these paths if you need to run it differently than I did

```
PROJECT_DIR = Path(r"C:\Users\Documents\Protein_Project\docking_redo\mustang_global")
PDB_IDS_FILE = PROJECT_DIR / "pdb_ids.txt"

OUTPUT_DIR = PROJECT_DIR / "output"
DATA_DIR = PROJECT_DIR / "data"
ALIGNED_DIR = OUTPUT_DIR / "aligned_pdb"          # aligned protein + ligand PDBs after MUSTANG
run
POCKET_DIR = OUTPUT_DIR / "pocket_pdb"          # trimmed pocket proteins after cleanup
PDBQT_DIR = OUTPUT_DIR / "pdbqt"
RMSD_DIR = OUTPUT_DIR / "rmsd"                  # rmsd calculations from 1elv
WORK_DIR = PROJECT_DIR / "work"
```

MUSTANG binary – invoked via WSL (Linux binary, built from source)

If you move the binary, update MUSTANG_WSL_PATH below

MUSTANG_BIN = "wsl"

MUSTANG_WSL_PATH = "/mnt/c/Users/rrempp/Downloads/MUSTANG_v3.2.4/bin/mustang-3.2.4"

```

# Reference structure for MUSTANG alignment (all others aligned to this one) - so pick a good
one!
REFERENCE_PDB = "1e1v"

# MGLTools - used for receptor PDBQT conversion if Meeko doesn't work
MGLTOOLS_DIR = Path(r"C:\Program Files (x86)\MGLTools-1.5.7")
MGLTOOLS_PY = MGLTOOLS_DIR / "python.exe"
PREP_RECEPTOR = (MGLTOOLS_DIR / "Lib" / "site-packages" /
                    "AutoDockTools" / "Utilities24" / "prepare_receptor4.py")

# Global docking box padding (Å added on each face beyond outermost ligand atom coordinates)
GLOBAL_BOX_PADDING = 8.0

# KLIFS ACTIVE-SITE RESIDUES (CDK2 / P24941 numbering)
#
# The 85 KLIFS residues define the kinase binding pocket. For CDK2 the
# mapping from KLIFS position → UniProt/PDB residue number was taken from
# the KLIFS database (https://klifs.net, structure 1e1v).
#
# Reference: van Linden et al. (2014) J. Med. Chem. 57(2):249-277.
#
# These residue numbers are used for trimming the protein to the pocket region and computing Cα
RMSD across aligned structures
#
# These residue numbers are used to identify which chain to keep (the chain
# that contains the most KLIFS residues). The entire chain is kept intact, with no residues
removed
# the residues are numbered and labelled by what the residues correspond to in the protein
KLIFS_RESIDUES_CDK2 = {
    # β1 (KLIFS 1-3)
    11, 12, 13,
    # β2 (KLIFS 4-7)
    14, 15, 16, 17,
    # β3 / P-loop (KLIFS 8-19)
    18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
    # αC helix (KLIFS 20-30)
    43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
    # β4 (KLIFS 31-34)
    55, 56, 57, 58,
    # β5 (KLIFS 35-38)
    59, 60, 61, 62,
    # GK / hinge (KLIFS 39-49) - includes hinge H-bond donors/acceptors
    63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
    # αD / linker (KLIFS 50-52)
    74, 75, 76,
    # αE (KLIFS 53-59)
    77, 78, 79, 80, 81, 82, 83,
    # β6 (KLIFS 60-64)
    116, 117, 118, 119, 120,
    # β7 (KLIFS 65-67)
    121, 122, 123,
    # β8 (KLIFS 68-69)
    124, 125,
    # DFG loop (KLIFS 70-76)
    145, 146, 147, 148, 149, 150, 151,
    # αF (KLIFS 77-85)
    152, 153, 154, 155, 156, 157, 158, 159, 160,
}

# Residues always excluded from ligand selection - this is necessary so they don't get
accidentally identified as the ligand itself
JUNK_RESIDUES = {
    "HOH", "WAT", "H2O",
    "NA", "K", "CL", "CA", "MG", "ZN", "MN", "FE", "CU", "CO",
    "NI", "CD", "HG", "PT", "AU", "AG",
    "SO4", "PO4", "GOL", "PEG", "MPD", "TRS", "MES", "HEP",
    "EDO", "PGE", "EOH", "DMS", "IMD", "ACT", "FMT", "BME",
    "DTT", "TCE", "MOH", "IPA",
    "ACE", "NME",
    "SGM", "NO3", "MGF", "DTU", "CSD",

```

```

# SCW removed - it is a drug-like inhibitor in some CDK2 structures (e.g. 2r3l)
"ALA", "ARG", "ASN", "ASP", "CYS", "GLN", "GLU", "GLY",
"HIS", "ILE", "LEU", "LYS", "MET", "PHE", "PRO", "SER",
"THR", "TRP", "TYR", "VAL",
}

# DIRECTORY SETUP

for _d in [DATA_DIR, OUTPUT_DIR, ALIGNED_DIR, POCKET_DIR, PDBQT_DIR, RMSD_DIR, WORK_DIR]:
    _d.mkdir(parents=True, exist_ok=True)

prody.confProDy(verbosity="none")

# DEPENDENCY CHECK
# necessary since there's a lot of dependencies and especially MUSTANG is a little hard to figure
out

def check_dependencies():
    # MUSTANG
    result = subprocess.run([MUSTANG_BIN, MUSTANG_WSL_PATH],
                           capture_output=True, text=True)
    if "mustang" not in (result.stdout + result.stderr).lower():
        raise FileNotFoundError(
            f"MUSTANG not found at WSL path '{MUSTANG_WSL_PATH}'.\n"
            f"Check the path is correct and the binary was compiled:\n"
            f"  cd /mnt/c/Users/rremp/Downloads/MUSTANG_v3.2.4 && make"
        )
    print(f" MUSTANG found (via WSL): {MUSTANG_WSL_PATH}")

    # MGLTools
    if not MGLTOOLS_PY.exists():
        raise FileNotFoundError(
            f"MGLTools python not found at: {MGLTOOLS_PY}\n"
            f"Update MGLTOOLS_DIR in this script."
        )
    print(f" MGLTools found: {MGLTOOLS_DIR}")
    print(f" Meeko version: {meeko.__version__}")

    # Dimorphite-DL (used for physiological pH protonation, if it doesn't work will fall back to
    RDKit)
    try:
        import dimorphite_dl
        v = getattr(dimorphite_dl, "__version__", "unknown")
        print(f" Dimorphite-DL found v{v} (will use for physiological pH protonation)")
    except ImportError:
        print(" Dimorphite-DL not found - falling back to RDKit MolStandardize")
        print("  Install: pip install dimorphite-dl")

# STEP 1 - Fetch and parse PDB

def fetch_and_parse_pdb(pdb_id):
    """Fetch from RCSB and return (full AtomGroup, protein chain A, all heteroatoms)."""
    print(f"\n Fetching {pdb_id.upper()} from RCSB...")
    fname = prody.fetchPDB(pdb_id, folder=str(DATA_DIR))
    if isinstance(fname, list):
        fname = fname[0]

    pdb = parsePDB(str(fname))

    # Prefer chain A; fall back to any protein
    protein = pdb.select("protein and chain A")
    if protein is None:
        print("  No chain A - selecting all protein atoms")
        protein = pdb.select("protein")
    if protein is None:
        raise ValueError(f"{pdb_id}: No protein atoms found")

```

```

heteroatoms = pdb.select("not protein and not resname HOH WAT H2O")
print(f"    Protein atoms: {protein.numAtoms()}")
if heteroatoms:
    print(f"    HETATM residues: {set(heteroatoms.getResnames())}")

return pdb, protein, heteroatoms

# STEP 2 – Identify drug-like ligand

def select_drug_ligand(pdb_id, pdb_obj, heteroatoms):
    """Pick the largest non-junk HETATM residue in chain A."""
    resnames = set(heteroatoms.getResnames())
    drug_resnames = resnames - JUNK_RESIDUES
    if not drug_resnames:
        print("    All HETATM matched junk list – falling back to largest residue")
        drug_resnames = resnames

    best_resname, best_count = None, 0
    for r in drug_resnames:
        sel = pdb_obj.select(f"resname {r}")
        if sel and sel.numAtoms() > best_count:
            best_count = sel.numAtoms()
            best_resname = r

    if best_resname is None:
        raise ValueError(f"{pdb_id}: Could not identify drug-like ligand")

    drug_ligand = pdb_obj.select(
        f"resname {best_resname} and chain A and (altloc _ A)"
    )
    if drug_ligand is None or drug_ligand.numAtoms() == 0:
        drug_ligand = pdb_obj.select(f"resname {best_resname} and chain A")
    if drug_ligand is None or drug_ligand.numAtoms() == 0:
        print(f"    No chain A copy of {best_resname} – using all chains")
        drug_ligand = pdb_obj.select(
            f"resname {best_resname} and (altloc _ A)"
        )
    if drug_ligand is None or drug_ligand.numAtoms() == 0:
        drug_ligand = pdb_obj.select(f"resname {best_resname}")

    print(f"    Drug ligand: {best_resname} ({drug_ligand.numAtoms()} atoms)")
    if drug_ligand.numAtoms() < 5:
        print(f"    Only {drug_ligand.numAtoms()} ligand atoms – check structure")
    _protein_resnames = {
        "ALA", "ARG", "ASN", "ASP", "CYS", "GLN", "GLU", "GLY", "HIS", "ILE",
        "LEU", "LYS", "MET", "PHE", "PRO", "SER", "THR", "TRP", "TYR", "VAL"
    }
    if best_resname in _protein_resnames:
        print(f"    WARNING: selected ligand {best_resname} is a protein residue!") #this
sometimes happens
        print(f"    Check if the actual drug ligand is in JUNK_RESIDUES.")

    return drug_ligand, best_resname

# STEP 3 – Write clean chain A protein for MUSTANG

# Standard amino acid residue names MUSTANG accepts
_STANDARD_AA = {
    "ALA", "ARG", "ASN", "ASP", "CYS", "GLN", "GLU", "GLY",
    "HIS", "ILE", "LEU", "LYS", "MET", "PHE", "PRO", "SER",
    "THR", "TRP", "TYR", "VAL",
    "HID", "HIE", "HIP", "HSD", "HSE", "HSP", # protonation variants
    "MSE", # selenomethionine
}

def write_mustang_input(pdb_id, protein):
    """
    Write a clean standard-residue ATOM-only PDB for MUSTANG.

```

```

Parses the ProDy AtomGroup atom-by-atom and writes PDB records
directly (same column-position format as the original pipeline's
clean_protein_and_ligand). This gives full control: only the 20
standard amino acids (plus common variants) are written, so
modified residues like CSD, ACE, NME that crash MUSTANG are
silently excluded. If this isn't run then MUSTANG will likely
crash on every protein.
"""
out = WORK_DIR / f"{pdb_id}_chainA_for_mustang.pdb"
kept = 0

with open(out, "w", newline="") as f:
    serial_out = 0
    for atom in protein.iterAtoms():
        resname = atom.getResname().strip()
        if resname not in _STANDARD_AA:
            continue

        # Build a fixed-width PDB ATOM record
        # Columns follow the PDB format spec exactly so MUSTANG
        # can parse them without issues.
        serial_out += 1
        name = atom.getName() or ""
        altloc = atom.getAltloc() or " "
        chain = atom.getChid() or "A"
        resnum = atom.getResnum() or 0
        icode = atom.getIcode() or " "
        x, y, z = atom.getCoords()
        occ = atom.getOccupancy() if atom.getOccupancy() is not None else 1.0
        bfac = atom.getBeta() if atom.getBeta() is not None else 0.0
        element = atom.getElement() or ""

        # Atom name formatting: 4 chars, left-padded for 1-char elements
        if len(name) < 4 and len(element) == 1:
            name_field = f"{name:<3s}"
        else:
            name_field = f"{name:<4s}"

        line = (
            f"ATOM "
            f"{serial_out:5d} "
            f"{name_field}"
            f"{altloc:1s}"
            f"{resname:3s} "
            f"{chain:1s}"
            f"{resnum:4d}"
            f"{icode:1s} "
            f"{x:8.3f}{y:8.3f}{z:8.3f}"
            f"{occ:6.2f}"
            f"{bfac:6.2f} "
            f"{element:>2s}\n"
        )
        f.write(line)
        kept += 1

    f.write("END\n")

print(f"    MUSTANG input: {kept} atoms, standard residues only ({pdb_id})")
return out

# STEP 4: Run MUSTANG and parse the transformation matrix

def run_mustang_and_get_transform(mobile_pdb: Path, reference_pdb: Path) -> tuple:
    """
    Align mobile to reference (in this case 1elv) using MUSTANG and extract the rigid-body
    transformation: rotation matrix R (3x3) and translation vector t (3,).

    MUSTANG v3.2.4 does NOT write the transformation matrix to any text file.
    Instead it writes superimposed PDB files to the output prefix.

```

The output files are named:
 <prefix><N>.pdb where N is the index of each input structure (1-based)
 So for two inputs:
 mustang_out1.pdb - superimposed mobile
 mustang_out2.pdb - superimposed reference (should match ref_pdb coords)

We compute R and t by solving the least-squares superposition between the original mobile C α coordinates and the superimposed mobile C α coordinates using ProDy's calcTransformation, then extract R and t from the result. If you are using a different version of MUSTANG it might output the transformation matrix, but for my version it was easier to back-calculate the transformation matrix and apply it to each ligand after MUSTANG alignment of the proteins in space. If you don't need to generate this, then comment out the transformation matrix section and update your output to output the matrix and aligned protein.

Returns:

```
R : np.ndarray (3, 3)
t : np.ndarray (3,)
rmsd : float MUSTANG-reported RMSD
n_aligned : int number of aligned residues
"""
wsl_tmpdir = subprocess.run(
    ["wsl", "mktemp", "-d", "/tmp/mustang_XXXXXX"],
    capture_output=True, text=True
).stdout.strip()
if not wsl_tmpdir:
    raise RuntimeError("Could not create WSL temp directory via mktemp")

try:
    def to_wsl_path(p: Path) -> str:
        """Convert C:\foo\bar -> /mnt/c/foo/bar"""
        p = Path(p).resolve()
        drive = p.drive.rstrip(":").lower()
        rest = p.as_posix().split(":", 1)[1]
        return f"/mnt/{drive}{rest}"

    mob_wsl = wsl_tmpdir + "/" + mobile_pdb.name
    ref_wsl = wsl_tmpdir + "/" + reference_pdb.name
    subprocess.run(["wsl", "cp", to_wsl_path(mobile_pdb), mob_wsl], check=True)
    subprocess.run(["wsl", "cp", to_wsl_path(reference_pdb), ref_wsl], check=True)

    out_prefix = wsl_tmpdir + "/mustang_out"

    cmd = [
        MUSTANG_BIN,          # "wsl"
        MUSTANG_WSL_PATH,
        "-i", mob_wsl, ref_wsl,
        "-o", out_prefix,
        # No -F flag - default output is superimposed PDB files
        # MUSTANG writes RMSD to <prefix>.rms_rot, not stdout
    ]

    result = subprocess.run(cmd, capture_output=True, text=True)
    stdout = result.stdout + result.stderr

    # Read superimposed PDB (mustang_out.pdb)
    # MUSTANG v3.2.4 writes superimposed coordinates to mustang_out.pdb.
    # It does NOT write a .rms_rot file unless -r ON is passed, but
    # -r ON is incompatible with the default output mode.

    sup_pdb_wsl = out_prefix + ".pdb"
    sup_content = subprocess.run(
        ["wsl", "cat", sup_pdb_wsl],
        capture_output=True, text=True
    ).stdout

    if not sup_content.strip():
        ls_diag = subprocess.run(
            ["wsl", "ls", "-la", wsl_tmpdir], capture_output=True, text=True
        ).stdout
        raise RuntimeError(
```

```

        f"MUSTANG produced no output PDB.\n"
        f"tmpdir:\n{ls_diag}\nstdout:\n{stdout}"
    )

    # Parse RMSD from HTML output
    rmsd = float("nan")
    n_aligned = 0
    html_wsl = out_prefix + ".html"
    html_text = subprocess.run(
        ["wsl", "cat", html_wsl], capture_output=True, text=True
    ).stdout
    for line in html_text.splitlines():
        m = re.search(r"\d+\\|s+---\s+([\d.]+)", line)
        if not m:
            m = re.search(r"\d+\\|s+([\d.]+)\s+---", line)
        if m:
            try:
                rmsd = float(m.group(1))
            except ValueError:
                pass
            break

    # Write superimposed PDB to Windows path for ProDy/numpy
    mob_sup_win = WORK_DIR / f"_mustang_sup_{mobile_pdb.stem}.pdb"
    mob_sup_win.write_text(sup_content, encoding="utf-8", errors="replace")

finally:
    subprocess.run(["wsl", "rm", "-rf", wsl_tmpdir], capture_output=True)

# Compute R, t via Kabsch on original vs superimposed Cα
try:
    mob_orig = parsePDB(str(mobile_pdb))
    mob_sup = parsePDB(str(mob_sup_win))

    if mob_orig is None or mob_sup is None:
        raise ValueError("Could not parse mobile or superimposed PDB")

    ca_orig = mob_orig.select("name CA")
    ca_sup = mob_sup.select("name CA")

    if ca_orig is None or ca_sup is None:
        raise ValueError("No Cα atoms found")

    orig_resmap = {atom.getResnum(): atom.getCoords()
                    for atom in ca_orig.iterAtoms()}

    # mustang_out.pdb contains BOTH structures concatenated with the
    # same residue numbers. Build sup_resmap from only the FIRST
    # structure (mobile) by stopping at the first chain break.
    # We do this by reading the raw PDB text and extracting CA coords
    # only up to the first TER/ENDMDL record.
    sup_resmap = {}
    sup_pdb_text = mob_sup_win.read_text(encoding="utf-8", errors="replace")
    for line in sup_pdb_text.splitlines():
        if line.startswith("TER") or line.startswith("ENDMDL") or line.startswith("END"):
            break
        if line.startswith("ATOM") and line[12:16].strip() == "CA":
            try:
                resnum = int(line[22:26])
                x = float(line[30:38])
                y = float(line[38:46])
                z = float(line[46:54])
                sup_resmap[resnum] = np.array([x, y, z])
            except (ValueError, IndexError):
                continue

    shared = sorted(set(orig_resmap) & set(sup_resmap))
    if len(shared) < 10:
        raise ValueError(f"Too few shared Cα: {len(shared)}")

    orig_arr = np.array([orig_resmap[r] for r in shared])

```

```

        sup_arr = np.array([sup_resmap[r] for r in shared])
        n_aligned = len(shared)

        R, t = calc_transform(orig_arr, sup_arr)
        rmsd_str = f"{rmsd:.3f}" if not np.isnan(rmsd) else "n/a"
        print(f"    Superposition: {n_aligned} shared Cα, RMSD={rmsd_str} Å")

    finally:
        if mob_sup_win.exists():
            mob_sup_win.unlink()

    return R, t, rmsd, n_aligned

    return R, t, rmsd, n_aligned

def calc_transform(orig: np.ndarray, sup: np.ndarray) -> tuple:
    """
    Kabsch algorithm: find R, t such that R @ orig + t ≈ sup.
    This is the calculation of the transformation matrix
    If you don't need it, get rid of it.

    Parameters:
    orig : (N, 3) original Cα coordinates
    sup  : (N, 3) superimposed Cα coordinates (from MUSTANG output)

    Returns:
    R : (3, 3) rotation matrix
    t : (3,) translation vector
    """
    orig_com = orig.mean(axis=0)
    sup_com = sup.mean(axis=0)
    A = orig - orig_com
    B = sup - sup_com

    H = A.T @ B
    U, S, Vt = np.linalg.svd(H)

    # Ensure proper rotation (det = +1, not a reflection)
    d = np.linalg.det(Vt.T @ U.T)
    D = np.diag([1.0, 1.0, d])

    R = Vt.T @ D @ U.T
    t = sup_com - R @ orig_com
    return R, t

def apply_transform(coords: np.ndarray, R: np.ndarray, t: np.ndarray) -> np.ndarray:
    """Apply rigid-body transform: x_new = R @ x + t (works on (N,3) array)."""
    return (R @ coords.T).T + t

# STEP 5 - Select the chain containing the KLIFS active site

def select_klifs_chain(protein, pdb_id) -> prody.AtomGroup:
    """
    Select the single chain that contains the most KLIFS active-site residues.
    The ENTIRE chain is kept intact - no residues are removed - so the native
    structure is fully preserved for docking. This just drops other chains
    (e.g. a second kinase copy in the ASU, or a fused domain on a different
    chain letter) that don't contain the CDK2 pocket.
    """
    best_chain = None
    best_overlap = 0

    for chain_id in set(protein.getChids()):
        chain = protein.select(f"chain {chain_id}")
        resids = set(chain.getResnums().tolist())
        overlap = len(resids & KLIFS_RESIDUES_CDK2)
        if overlap > best_overlap:
            best_overlap = overlap

```

```

        best_chain = chain_id

    if best_chain is None:
        print(f"    {pdb_id}: Could not identify KLIFS chain - keeping full protein")
        return protein

    selected = protein.select(f"chain {best_chain}")
    n_res = len(set(selected.getResindices()))
    print(f"    KLIFS chain: {best_chain} "
          f"({n_res} residues, {selected.numAtoms()} atoms, "
          f"{best_overlap}/{len(KLIFS_RESIDUES_CDK2)} KLIFS residues matched)")
    return selected

# STEP 6 - Clean ligand with RDKit (physiological pH, remove non-drug atoms)

def fetch_smiles(resname: str) -> str | None:
    """Fetch canonical SMILES from RCSB for a CCD residue code."""
    url = f"https://data.rcsb.org/rest/v1/core/chemcomp/{resname}"
    try:
        r = requests.get(url, timeout=10)
        r.raise_for_status()
        data = r.json()
        for entry in data.get("pdbx_chem_comp_descriptor", []):
            if entry.get("type") == "SMILES_CANONICAL" and "OEToolkits" in entry.get("program",
"""):
                return entry["descriptor"]
        for entry in data.get("pdbx_chem_comp_descriptor", []):
            if "SMILES" in entry.get("type", ""):
                return entry["descriptor"]
        return data.get("chem_comp", {}).get("smiles")
    except Exception as e:
        print(f"    RCSB SMILES fetch failed: {e}")
        return None

def neutralize_smiles(smiles: str) -> str:
    """Neutralize common charged groups (RCSB returns zwitterionic forms). Necessary to prevent
    duplicates due to charges"""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return smiles
    mol = Chem.RWMol(mol)
    changed = False
    for atom in mol.GetAtoms():
        fc = atom.GetFormalCharge()
        sym = atom.GetSymbol()
        if fc > 0 and sym in ("N", "O", "S", "P"): #possible charged atoms
            nH = atom.GetTotalNumHs()
            if nH > 0:
                atom.SetFormalCharge(fc - 1)
                atom.SetNumExplicitHs(max(0, nH - 1))
                changed = True
            elif fc < 0 and sym in ("O", "S", "N"):
                atom.SetFormalCharge(fc + 1)
                atom.SetNumExplicitHs(atom.GetTotalNumHs() + 1)
                changed = True
    if changed:
        try:
            Chem.SanitizeMol(mol)
            return Chem.MolToSmiles(mol)
        except Exception:
            pass
    return smiles

def _protonate_rdkit_molfixer(smiles: str, ph: float) -> str | None:
    """
    Protonate using RDKit MolStandardize at a target pH.
    This is the fallback option if Dimorpate doesn't work to protonate
    MolStandardize's Uncharger is pH-unaware, so we implement a simple

```

```

rule-based protonation that covers the most common drug-like ionisable
groups:
- Amines (pKa ~9-10) to protonated [NH3+] / [NH2+] / [NH+]
- Carboxylic acids (pKa ~4-5) to deprotonated [COO-]
- Sulfonamides (pKa ~10) to neutral NH
- Phenols (pKa ~10) to neutral OH
- Imidazoles / pyridines (pKa ~5-7) to neutral at pH 7.4
- Phosphates (pKa2 ~6.8) to monodeprotonated [PO4H]-

This is a decent approximation for pH 7.4.
For edge cases (e.g. pKa very close to 7.4) Dimorphite-DL is more
accurate.
"""
from rdkit.Chem.MolStandardize import rdMolStandardize

mol = Chem.MolFromSmiles(smiles)
if mol is None:
    return None

# Standardize first (canonical tautomer, etc.)
try:
    te = rdMolStandardize.TautomerEnumerator()
    mol = te.Canonicalize(mol)
except Exception:
    pass

# Apply a neutral/physiological charge model:
# Uncharger removes all charges → good starting point for pH 7.4
# (most drug-like groups are neutral; only strong acids/bases differ)
try:
    uncharger = rdMolStandardize.Uncharger(canonicalOrder=True)
    mol = uncharger.uncharge(mol)
except Exception:
    pass

try:
    Chem.SanitizeMol(mol)
    return Chem.MolToSmiles(mol)
except Exception:
    return None

def protonate_smiles_at_ph(smiles: str, ph: float = 7.4) -> tuple[str, str]:
    """
    Return (smiles_at_ph, method_used).

    Protonation priority:
    1. Dimorphite-DL – most accurate, pKa-based
    2. RDKit MolStandardize Uncharger – rule-based, good for most CDK inhibitors
    3. Original RCSB SMILES – last resort, usually near-neutral anyway
    """
    # Option 1: Dimorphite-DL
    try:
        # dimorphite-dl >= 1.3 uses protonate_smiles
        from dimorphite_dl import protonate_smiles as _dim_protonate
        # dimorphite-dl v2+ uses ph_min/ph_max (v1 used min_ph/max_ph)
        protonated = _dim_protonate(smiles, ph_min=ph - 0.1, ph_max=ph + 0.1,
                                    precision=0.5)

        if protonated:
            result = protonated[0] if isinstance(protonated, list) else protonated
            return result, "dimorphite-dl"
    except ImportError:
        try:
            # older API fallback
            from dimorphite_dl import run_with_mol_list
            protonated = run_with_mol_list(
                [smiles], min_ph=ph - 0.1, max_ph=ph + 0.1,
                pka_precision=0.5, label_states=False, silent=True,
            )
            if protonated:
                return protonated[0], "dimorphite-dl"

```

```

        except ImportError:
            pass # not installed - try next tier
    except Exception as e:
        print(f"        Dimorphite-DL failed ({e}) - trying RDKit MolStandardize")

    # Option 2: RDKit MolStandardize
    try:
        result = _protonate_rdkit_molfixer(smiles, ph)
        if result is not None:
            print(f"        pH protonation: RDKit MolStandardize (Dimorphite-DL unavailable)")
            return result, "rdkit-molstandardize-fallback"
    except Exception as e:
        print(f"        RDKit MolStandardize protonation failed ({e}) - using RCSB SMILES")

    # Option 3: original RCSB SMILES
    print(f"        pH protonation: using original RCSB SMILES (all methods failed)")
    return smiles, "rcsb-smiles-fallback"

def prepare_ligand_rdkit(pdb_id: str, drug_resname: str,
                        aligned_lig_pdb: Path, out_pdb: Path,
                        log: dict, out_sdf: Path = None) -> bool:
    """
    Given the aligned ligand PDB (heavy atoms from crystal, MUSTANG frame):
    1. Load with RDKit (no sanitization yet).
    2. Assign bond orders from RCSB SMILES protonated at pH 7.4.
    3. Add Hs with 3-D coordinates.
    4. Sanitize.
    5. Assign Gasteiger charges.
    6. Write clean PDB (no extra junk atoms).
    7. ALSO write an SDF (out_sdf), since PDB cannot encode bond orders -
       downstream tools that need correct bond orders/aromaticity (ROCS,
       EON, Morgan fingerprinting) should consume the SDF, not the PDB.

    Populates log['lig_ph_method'] and log['lig_bond_orders'].
    Returns True on success.
    """
    # 1. Fetch + process SMILES
    smiles = fetch_smiles(drug_resname)
    if smiles is None:
        print(f"        No SMILES for {drug_resname} - will attempt RDKit parse without template")
        log["lig_ph_method"] = "none-no-smiles"
        log["lig_bond_orders"] = "none-no-smiles"
    else:
        smiles = neutralize_smiles(smiles)
        smiles, ph_method = protonate_smiles_at_ph(smiles, ph=7.4)
        log["lig_ph_method"] = ph_method
        print(f"        SMILES (pH 7.4, {ph_method}): {smiles[:70]}{'...' if len(smiles)>70 else ''}")

    # 2. Load PDB
    mol_pdb = Chem.MolFromPDBFile(str(aligned_lig_pdb), removeHs=True, sanitize=False)
    if mol_pdb is None:
        print(f"        RDKit cannot parse {aligned_lig_pdb.name}")
        log["lig_bond_orders"] = "failed-rdkit-parse"
        return False, None

    # Keep largest fragment (should already be clean, belt-and-suspenders)
    frags = Chem.GetMolFrags(mol_pdb, asMols=True, sanitizeFrags=False)
    mol_pdb = max(frags, key=lambda m: m.GetNumAtoms())

    # 3. Assign bond orders from SMILES template
    if smiles:
        mol_ref = Chem.MolFromSmiles(smiles)
        if mol_ref is None:
            mol_ref = Chem.MolFromSmiles(neutralize_smiles(smiles))
        if mol_ref is not None:
            with warnings.catch_warnings():
                warnings.simplefilter("ignore")
                assigned = False
            # Try progressively simplified SMILES variants

```

```

        smiles_variants = [smiles]
        # Also try with neutralized nitro groups (Dimorphite charges them)
        neutral = smiles.replace("[N+] (=O) [O-]", "N(=O)=O")
        if neutral != smiles:
            smiles_variants.append(neutral)
        # Also try original uncharged RCSB SMILES
        orig_smiles = fetch_smiles(drug_resname)
        if orig_smiles and orig_smiles not in smiles_variants:
            smiles_variants.append(orig_smiles)
        for smi_try in smiles_variants:
            ref_try = Chem.MolFromSmiles(smi_try)
            if ref_try is None:
                continue
            try:
                mol_pdb = AllChem.AssignBondOrdersFromTemplate(ref_try, mol_pdb)
                log["lig_bond_orders"] = "rcsb-smiles-template"
                assigned = True
                break
            except Exception:
                continue
        if not assigned:
            print(f"        Bond order assignment failed (all SMILES variants tried) - using
raw PDB connectivity")
            log["lig_bond_orders"] = "raw-pdb-connectivity-fallback"
        else:
            log["lig_bond_orders"] = "raw-pdb-connectivity-fallback"
    else:
        log.setdefault("lig_bond_orders", "raw-pdb-connectivity-fallback")

# 4. Add Hs with 3-D coordinates (important for Meeko / Gasteiger)
mol_pdb = Chem.AddHs(mol_pdb, addCoords=True)

# 5. Sanitize
try:
    Chem.SanitizeMol(mol_pdb)
except Exception as e:
    print(f"        Sanitization failed ({e})")
    log["lig_bond_orders"] += "+sanitize-failed"
    return False, None

# 6. Gasteiger charges
AllChem.ComputeGasteigerCharges(mol_pdb)
# Ensure charges are plain floats, not lists (can happen with some RDKit versions)
for atom in mol_pdb.GetAtoms():
    chg = atom.GetPropsAsDict().get("_GasteigerCharge", 0.0)
    if isinstance(chg, (list, tuple)):
        chg = float(chg[0]) if chg else 0.0
    atom.SetDoubleProp("_GasteigerCharge", float(chg))

# 7. Write clean PDB (for backward compatibility / visual inspection)
Chem.MolToPDBFile(mol_pdb, str(out_pdb))
print(f"        Ligand clean PDB → {out_pdb.name}")

# 8. ALSO write clean SDF - this is the file that actually preserves the
#     bond orders / aromaticity just assigned above. PDB format has no
#     bond-order field, so anything reading the PDB back in later (ROCS,
#     EON, Morgan fingerprinting) would have to re-derive bonds from
#     geometry alone and could get aromaticity wrong for fused-ring
#     systems. Downstream similarity-pipeline steps should read THIS
#     file, not out_pdb.
if out_sdf is not None:
    try:
        writer = Chem.SDWriter(str(out_sdf))
        mol_for_sdf = Chem.Mol(mol_pdb)
        mol_for_sdf.SetProp("Name", pdb_id.upper())
        mol_for_sdf.SetProp("pdb_id", pdb_id.upper())
        mol_for_sdf.SetProp("ligand_code", drug_resname)
        mol_for_sdf.SetProp("lig_bond_orders", log.get("lig_bond_orders", ""))
        writer.write(mol_for_sdf)
        writer.close()
        print(f"        Ligand clean SDF (bond orders preserved) → {out_sdf.name}")
    
```

```

        except Exception as e:
            print(f"Failed to write ligand SDF ({e}) - "
                  f"downstream similarity steps will need the PDB fallback")

    return True, mol_pdb

# STEP 7 - Protonate & relax pocket protein

def protonate_and_relax_protein(pdb_id: str, pocket_pdb: Path,
                                ph: float = 7.4) -> tuple[Path, str]:
    """
    Run PDBFixer at pH 7.4 then relax H positions with OpenMM.

    missingResidues is cleared after findMissingResidues() so PDBFixer only
    adds missing atoms within existing residues (e.g. disordered sidechains)
    and never tries to rebuild loop regions that are genuinely absent in the
    crystal structure.

    Returns (path_to_protonated_pdb, method_used).
    """
    fixer = pdbfixer.PDBFixer(str(pocket_pdb))
    fixer.findMissingResidues()
    fixer.missingResidues = {} # don't rebuild missing loops / termini
    fixer.findMissingAtoms()
    fixer.addMissingAtoms()
    # Add terminal capping atoms (OXT at C-terminus, H2/H3 at N-terminus)
    # Without this, truncated chains missing OXT cause OpenMM template errors
    # and confuse AutoDock4's bond assignment.
    try:
        fixer.addMissingTerminalResidues()
    except Exception:
        pass # not all PDBFixer versions have this method; safe to skip
    fixer.addMissingHydrogens(ph)

    protH_path = WORK_DIR / f"{pdb_id}_pocketH.pdb"
    with open(protH_path, "w") as f:
        app.PDBFile.writeFile(fixer.topology, fixer.positions, f, keepIds=True)
    print(f"Pocket protonated at pH {ph} -> {protH_path.name}")

    # Relax Hs only (freeze heavy atoms)
    try:
        forcefield = app.ForceField("amber14-all.xml", "amber14/tip3pfb.xml")
        system = forcefield.createSystem(
            fixer.topology,
            nonbondedMethod=app.CutoffNonPeriodic,
            nonbondedCutoff=0.9 * unit.nanometer,
        )
        elements = [atom.element.name for atom in fixer.topology.atoms()]
        for i in range(system.getNumParticles()):
            if elements[i] != "hydrogen":
                system.setParticleMass(i, 0.0)

        integrator = openmm.LangevinIntegrator(
            298 * unit.kelvin, 1 / unit.picosecond, 1 * unit.femtosecond
        )
        sim = app.Simulation(
            fixer.topology, system, integrator,
            openmm.Platform.getPlatformByName("CPU"),
        )
        sim.context.setPositions(fixer.positions)
        sim.minimizeEnergy()

        pos = sim.context.getState(getPositions=True).getPositions()
        out_path = POCKET_DIR / f"{pdb_id}_pocketH_relaxed.pdb"
        with open(out_path, "w") as f:
            app.PDBFile.writeFile(fixer.topology, pos, f, keepIds=True)
        print(f"H relaxation done -> {out_path.name}")
        return out_path, "pdbfixer+openmm-relaxed"

    except Exception as e:

```

```

        print(f"      H relaxation failed ({e}) - using unrelaxed protonated structure")
        fallback = POCKET_DIR / f"{pdb_id}_pocketH_relaxed.pdb"
        shutil.copy(proth_path, fallback)
        return fallback, "pdbfixer-protonated-only"

# STEP 8 - Ligand to PDBQT (Meeko, with MGLTools fallback)

def ligand_to_pdbqt(pdb_id: str, lig_pdb: Path, log: dict, mol_prepared=None) -> bool:
    dst = PDBQT_DIR / f"{pdb_id}_ligand.pdbqt"

    # Meeko path
    try:
        if mol_prepared is not None:
            # Use the already-prepared molecule (has correct bond orders,
            # explicit Hs, and Gasteiger charges already computed)
            mol = mol_prepared
        else:
            mol = Chem.MolFromPDBFile(str(lig_pdb), removeHs=False, sanitize=False)
            if mol is None:
                raise ValueError("RDKit could not parse ligand PDB")
            mol = Chem.AddHs(mol, addCoords=True)
            try:
                Chem.SanitizeMol(mol)
            except Exception as e:
                raise ValueError(f"Sanitization failed before Meeko: {e}")
            AllChem.ComputeGasteigerCharges(mol)
            for atom in mol.GetAtoms():
                chg = atom.GetPropsAsDict().get("_GasteigerCharge", 0.0)
                if isinstance(chg, (list, tuple)):
                    chg = float(chg[0]) if chg else 0.0
                atom.SetDoubleProp("_GasteigerCharge", float(chg))

            preparator = meeko.MoleculePreparation()
            with warnings.catch_warnings():
                warnings.simplefilter("ignore", DeprecationWarning)
                mol_setups = preparator.prepare(mol)

            if mol_setups and hasattr(meeko, "PDBQTWriterLegacy"):
                pdbqt_str, ok, err = meeko.PDBQTWriterLegacy.write_string(mol_setups[0])
                if not ok:
                    raise ValueError(f"PDBQTWriterLegacy: {err}")
            else:
                with warnings.catch_warnings():
                    warnings.simplefilter("ignore", DeprecationWarning)
                    pdbqt_str = preparator.write_pdbqt_string()

            dst.write_text(pdbqt_str)
            if dst.stat().st_size > 0:
                print(f"      Ligand PDBQT (Meeko) -> {dst.name}")
                log["lig_pdbqt_method"] = "meeko"
                return True

    except Exception as e:
        print(f"      Meeko failed ({e}) - trying MGLTools")

    # MGLTools fallback
    PREP_LIGAND = (MGLTOOLS_DIR / "Lib" / "site-packages" /
                   "AutoDockTools" / "Utilities24" / "prepare_ligand4.py")
    cmd = [str(MGLTOOLS_PY), str(PREP_LIGAND),
           "-l", str(lig_pdb), "-o", str(dst)]
    result = subprocess.run(cmd, capture_output=True, text=True)
    if dst.exists() and dst.stat().st_size > 0:
        print(f"      Ligand PDBQT (MGLTools fallback) -> {dst.name}")
        log["lig_pdbqt_method"] = "mgltools-fallback"
        return True

    print(f"      Ligand PDBQT conversion failed for {pdb_id}")
    log["lig_pdbqt_method"] = "failed"
    return False

```

```

# STEP 9 - Receptor to PDBQT via AutoDock4 / prepare_receptor4.py

def receptor_to_pdbqt(pdb_id: str, receptor_pdb: Path, log: dict) -> bool:
    """
    Convert protonated pocket PDB → PDBQT using AutoDock4's prepare_receptor4.py.
    Falls back to OpenBabel if MGLTools crashes.

    Flags:
        -A bonds : assign bond orders only (we already added Hs via PDBFixer)
        -U nphs  : merge non-polar Hs (AutoDock / Vina convention)
    """
    dst = PDBQT_DIR / f"{pdb_id}_receptor.pdbqt"

    cmd = [str(MGLTOOLS_PY), str(PREP_RECEPTOR),
            "-r", str(receptor_pdb), "-o", str(dst),
            "-A", "bonds", "-U", "nphs"]
    result = subprocess.run(cmd, capture_output=True, text=True,
                           cwd=str(POCKET_DIR))

    if dst.exists() and dst.stat().st_size > 0:
        print(f"    Receptor PDBQT (AutoDock4) to {dst.name}")
        log["rec_pdbqt_method"] = "autodock4"
        return True

    print("    AutoDock4 receptor conversion failed - trying OpenBabel")
    for line in result.stderr.splitlines():
        if line.strip() and "memory leak" not in line.lower():
            print(f"    AD4 error: {line.strip()}")
            break

    cmd_ob = ["obabel", str(receptor_pdb), "-xr", "-O", str(dst)]
    subprocess.run(cmd_ob, capture_output=True, text=True)
    if dst.exists() and dst.stat().st_size > 0:
        print(f"    Receptor PDBQT (OpenBabel fallback) to {dst.name}")
        log["rec_pdbqt_method"] = "openbabel-fallback"
        return True

    print(f"    Receptor PDBQT conversion failed for {pdb_id}")
    log["rec_pdbqt_method"] = "failed"
    return False

# STEP 10 - Global docking box from all aligned ligands

def compute_global_box(aligned_lig_coords: dict, padding: float = GLOBAL_BOX_PADDING) -> dict:
    """
    Compute a single docking box that encloses all aligned ligand poses.

    Parameters
    -----
    aligned_lig_coords : dict {pdb_id to np.ndarray (N, 3)}
    padding             : float Å added on each face

    Returns
    -----
    dict with keys 'center' (list[float]) and 'size' (list[float])
    """
    all_coords = np.vstack(list(aligned_lig_coords.values()))
    min_xyz = all_coords.min(axis=0)
    max_xyz = all_coords.max(axis=0)
    center = ((min_xyz + max_xyz) / 2.0).tolist()
    size = ((max_xyz - min_xyz) + padding).tolist()
    print(f"\n Global box center: {[f'{c:.2f}' for c in center]}")
    print(f" Global box size: {[f'{s:.2f}' for s in size]}")
    return {"center": center, "size": size}

# STEP 11 - Cα RMSD on KLIFS active site

def collect_klifs_ca_coords(aligned_proteins: dict) -> dict:

```

```

"""
From each aligned pocket PDB (after MUSTANG transform), extract the Cα
coordinates for KLIFS residues present in ALL structures (intersection).

Returns:
dict {pdb_id to np.ndarray (K, 3)} where K = |shared KLIFS residues|
shared_residues : sorted list of shared residue numbers
"""
per_struct = {}
for pdb_id, prot_path in aligned_proteins.items():
    prot = parsePDB(str(prot_path))
    ca = prot.select("name CA")
    if ca is None:
        continue
    resid_map = {}
    for atom in ca.iterAtoms():
        r = atom.getResnum()
        if r in KLIFS_RESIDUES_CDK2:
            resid_map[r] = atom.getCoords()
    per_struct[pdb_id] = resid_map

# Intersection of KLIFS residues present in every structure
common = set(KLIFS_RESIDUES_CDK2)
for resid_map in per_struct.values():
    common &= set(resid_map.keys())
shared = sorted(common)

print(f"\n Shared KLIFS Cα residues for RMSD: {len(shared)} / {len(KLIFS_RESIDUES_CDK2)}")

coords = {}
for pdb_id, resid_map in per_struct.items():
    coords[pdb_id] = np.array([resid_map[r] for r in shared])

return coords, shared

def compute_rmsd_matrix(ca_coords: dict) -> tuple:
    """
    Compute pairwise Cα RMSD matrix for all structures.

    Returns:
    pdb_ids : list of pdb_id strings (row/col labels)
    rmsd_mat : np.ndarray (N, N) symmetric RMSD matrix
    """
    pdb_ids = sorted(ca_coords.keys())
    n = len(pdb_ids)
    mat = np.zeros((n, n))

    for i in range(n):
        for j in range(i + 1, n):
            diff = ca_coords[pdb_ids[i]] - ca_coords[pdb_ids[j]]
            rmsd = np.sqrt((diff ** 2).sum(axis=1).mean())
            mat[i, j] = rmsd
            mat[j, i] = rmsd

    return pdb_ids, mat

def save_rmsd_results(pdb_ids, rmsd_mat, reference: str):
    """Save RMSD matrix as CSV and a simple text summary."""
    import csv

    # Full pairwise CSV
    csv_path = RMSD_DIR / "klifs_ca_rmsd_matrix.csv"
    with open(csv_path, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([""] + pdb_ids)
        for i, pid in enumerate(pdb_ids):
            writer.writerow([pid] + [f"{v:.3f}" for v in rmsd_mat[i]])
    print(f" RMSD matrix → {csv_path.name}")

```

```

# vs-reference column
if reference in pdb_ids:
    ref_idx = pdb_ids.index(reference)
    ref_col = rmsd_mat[:, ref_idx]
    vs_path = RMSD_DIR / f"klifs_ca_rmsd_vs_{reference}.csv"
    with open(vs_path, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["pdb_id", f"rmsd_vs_{reference}"])
        for pid, rmsd in zip(pdb_ids, ref_col):
            writer.writerow([pid, f"{rmsd:.3f}"])
    print(f"    RMSD vs reference → {vs_path.name}")

# Quick summary stats
vals = [v for pid, v in zip(pdb_ids, ref_col) if pid != reference]
if vals:
    print(f"\n    KLIFS Cα RMSD vs {reference}:")
    print(f"        mean = {np.mean(vals):.3f} Å")
    print(f"        max   = {np.max(vals):.3f} Å")
    print(f"        min   = {np.min(vals):.3f} Å")
    print(f"        >1 Å = {sum(v > 1.0 for v in vals)}/{len(vals)} structures")
    print(f"        >2 Å = {sum(v > 2.0 for v in vals)}/{len(vals)} structures")

return csv_path

# MAIN PIPELINE

def run_pipeline(pdb_list: list, rmsd_only: bool = False):
    check_dependencies()

    # Tracks coords of all aligned ligands for global box computation
    aligned_lig_coords: dict[str, np.ndarray] = {}
    # Tracks paths of aligned pocket proteins for RMSD
    aligned_pocket_paths: dict[str, Path] = {}

    # Preparation audit log – one dict per structure, written to CSV at end
    prep_logs: list[dict] = []

    done = 0
    failed = []

    # Ensure reference structure is processed first
    ref = REFERENCE_PDB.lower()
    if ref in pdb_list and pdb_list[0] != ref:
        pdb_list = [ref] + [p for p in pdb_list if p != ref]

    # Write reference protein for MUSTANG
    # Pre-load reference path if it already exists on disk from a previous run
    ref_mustang_pdb_candidate = WORK_DIR / f"{ref}_chainA_for_mustang.pdb"
    ref_mustang_pdb: Path | None = (
        ref_mustang_pdb_candidate
        if ref_mustang_pdb_candidate.exists()
        else None
    )
    if ref_mustang_pdb is not None:
        print(f"    Reference MUSTANG input found on disk: {ref_mustang_pdb.name}")

    for i, pdb_id in enumerate(pdb_list, start=1):
        pdb_id = pdb_id.lower().strip()
        print(f"\n{'='*60}")
        print(f"    {pdb_id.upper()}    ({i}/{len(pdb_list)})")
        print(f"{'='*60}")

        # Per-structure log entry – populated incrementally below
        log: dict = {
            "pdb_id":      pdb_id,
            "status":      "unknown",
            "lig_ph_method": "n/a",
            "lig_bond_orders": "n/a",
            "lig_pdbqt_method": "n/a",
            "rec_prot_method": "n/a",

```

```

        "rec_pdbqt_method": "n/a",
        "notes": ""
    }

    # Skip if already done (both PDBQT files exist AND the ligand SDF
    # exists). The SDF check matters specifically for structures that
    # were already fully processed by an earlier version of this script
    # (before the SDF write was added) - without it, those structures
    # would be skipped forever and never get a ligand_clean.sdf written.
    lig_pdbqt = PDBQT_DIR / f"{pdb_id}_ligand.pdbqt"
    rec_pdbqt = PDBQT_DIR / f"{pdb_id}_receptor.pdbqt"
    lig_clean_sdf_check = WORK_DIR / f"{pdb_id}_ligand_clean.sdf"
    aligned_pock = POCKET_DIR / f"{pdb_id}_pocketH_relaxed.pdb"
    aligned_lig = ALIGNED_DIR / f"{pdb_id}_ligand_aligned.pdb"

    if (not rmsd_only and lig_pdbqt.exists() and rec_pdbqt.exists()
        and lig_clean_sdf_check.exists()):
        print("    Already processed (incl. SDF) - skipping")
        log["status"] = "skipped-already-done"
        # Still collect coords for global box / RMSD
        if aligned_lig.exists():
            _lig = Chem.MolFromPDBFile(str(aligned_lig), removeHs=True, sanitize=False)
            if _lig:
                aligned_lig_coords[pdb_id] = _lig.GetConformer().GetPositions()
        if aligned_pock.exists():
            aligned_pocket_paths[pdb_id] = aligned_pock
        prep_logs.append(log)
        done += 1
        continue
    elif (not rmsd_only and lig_pdbqt.exists() and rec_pdbqt.exists()
          and not lig_clean_sdf_check.exists()):
        print("    PDBQTs already exist, but ligand_clean.sdf is "
              "missing (older run, pre-SDF-fix).")
        backfilled = False
        if aligned_lig.exists():
            # Fast path: the ligand is already aligned and was already
            # successfully PDBQT-converted previously, so we know its
            # chemistry worked before. Read the resname directly off the
            # existing aligned PDB (every HETATM line already carries
            # it) and regenerate just the SDF, without re-fetching the
            # structure or re-running MUSTANG.
            print("    → Backfilling SDF only (fast path, no re-fetch/re-align)")
            try:
                _aligned_obj = prody.parsePDB(str(aligned_lig))
                _resnames_bf = set(_aligned_obj.getResnames()) if _aligned_obj is not None
            except Exception:
                _resnames_bf = set()

            if len(_resnames_bf) == 1:
                _drug_resname_bf = next(iter(_resnames_bf))
                lig_clean_pdb_bf = WORK_DIR / f"{pdb_id}_ligand_clean.pdb"
                bf_log = dict(log)
                bf_ok, _ = prepare_ligand_rdkit(
                    pdb_id, _drug_resname_bf, aligned_lig,
                    lig_clean_pdb_bf, bf_log, out_sdf=lig_clean_sdf_check
                )
                if bf_ok:
                    log.update(bf_log)
                    log["status"] = "backfilled-sdf-only"
                    prep_logs.append(log)
                    done += 1
                    backfilled = True
                else:
                    print("        Fast-path SDF backfill failed; doing "
                          "full reprocessing for this structure instead.")
            else:
                print(f"        Could not uniquely identify ligand resname "
                      f"from {aligned_lig.name} ({_resnames_bf or 'none found'}); "
                      f"doing full reprocessing for this structure instead.")
        if backfilled:

```

```

        continue
    # Either aligned_lig doesn't exist, or the fast path failed -
    # fall through to full reprocessing below.

try:
    # 1. Fetch
    pdb_obj, protein, heteroatoms = fetch_and_parse_pdb(pdb_id)
    if heteroatoms is None:
        print("    No heteroatoms found - skipping")
        log["status"] = "failed"
        log["notes"] = "no heteroatoms"
        failed.append((pdb_id, "no heteroatoms"))
        prep_logs.append(log)
        continue

    # 2. Identify drug ligand
    drug_ligand, drug_resname = select_drug_ligand(pdb_id, pdb_obj, heteroatoms)

    # 3. Write clean protein for MUSTANG
    mustang_input = write_mustang_input(pdb_id, protein)

    # Save reference path immediately after writing so it is
    # available to subsequent structures even if this structure
    # later fails at a downstream step.
    if pdb_id == ref and ref_mustang_pdb is None:
        ref_mustang_pdb = mustang_input

    if pdb_id == ref:
        ref_mustang_pdb = mustang_input
        R = np.eye(3)
        t = np.zeros(3)
        rmsd_mustang = 0.0
        n_aligned_res = len(set(protein.getResindices()))
        print(f"    Reference structure - using identity transform")
    else:
        if ref_mustang_pdb is None:
            raise RuntimeError(
                f"Reference PDB ({ref}) has not been processed yet. "
                f"Make sure {ref} is in your pdb_ids.txt."
            )
        # 4. MUSTANG alignment
        print(f"    Running MUSTANG: {pdb_id} → {ref}...")
        R, t, rmsd_mustang, n_aligned_res = run_mustang_and_get_transform(
            mustang_input, ref_mustang_pdb
        )
        print(f"    MUSTANG RMSD = {rmsd_mustang:.3f} Å "
              f"f({n_aligned_res} residues aligned)")

    # 5. Apply transform to protein AND ligand
    prot_coords_aligned = apply_transform(protein.getCoords(), R, t)
    lig_coords_aligned = apply_transform(drug_ligand.getCoords(), R, t)

    protein.setCoords(prot_coords_aligned)
    drug_ligand.setCoords(lig_coords_aligned)

    full_aligned_prot_pdb = WORK_DIR / f"{pdb_id}_fullprot_aligned.pdb"
    writePDB(str(full_aligned_prot_pdb), protein)

    aligned_lig = ALIGNED_DIR / f"{pdb_id}_ligand_aligned.pdb"
    writePDB(str(aligned_lig), drug_ligand)
    aligned_lig_coords[pdb_id] = lig_coords_aligned

    # 6. Select KLIFS chain (whole chain, no residues removed)
    pocket = select_klifs_chain(protein, pdb_id)
    pocket_aligned_pdb = POCKET_DIR / f"{pdb_id}_pocket_aligned.pdb"
    writePDB(str(pocket_aligned_pdb), pocket)

    # 7. Clean ligand with RDKit (pH 7.4, correct bond orders)
    lig_clean_pdb = WORK_DIR / f"{pdb_id}_ligand_clean.pdb"
    lig_clean_sdf = WORK_DIR / f"{pdb_id}_ligand_clean.sdf"
    lig_raw_pdb = WORK_DIR / f"{pdb_id}_ligand_raw.pdb"

```

```

writePDB(str(lig_raw_pdb), drug_ligand)

lig_ok, lig_mol = prepare_ligand_rdkit(pdb_id, drug_resname,
                                       lig_raw_pdb, lig_clean_pdb, log,
                                       out_sdf=lig_clean_sdf)

if not lig_ok:
    print(f"    Ligand RDKit prep failed - using raw aligned PDB as fallback")
    shutil.copy(lig_raw_pdb, lig_clean_pdb)
    lig_mol = None
    log.setdefault("lig_ph_method", "failed-raw-fallback")
    log.setdefault("lig_bond_orders", "failed-raw-fallback")

# 8. Protonate and relax pocket protein
protH_relaxed, prot_method = protonate_and_relax_protein(
    pdb_id, pocket_aligned_pdb
)
log["rec_prot_method"] = prot_method
aligned_pocket_paths[pdb_id] = protH_relaxed

# 9. Ligand to PDBQT
ligand_to_pdbqt(pdb_id, lig_clean_pdb, log, mol_prepared=lig_mol)

# 10. Receptor to PDBQT (AutoDock4)
receptor_to_pdbqt(pdb_id, protH_relaxed, log)

log["status"] = "ok"
prep_logs.append(log)
done += 1

except Exception as e:
    import traceback
    print(f"    FAILED: {e}")
    traceback.print_exc()
    log["status"] = "failed"
    log["notes"] = str(e)
    prep_logs.append(log)
    failed.append((pdb_id, str(e)))

# POST-PROCESSING: global box and Cα RMSD

print(f"\n{'='*60}")
print("  POST-PROCESSING")
print(f"{'='*60}")

# Global docking box
if aligned_lig_coords:
    global_box = compute_global_box(aligned_lig_coords)
    box_path = OUTPUT_DIR / "global_docking_box.json"
    with open(box_path, "w") as f:
        json.dump(global_box, f, indent=2)
    print(f"    Global docking box: {box_path.name}")
else:
    print("    No aligned ligand coordinates - cannot compute global box")

# Cα RMSD on KLIFS active site
if aligned_pocket_paths:
    print("\n    Computing KLIFS Cα RMSD...")
    ca_coords, shared_res = collect_klifs_ca_coords(aligned_pocket_paths)
    if len(ca_coords) >= 2:
        pdb_ids, rmsd_mat = compute_rmsd_matrix(ca_coords)
        save_rmsd_results(pdb_ids, rmsd_mat, reference=ref)
    else:
        print("    Fewer than 2 structures with KLIFS residues - skipping RMSD")
else:
    print("    No aligned pocket proteins found - skipping RMSD")

# Summary

# Write preparation audit CSV
import csv

```

```

audit_path = OUTPUT_DIR / "preparation_audit.csv"
if prep_logs:
    fieldnames = list(prep_logs[0].keys())
    with open(audit_path, "w", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(prep_logs)
    print(f"\n    Preparation audit: {audit_path.name}")

# Identify anything that hit a fallback or failed
fallbacks = [
    lg for lg in prep_logs
    if "fallback" in lg.get("lig_ph_method", "")
    or "fallback" in lg.get("lig_bond_orders", "")
    or "fallback" in lg.get("lig_pdbqt_method", "")
    or "fallback" in lg.get("rec_pdbqt_method", "")
    or "failed" in lg.get("lig_pdbqt_method", "")
    or "failed" in lg.get("rec_pdbqt_method", "")
    or "failed" in lg.get("rec_prot_method", "")
    or lg.get("status") == "failed"
]

print(f"\n{'='*60}")
print(f" Pipeline complete")
print(f" Processed: {done}")
print(f" Failed: {len(failed)}")
for pid, reason in failed:
    print(f" {pid}: {reason}")

if fallbacks:
    print(f"\n — Preparation warnings / fallbacks ({len(fallbacks)}) —")
    print(f" {'PDB':<8} {'lig_ph':<25} {'lig_bonds':<30} {'lig_pdbqt':<22} {'rec_prot':<28} {'rec_pdbqt':<22}")
    print(f" {'-'*8} {'-'*25} {'-'*30} {'-'*22} {'-'*28} {'-'*22}")
    for lg in fallbacks:
        print(f" {lg['pdb_id']:<8} "
              f"{lg['lig_ph_method']:<25} "
              f"{lg['lig_bond_orders']:<30} "
              f"{lg['lig_pdbqt_method']:<22} "
              f"{lg['rec_prot_method']:<28} "
              f"{lg['rec_pdbqt_method']:<22}")
    else:
        print(f"\n    All structures used primary preparation methods")

print(f"\n PDBQT outputs: {PDBQT_DIR}")
print(f" Aligned PDBs: {ALIGN_DIR}")
print(f" Pocket PDBs: {POCKET_DIR}")
print(f" RMSD outputs: {RMSD_DIR}")
print(f" Audit log: {audit_path}")
print(f"{'='*60}")

# ENTRY POINT (COMMENT THIS OUT, WAS FOR PROBLEM LIGANDS)

if __name__ == "__main__":
    args = sys.argv[1:]
    rmsd_only = "--rmsd-only" in args
    pdb_args = [a for a in args if not a.startswith("--")]

    if pdb_args:
        pdb_list = [p.lower() for p in pdb_args]
        print(f"Processing {len(pdb_list)} PDB IDs from command line.")
    else:
        if not PDB_IDS_FILE.exists():
            raise FileNotFoundError(
                f"PDB ID list not found: {PDB_IDS_FILE}\n"
                f"Create that file (one PDB ID per line) or pass IDs as args:\n"
                f"python process_pdb_ids.py 3ddq 4cfv loiu"
            )
        with open(PDB_IDS_FILE) as f:

```

```

        pdb_list = [line.strip().lower() for line in f if line.strip()]
        print(f"Loaded {len(pdb_list)} PDB IDs from {PDB_IDS_FILE}")

run_pipeline(pdb_list, rmsd_only=rmsd_only)

```

7.4.2 Cluster Ligands via α C RMSD using GMM selection of clusters

```

#!/usr/bin/env python3
"""
ac_helix_rmsd_classifier.py

Automated conformational classification of CDK2 crystal structures
from  $\alpha$ C helix C $\alpha$  RMSD using Gaussian Mixture Model selection. Needed to do this
in order to conformationally cluster the active site conformations

Step in the script:
- Compute  $\alpha$ C helix (residues 43-53) C $\alpha$  RMSD vs reference for every
  aligned pocket PDB
- Plot KDE of the RMSD distribution to visualise natural clusters
- Fit GMM for n=1..8 components, plot BIC and AIC to select optimal n.
- Fit final GMM with optimal n, assign cluster labels ordered by
  mean RMSD (0 = most similar to reference, n-1 = most different).
- Plot  $\alpha$ C RMSD vs pIC50 coloured by GMM-assigned cluster.

All figures saved to OUTPUT_DIR. A CSV with per-structure RMSD,
GMM cluster, and validation columns is also written.

How to run:
    python ac_helix_rmsd_classifier.py

Dependencies you need to install:
    pip install pandas openpyxl matplotlib seaborn scikit-learn scipy
    prody already in docking_env
"""

from pathlib import Path
import warnings
import numpy as np
import pandas as pd
import prody
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
from scipy import stats
from sklearn.mixture import GaussianMixture

# USER CONFIG - change with your own paths

ALIGNED_POCKET_DIR = Path(
    r"C:\Users\rremp\Documents\Protein_Project\docking_redo\mustang_global\output\pocket_pdbs"
) # contains <pdb_id>_pocket_aligned.pdb files
EXCEL_FILE = Path(
    r"C:\Users\rremp\Documents\Protein_Project\CDK2_PDBIDs_pIC50s.xlsx"
)
REFERENCE_PDB_ID = "1e1v" # IMPORTANT THIS IS YOUR REFERENCE STRUCTURE, WHICH ALL OTHERS
# ARE COMPARED TO, IDEALLY SET IT TO SOMETHING WHICH REPRESENTS THE ACTIVE SITE OF THE
# PROTEIN WHEN THE NATIVE LIGAND IS BOUND

OUTPUT_DIR = ALIGNED_POCKET_DIR.parent / "rmsd_classifier"
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

# CDK2  $\alpha$ C helix residues (inclusive)
AC_HELIX_RESIDS = set(range(43, 54))

# GMM search range
N_COMPONENTS_RANGE = range(1, 9)

```

```

# MATPLOTLIB DEFAULTS - graph settings

matplotlib.rcParams.update({
    "font.family":      "sans-serif",
    "font.size":        12,
    "axes.linewidth":   1.2,
    "xtick.major.width": 1.2,
    "ytick.major.width": 1.2,
    "xtick.direction":  "out",
    "ytick.direction":  "out",
})

prody.confProDy(verbosity="none")

# STEP 1 - RMSD COMPUTATION

def find_pocket_pdb(pdb_id: str) -> Path | None:
    # pocket_aligned.pdb is written directly by ProDy after MUSTANG transform
    # and has guaranteed correct coordinates. pocketH_relaxed goes through
    # PDBFixer/OpenMM which can shift coordinates which is why we use this file.
    for suffix in ("_pocket_aligned.pdb", "_pocketH_relaxed.pdb"):
        for case in (pdb_id.lower(), pdb_id.upper()):
            p = ALIGNED_POCKET_DIR / f"{case}{suffix}"
            if p.exists():
                return p
    return None

def get_ac_helix_ca(pdb_path: Path) -> dict:
    """Return {resnum: coords (3,)} for αC helix Cα atoms."""
    prot = prody.parsePDB(str(pdb_path))
    if prot is None:
        return {}
    ca = prot.select("name CA")
    if ca is None:
        return {}
    return {
        atom.getResnum(): atom.getCoords().copy()
        for atom in ca.iterAtoms()
        if atom.getResnum() in AC_HELIX_RESIDS
    }

def ca_rmsd(coords_a: np.ndarray, coords_b: np.ndarray) -> float:
    """No-superposition Cα RMSD - structures already aligned by MUSTANG."""
    diff = coords_a - coords_b
    return float(np.sqrt((diff ** 2).sum(axis=1).mean()))

def compute_all_rmsds(df: pd.DataFrame) -> pd.DataFrame:
    ref_path = find_pocket_pdb(REFERENCE_PDB_ID)
    if ref_path is None:
        raise FileNotFoundError(
            f"Reference PDB not found for {REFERENCE_PDB_ID} in {ALIGNED_POCKET_DIR}"
        )
    ref_map = get_ac_helix_ca(ref_path)
    if not ref_map:
        raise ValueError(
            f"No αC helix residues (43-53) found in reference: {ref_path}"
        )
    print(f"Reference ({REFERENCE_PDB_ID}): {len(ref_map)} αC Cα residues")

    records = []
    missing_pdb, missing_ca = [], []

    for _, row in df.iterrows():
        pdb_id = row["pdb_id"]
        pdb_path = find_pocket_pdb(pdb_id)

        if pdb_path is None:
            missing_pdb.append(pdb_id)
            continue

```

```

mob_map = get_ac_helix_ca(pdb_path)
if not mob_map:
    missing_ca.append(pdb_id)
    continue

shared = sorted(set(ref_map) & set(mob_map))
if len(shared) < 3:
    print(f"    {pdb_id}: only {len(shared)} shared  $\alpha$ C C $\alpha$  - skipping")
    missing_ca.append(pdb_id)
    continue

ref_arr = np.array([ref_map[r] for r in shared])
mob_arr = np.array([mob_map[r] for r in shared])

records.append({
    "pdb_id":      pdb_id,
    "ligand_code": row.get("Ligand Code", ""),
    "pIC50":       row.get("pIC50", np.nan),
    "ac_helix_xl": str(row.get("AC_HELIX", "")).strip().lower(),
    "dfg_conf_xl": str(row.get("DFG Conf A/B", "")).strip().upper(),
    "n_shared_ca": len(shared),
    "ac_rmsd_A":   ca_rmsd(ref_arr, mob_arr),
})

if missing_pdb:
    print(f" Missing PDB files ({len(missing_pdb)}): {missing_pdb}")
if missing_ca:
    print(f" Missing  $\alpha$ C residues ({len(missing_ca)}): {missing_ca}")

print(f" RMSD computed for {len(records)} structures\n")
return pd.DataFrame(records)

# STEP 2: KDE PLOT - kernel density estimate of raw distribution to examine for clusters at the
start

def plot_kde(results: pd.DataFrame):
    """
    KDE of the raw RMSD distribution.
    Rug plot underneath shows individual structures.
    """
    fig, ax = plt.subplots(figsize=(8, 4.5))

    sns.kdeplot(
        data=results,
        x="ac_rmsd_A",
        bw_adjust=0.5,          # narrower bandwidth to resolve tight clusters
        fill=True,
        color="#5B8DB8",
        alpha=0.35,
        ax=ax,
    )
    sns.rugplot(
        data=results,
        x="ac_rmsd_A",
        color="#2C5F8A",
        height=0.06,
        alpha=0.6,
        ax=ax,
    )

    ax.set_xlabel(" $\alpha$ C Helix C $\alpha$  RMSD vs reference ( $\text{\AA}$ )", fontsize=13)
    ax.set_ylabel("Density", fontsize=13)
    ax.set_title("Distribution of  $\alpha$ C Helix RMSD - CDK2 crystal poses", fontsize=13)
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)

    plt.tight_layout()
    out = OUTPUT_DIR / "01_kde_ac_rmsd.png"
    fig.savefig(out, dpi=300, bbox_inches="tight")
    fig.savefig(out.with_suffix(".svg"), bbox_inches="tight")

```

```

print(f" KDE → {out}")
plt.show()

# STEP 3 – BIC / AIC MODEL SELECTION - determines the clustering and how many clusters are used
# GMM is done in scikit-learn and samples kernel widths to determine the best kernel width, as
well as
# minimize complexity of the model while also adding the max amount of clusters
# determines where there are diminishing returns for adding more clusters vs. adding complexity
to the model

def select_n_components(rmsd_vals: np.ndarray) -> int:
    """
    Fit GMM for each n in N_COMPONENTS_RANGE.
    Plot BIC and AIC. Return the n with the lowest BIC.
    """
    X      = rmsd_vals.reshape(-1, 1)
    bic    = []
    aic    = []
    models = []

    for n in N_COMPONENTS_RANGE:
        gmm = GaussianMixture(
            n_components=n,
            covariance_type="full",
            n_init=20,      # set this for multiple restarts to avoid local minima
            random_state=42,
        )
        gmm.fit(X)
        bic.append(gmm.bic(X))
        aic.append(gmm.aic(X))
        models.append(gmm)

    best_n    = list(N_COMPONENTS_RANGE)[int(np.argmin(bic))]
    best_gmm  = models[int(np.argmin(bic))]

    # Plot:
    fig, ax = plt.subplots(figsize=(7, 4.5))

    ns = list(N_COMPONENTS_RANGE)
    ax.plot(ns, bic, "o-", color="#D95F02", linewidth=1.8, markersize=7, label="BIC")
    ax.plot(ns, aic, "s--", color="#1B9E77", linewidth=1.8, markersize=7, label="AIC")
    ax.axvline(x=best_n, color="#D95F02", linewidth=1.2,
               linestyle=":", alpha=0.7, label=f"BIC optimum n={best_n}")

    ax.set_xlabel("Number of GMM components", fontsize=13)
    ax.set_ylabel("Score (lower = better)", fontsize=13)
    ax.set_title("GMM model selection – αC helix RMSD", fontsize=13)
    ax.set_xticks(ns)
    ax.legend(fontsize=10)
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)

    plt.tight_layout()
    out = OUTPUT_DIR / "02_gmm_bic_aic.png"
    fig.savefig(out, dpi=300, bbox_inches="tight")
    fig.savefig(out.with_suffix(".svg"), bbox_inches="tight")
    print(f" BIC/AIC → {out}")
    plt.show()

    print(f"\nBIC scores: { {n: f'{b:.1f}' for n, b in zip(ns, bic)} }")
    print(f"Optimal n (BIC) = {best_n}\n")

    return best_n, best_gmm, models

# STEP 4 – FIT FINAL GMM AND ASSIGN CLUSTERS - re-clustering after gmm calculations performed

def assign_clusters(results: pd.DataFrame, best_n: int, best_gmm: GaussianMixture) ->
pd.DataFrame:
    """
    Assign each structure to a GMM cluster ordered by mean RMSD.

```

```

Cluster 0 = lowest RMSD (most similar to reference).
Also compute posterior probabilities for each cluster.
"""
X          = results["ac_rmsd_A"].values.reshape(-1, 1)
raw_labels = best_gmm.predict(X)
probs      = best_gmm.predict_proba(X)    # shape (N, n_components)

# Re-order clusters by ascending mean RMSD
means = best_gmm.means_.flatten()
order = np.argsort(means)                # order[0] = index of lowest-mean component
remap = {old_idx: new_idx for new_idx, old_idx in enumerate(order)}

results = results.copy()
results["gmm_cluster"] = [remap[c] for c in raw_labels]

# Posterior probabilities, re-ordered
for new_idx, old_idx in enumerate(order):
    results[f"prob_cluster_{new_idx}"] = probs[:, old_idx]

# Component means and stds (for reporting)
comp_means = means[order]
comp_stds  = np.sqrt(best_gmm.covariances_.flatten()[order])
comp_weights = best_gmm.weights_[order]

print(f"GMM components (n={best_n}), ordered by mean RMSD:")
for i in range(best_n):
    n_assigned = (results["gmm_cluster"] == i).sum()
    print(f"Cluster {i}: mean={comp_means[i]:.3f} Å  "
          f"std={comp_stds[i]:.3f} Å  "
          f"weight={comp_weights[i]:.3f}  "
          f"n={n_assigned}")

return results, comp_means, comp_stds

# STEP 5 - SCATTER: RMSD vs pIC50 coloured by GMM CLUSTER - replot after gmm clustering

# Palette: one colour per cluster, assigned after we know n
CLUSTER_PALETTE = [
    "#5B8DB8", # cluster 0 - low RMSD (αC-in-like)
    "#E8956D", # cluster 1
    "#D4688E", # cluster 2
    "#6DBE9B", # cluster 3
    "#9B7FD4", # cluster 4 (unlikely to need this many)
    "#D4C96D",
    "#D47B6D",
    "#6DB8D4",
]

def plot_rmsd_vs_pic50(results: pd.DataFrame, best_n: int,
                      comp_means: np.ndarray, comp_stds: np.ndarray):
    """
    Scatter: x = αC RMSD, y = pIC50, colour = GMM cluster.
    Vertical shaded bands show ±1 std of each GMM component.
    Pearson r and Spearman ρ over all data in annotation box.
    """
    plot_df = results.dropna(subset=["ac_rmsd_A", "pIC50"]).copy()

    pearson_r, _ = stats.pearsonr(plot_df["ac_rmsd_A"], plot_df["pIC50"])
    spearman_r, _ = stats.spearmanr(plot_df["ac_rmsd_A"], plot_df["pIC50"])
    n_total = len(plot_df)

    fig, ax = plt.subplots(figsize=(8.5, 6.5))

    # Shaded GMM component bands:
    for i in range(best_n):
        mu = comp_means[i]
        sig = comp_stds[i]
        ax.axvspan(
            max(0, mu - sig), mu + sig,

```

```

        alpha=0.08,
        color=CLUSTER_PALETTE[i % len(CLUSTER_PALETTE)],
        zorder=1,
        label=None,
    )
    ax.axvline(mu, color=CLUSTER_PALETTE[i % len(CLUSTER_PALETTE)],
              linewidth=1.0, linestyle="--", alpha=0.45, zorder=2)

# Scatter per cluster:
for c in range(best_n):
    sub = plot_df[plot_df["gmm_cluster"] == c]
    ax.scatter(
        sub["ac_rmsd_A"],
        sub["pIC50"],
        c=CLUSTER_PALETTE[c % len(CLUSTER_PALETTE)],
        s=60,
        alpha=0.78,
        linewidths=0.4,
        edgecolors="black",
        label=f"Cluster {c}    (n={len(sub)},  $\mu$ ={comp_means[c]:.2f} Å)",
        zorder=3,
    )

# Labels & limits:
ax.set_xlabel("αC Helix Cα RMSD vs reference (Å)", fontsize=13, labelpad=8)
ax.set_ylabel("pIC50", fontsize=13, labelpad=8)
ax.set_title(f"CDK2 αC Helix Conformation vs Potency\n"
            f"(GMM, n={best_n} components)", fontsize=13, pad=10)

x_max = plot_df["ac_rmsd_A"].max()
ax.set_xlim(left=-0.05, right=x_max * 1.08)
ax.set_ylim(
    bottom=plot_df["pIC50"].min() - 0.3,
    top=plot_df["pIC50"].max() + 0.4,
)

# Correlation annotation
stat_text = (
    f"Pearson r = {pearson_r:.3f}\n"
    f"Spearman ρ = {spearman_r:.3f}\n"
    f"n = {n_total}"
)
ax.text(
    0.03, 0.97, stat_text,
    transform=ax.transAxes,
    fontsize=10,
    verticalalignment="top",
    bbox=dict(boxstyle="round,pad=0.4", facecolor="white",
              edgecolor="black", linewidth=0.8),
)

ax.legend(title="GMM cluster", title_fontsize=10,
          fontsize=9, loc="upper right",
          framealpha=0.9, edgecolor="black")
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.tick_params(labelsize=11)

plt.tight_layout()
out = OUTPUT_DIR / "04_rmsd_vs_pic50_gmm.png"
fig.savefig(out, dpi=300, bbox_inches="tight")
fig.savefig(out.with_suffix(".svg"), bbox_inches="tight")
print(f"  Scatter → {out}")
plt.show()

# KDE coloured by GMM cluster (overlay)

def plot_kde_by_cluster(results: pd.DataFrame, best_n: int):
    """
    KDE of RMSD distribution with each GMM cluster shown separately.

```

```

Useful for confirming the clusters are well-separated.
"""
fig, ax = plt.subplots(figsize=(8, 4.5))

for c in range(best_n):
    sub = results[results["gmm_cluster"] == c]["ac_rmsd_A"]
    sns.kdeplot(
        sub,
        bw_adjust=0.6,
        fill=True,
        color=CLUSTER_PALETTE[c % len(CLUSTER_PALETTE)],
        alpha=0.35,
        label=f"Cluster {c} (n={len(sub)} )",
        ax=ax,
    )
    sns.rugplot(
        sub,
        color=CLUSTER_PALETTE[c % len(CLUSTER_PALETTE)],
        height=0.06,
        alpha=0.7,
        ax=ax,
    )

ax.set_xlabel("αC Helix Cα RMSD vs reference (Å)", fontsize=13)
ax.set_ylabel("Density", fontsize=13)
ax.set_title(f"αC Helix RMSD distribution by GMM cluster (n={best_n})", fontsize=13)
ax.legend(fontsize=10)
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)

plt.tight_layout()
out = OUTPUT_DIR / "05_kde_by_cluster.png"
fig.savefig(out, dpi=300, bbox_inches="tight")
fig.savefig(out.with_suffix(".svg"), bbox_inches="tight")
print(f" KDE by cluster → {out}")
plt.show()

# MAIN
def main():
    # Load Excel
    df = pd.read_excel(EXCEL_FILE)
    df.columns = df.columns.str.strip()
    df["pdb_id"] = df["pdb_id"].str.lower().str.strip()
    print(f"Loaded {len(df)} entries from {EXCEL_FILE.name}\n")

    # Step 1: RMSD
    print("STEP 1 – Computing αC helix Cα RMSD")
    results = compute_all_rmsds(df)

    # Step 2: KDE
    print("STEP 2 – KDE of raw RMSD distribution")
    plot_kde(results)

    # Step 3: BIC/AIC
    print("STEP 3 – GMM model selection (BIC / AIC)")
    rmsd_vals = results["ac_rmsd_A"].values
    best_n, best_gmm, _ = select_n_components(rmsd_vals)

    # Step 4: Assign clusters
    print(f"STEP 4 – Assigning clusters (n={best_n})")
    results, comp_means, comp_stds = assign_clusters(results, best_n, best_gmm)

    # Step 5: Validate vs Excel labels
    print("STEP 5 – Cross-tabulation vs Excel labels")
    results = validate_against_excel(results, best_n)

    # Step 6: Scatter plot
    print("STEP 6 – Scatter: RMSD vs pIC50")
    plot_rmsd_vs_pic50(results, best_n, comp_means, comp_stds)

```

```

# KDE by cluster
plot_kde_by_cluster(results, best_n)

# Save full results CSV
csv_path = OUTPUT_DIR / "ac_helix_rmsd_classified.csv"
results.to_csv(csv_path, index=False)
print(f"\n Full results to {csv_path}")
print(f"    Columns: {list(results.columns)}")

print(f" All outputs to {OUTPUT_DIR}")

if __name__ == "__main__":
    main()

```

7.4.3 Cleanup and Prepare Ligands for ROCS Calculations

```

#!/usr/bin/env python3
"""
cleanup_and_rocs_prep.py

performs two jobs in one script:

cleanup (removes all problematic files in directory)
rocs prep (converts aligned crystal ligands to SDF for OpenEye ROCS shape/colour similarity
screening,
with a quality check on each ligand)

cleanup checks (on raw PDB file):
    multiple conformers merged, Low occupancy atoms (< 0.5), low resolution (b-factor > 60 Å²),
    too high resolution (b-factor < 5 Å²), missing atoms vs expected count from SMILES,
    disconnected fragments

cleanup checks (on RDKit mol after conversion):
    Zero coordinates, correct atom count vs SMILES, aromatic atom types present,
    3D coordinates present, stereocentres assigned

Outputs (in output/rocs/):
    all_ligands_crystal_poses.sdf    - all ligands in one SDF, this is split in the ROCS scripts
later
    *pdb_id*_ligand.sdf             - one SDF per structure
    rocs_prep_report.csv            - full quality report
    rocs_prep_issues.csv            - only flagged structures which had issues

Usage:
    python cleanup_and_rocs_prep.py

Dependencies:
    pip install rdkit pandas openpyxl requests
"""

from pathlib import Path
import numpy as np
import pandas as pd
import requests
import warnings
from rdkit import Chem
from rdkit.Chem import AllChem, rdMolDescriptors

# USER CONFIG - update with your pathways

PROJECT_DIR = Path(r"C:\Users\rremp\Documents\Protein_Project\docking_redo\mustang_global")
OUTPUT_DIR = PROJECT_DIR / "output"
ALIGNED_DIR = OUTPUT_DIR / "aligned_pdb"
POCKET_DIR = OUTPUT_DIR / "pocket_pdb"
PDBQT_DIR = OUTPUT_DIR / "pdbqt"
WORK_DIR = PROJECT_DIR / "work"
ROCS_DIR = OUTPUT_DIR / "rocs"

```

```

CLASSIFIER_CSV = OUTPUT_DIR / "rmsd_classifier" / "ac_helix_rmsd_classified.csv"
EXCEL_FILE =
Path(r"C:\Users\rrempe\Documents\Protein_Project\docking_redo\CDK2_PDBIDs_pIC50s.xlsx")

ROCS_DIR.mkdir(parents=True, exist_ok=True)

# Structures to remove - can ignore this if all your ligands run properly and don't have issues,
# but if
# you have problem ligands put them here and they will be removed from the ROCS run

REMOVE_IDS = {
    "1pxo": "incomplete crystal - nitrophenyl ring not resolved",
    "6q3c": "FragLite - not in activity dataset",
    "6q4f": "FragLite - not in activity dataset",
    "6q48": "FragLite - not in activity dataset",
}

# B-factor thresholds
BFAC_HIGH = 60.0 # Å² - very disordered
BFAC_LOW = 5.0 # Å² - likely placeholder/unresolved
OCC_MIN = 0.5 # minimum acceptable occupancy

# STEP 1) CLEANUP OF PROTEINS

def cleanup():
    print("=" * 60)
    print("STEP 1: CLEANUP")
    print("=" * 60)

    patterns = [
        PDBQT_DIR / "{id}_ligand.pdbqt",
        PDBQT_DIR / "{id}_receptor.pdbqt",
        ALIGNED_DIR / "{id}_ligand_aligned.pdb",
        POCKET_DIR / "{id}_pocket_aligned.pdb",
        POCKET_DIR / "{id}_pocketH_relaxed.pdb",
        WORK_DIR / "{id}_ligand_raw.pdb",
        WORK_DIR / "{id}_ligand_clean.pdb",
        WORK_DIR / "{id}_fullprot_aligned.pdb",
        WORK_DIR / "{id}_pocketH.pdb",
        WORK_DIR / "{id}_chainA_for_mustang.pdb",
    ]

    for pdb_id, reason in REMOVE_IDS.items():
        print(f"\n {pdb_id.upper()} - {reason}")
        removed = 0
        for pattern in patterns:
            p = Path(str(pattern).replace("{id}", pdb_id.lower()))
            if p.exists():
                p.unlink()
                print(f"    {p.name}")
                removed += 1
        print(f"    ({removed} files removed)")

    print(f"\n Done - {len(REMOVE_IDS)} structures removed\n")

# HELPERS

def fetch_smiles(resname: str) -> str | None:
    """Fetch canonical SMILES from RCSB CCD."""
    url = f"https://data.rcsb.org/rest/v1/core/chemcomp/{resname}"
    try:
        r = requests.get(url, timeout=8)
        r.raise_for_status()
        data = r.json()
        for entry in data.get("pdbx_chem_comp_descriptor", []):
            if "SMILES" in entry.get("type", ""):
                return entry.get("descriptor")
    except Exception:
        pass

```

```

return None

def parse_ligand_pdb(path: Path) -> dict:
    """
    Parse a ligand PDB file and extract quality metrics.
    Returns dict with per-atom data and summary statistics.
    """
    atoms = []
    altlocs_seen = set()

    with open(path, encoding="utf-8", errors="replace") as f:
        for line in f:
            if not line.startswith(("ATOM", "HETATM")):
                continue
            try:
                name = line[12:16].strip()
                altloc = line[16].strip()
                resname = line[17:20].strip()
                x = float(line[30:38])
                y = float(line[38:46])
                z = float(line[46:54])
                occ = float(line[54:60]) if line[54:60].strip() else 1.0
                bfac = float(line[60:66]) if line[60:66].strip() else 0.0
                element = line[76:78].strip().upper() if len(line) > 76 else ""

                # Skip H atoms
                if element == "H" or name.startswith("H"):
                    continue

                if altloc:
                    altlocs_seen.add(altloc)

                atoms.append({
                    "name": name, "altloc": altloc, "resname": resname,
                    "x": x, "y": y, "z": z,
                    "occ": occ, "bfac": bfac, "element": element,
                })
            except (ValueError, IndexError):
                continue

    if not atoms:
        return {"n_atoms": 0, "error": "no atoms parsed"}

    bfacs = [a["bfac"] for a in atoms]
    occs = [a["occ"] for a in atoms]
    coords = np.array([[a["x"], a["y"], a["z"]] for a in atoms])

    # Check for disconnected fragments (rough: are any atoms > 5Å from all others?)
    n_zero = sum(1 for c in coords
                  if abs(c[0]) < 0.01 and abs(c[1]) < 0.01 and abs(c[2]) < 0.01)

    # Fragment check: build rough distance matrix, find isolated atoms
    n_isolated = 0
    if len(coords) > 1:
        from scipy.spatial.distance import cdist
        dists = cdist(coords, coords)
        np.fill_diagonal(dists, np.inf)
        min_dists = dists.min(axis=1)
        n_isolated = int((min_dists > 4.0).sum()) # atoms with no neighbour < 4Å

    return {
        "n_atoms": len(atoms),
        "altlocs": sorted(altlocs_seen),
        "has_altloc": len(altlocs_seen) > 1,
        "mean_bfac": float(np.mean(bfacs)),
        "max_bfac": float(np.max(bfacs)),
        "min_bfac": float(np.min(bfacs)),
        "min_occ": float(np.min(occs)),
        "n_low_occ": sum(1 for o in occs if o < OCC_MIN),
        "n_zero_coords": n_zero,
    }

```

```

        "n_isolated_atoms": n_isolated,
        "centre":          coords.mean(axis=0).tolist(),
    }

def sdf_to_mol(path: Path) -> Chem.Mol | None:
    """
    Load the Step-1 clean SDF (written by process_pdb_mustang.py's
    prepare_ligand_rdkit) directly. This SDF already carries the correct
    bond orders/aromaticity assigned from the RCSB SMILES template – unlike
    the PDB file, which cannot encode bond orders at all.

    Preferred over pdb_to_mol() whenever the SDF is available: it avoids
    re-deriving bonds from geometry a second, independent time (which is
    what pdb_to_mol's MolFromPDBFile + SanitizeMol path does, and which can
    get aromaticity wrong for fused-ring systems
    """
    try:
        suppl = Chem.SDMolSupplier(str(path), removeHs=True, sanitize=True)
        mol = next((m for m in suppl if m is not None), None)
        return mol
    except Exception:
        return None

def pdb_to_mol(path: Path) -> Chem.Mol | None:
    """
    Load PDB → RDKit mol, keep largest fragment, sanitize.

    WARNING: this re-derives bond orders/aromaticity from atomic distances
    alone (MolFromPDBFile has no bond-order information to work with, since
    PDB format cannot encode it). This is independent of, and can disagree
    with, the RCSB-SMILES-template bond orders already assigned upstream in
    process_pdb_mustang.py. Use sdf_to_mol() instead whenever the Step-1
    clean SDF is available; this function exists only as a fallback for
    structures where that SDF is missing (e.g. older pipeline runs, or
    structures where Step 1's RDKit prep failed and fell back to the raw
    PDB).
    """
    mol = Chem.MolFromPDBFile(str(path), removeHs=True, sanitize=False)
    if mol is None:
        return None
    frags = Chem.GetMolFrags(mol, asMols=True, sanitizeFrags=False)
    if not frags:
        return None
    mol = max(frags, key=lambda m: m.GetNumAtoms())
    try:
        Chem.SanitizeMol(mol)
    except Exception:
        try:
            Chem.SanitizeMol(mol,
                             Chem.SanitizeFlags.SANITIZE_ALL ^
                             Chem.SanitizeFlags.SANITIZE_PROPERTIES)
        except Exception:
            return None
    return mol

def load_metadata() -> pd.DataFrame:
    meta = pd.DataFrame()
    if CLASSIFIER_CSV.exists():
        clf = pd.read_csv(CLASSIFIER_CSV)
        clf["pdb_id"] = clf["pdb_id"].str.lower().str.strip()
        # Only keep columns that actually exist in the CSV
        keep = [c for c in ["pdb_id", "ac_rmsd_A", "gmm_cluster",
                           "conformation", "ligand_code", "pIC50"]
                if c in clf.columns]
        meta = clf[keep].copy()
    elif EXCEL_FILE.exists():
        xl = pd.read_excel(EXCEL_FILE)
        xl.columns = xl.columns.str.strip()

```

```

        xl["pdb_id"] = xl["pdb_id"].str.lower().str.strip()
        meta = xl[["pdb_id", "Ligand Code", "pIC50"]].rename(
            columns={"Ligand Code": "ligand_code"})
        return meta

# STEP 2: ROCS PREP + QUALITY REPORT

def prepare_rocs():

    print("STEP 2: ROCS SDF PREPARATION + QUALITY CHECK")

    meta = load_metadata()
    print(f"  Loaded metadata for {len(meta)} structures")

    # All aligned ligand PDBs excluding removed structures
    lig_files = sorted(ALIGNED_DIR.glob("*_ligand_aligned.pdb"))
    pdb_ids = [f.name.replace("_ligand_aligned.pdb", "").lower()
                for f in lig_files
                if f.name.replace("_ligand_aligned.pdb", "").lower()
                not in REMOVE_IDS]
    print(f"  Processing {len(pdb_ids)} ligands\n")

    writer_all = Chem.SDWriter(str(ROCS_DIR / "all_ligands_crystal_poses.sdf"))
    records = []
    n_ok = n_warn = n_fail = 0

    for pdb_id in sorted(pdb_ids):

        rec = {
            "pdb_id": pdb_id,
            # PDB checks
            "pdb_n_atoms": None,
            "pdb_mean_bfac": None,
            "pdb_max_bfac": None,
            "pdb_min_bfac": None,
            "pdb_min_occ": None,
            "pdb_has_altloc": None,
            "pdb_n_low_occ": None,
            "pdb_n_zero_coords": None,
            "pdb_n_isolated": None,
            # Mol checks
            "mol_n_heavy": None,
            "mol_has_aromatic": None,
            "mol_has_stereo": None,
            "mol_n_rot_bonds": None,
            "mol_zero_coords": None,
            # Flags
            "flag_high_bfac": False,
            "flag_low_bfac": False,
            "flag_low_occ": False,
            "flag_altloc": False,
            "flag_isolated": False,
            "flag_zero_coords": False,
            "flag_no_aromatics": False,
            "flag_parse_fail": False,
            "bond_source": None, # "sdf-template" or "pdb-geometry-fallback"
            "issues": "",
        }

        # Source: prefer the Step-1 clean SDF (carries correct, RCSB-SMILES-
        # template bond orders/aromaticity). Fall back to the clean PDB
        # (post-RDKit, but bonds re-derived from geometry since PDB cannot
        # store bond order), then to the raw aligned PDB as a last resort.
        sdf_candidate = WORK_DIR / f"{pdb_id}_ligand_clean.sdf"
        src_path = None
        from_sdf = False
        if sdf_candidate.exists():
            src_path = sdf_candidate
            from_sdf = True
        else:

```

```

for candidate in [
    WORK_DIR / f"{pdb_id}_ligand_clean.pdb",
    ALIGNED_DIR / f"{pdb_id}_ligand_aligned.pdb",
]:
    if candidate.exists():
        src_path = candidate
        break

if src_path is None:
    rec["flag_parse_fail"] = True
    rec["issues"] = "NO_SOURCE_FILE"
    records.append(rec)
    n_fail += 1
    print(f"    {pdb_id.upper():<10} no source file found")
    continue

# PDB-level checks
# Use raw aligned PDB for PDB checks (has original B-factors/occupancy)
raw_path = ALIGNED_DIR / f"{pdb_id}_ligand_aligned.pdb"
if raw_path.exists():
    pdb_info = parse_ligand_pdb(raw_path)
    if "error" not in pdb_info:
        rec["pdb_n_atoms"] = pdb_info["n_atoms"]
        rec["pdb_mean_bfac"] = round(pdb_info["mean_bfac"], 1)
        rec["pdb_max_bfac"] = round(pdb_info["max_bfac"], 1)
        rec["pdb_min_bfac"] = round(pdb_info["min_bfac"], 1)
        rec["pdb_min_occ"] = round(pdb_info["min_occ"], 2)
        rec["pdb_has_altloc"] = pdb_info["has_altloc"]
        rec["pdb_n_low_occ"] = pdb_info["n_low_occ"]
        rec["pdb_n_zero_coords"] = pdb_info["n_zero_coords"]
        rec["pdb_n_isolated"] = pdb_info["n_isolated_atoms"]

        issues = []
        if pdb_info["mean_bfac"] > BFAC_HIGH:
            rec["flag_high_bfac"] = True
            issues.append(f"HIGH_BFAC({pdb_info['mean_bfac']:.0f}Å²)")
        if pdb_info["min_bfac"] < BFAC_LOW and pdb_info["min_bfac"] > 0:
            rec["flag_low_bfac"] = True
            issues.append(f"LOW_BFAC({pdb_info['min_bfac']:.1f}Å²)")
        if pdb_info["n_low_occ"] > 0:
            rec["flag_low_occ"] = True
            issues.append(f"LOW_OCC({pdb_info['n_low_occ']}atoms)")
        if pdb_info["has_altloc"]:
            rec["flag_altloc"] = True
            issues.append(f"ALTLOC({pdb_info['altlocs']}")
        if pdb_info["n_isolated_atoms"] > 0:
            rec["flag_isolated"] = True
            issues.append(f"ISOLATED({pdb_info['n_isolated_atoms']}atoms)")
        if pdb_info["n_zero_coords"] > 0:
            rec["flag_zero_coords"] = True
            issues.append(f"ZERO_COORDS({pdb_info['n_zero_coords']}")

        rec["issues"] = "; ".join(issues)

# Molecule-level checks + SDF writing
mol = sdf_to_mol(src_path) if from_sdf else pdb_to_mol(src_path)
rec["bond_source"] = "sdf-template" if from_sdf else "pdb-geometry-fallback"
if mol is None:
    rec["flag_parse_fail"] = True
    rec["issues"] += (" " if rec["issues"] else "") + "PARSE_FAIL"
    records.append(rec)
    n_fail += 1
    print(f"    {pdb_id.upper():<10} RDKit parse failed")
    continue

rec["mol_n_heavy"] = mol.GetNumAtoms()
rec["mol_has_stereo"] = any(
    b.GetStereo() != Chem.BondStereo.STEREONONE
    for b in mol.GetBonds()
)
rec["mol_has_aromatic"] = any(a.GetIsAromatic() for a in mol.GetAtoms())

```

```

rec["mol_n_rot_bonds"] = rdMolDescriptors.CalcNumRotatableBonds(mol)

mol_issues = []
if mol.GetNumConformers() > 0:
    coords = mol.GetConformer().GetPositions()
    n_zero = sum(1 for c in coords
                  if abs(c[0]) < 0.01 and abs(c[1]) < 0.01 and abs(c[2]) < 0.01)
    rec["mol_zero_coords"] = n_zero > 0
    if n_zero > 0:
        rec["flag_zero_coords"] = True
        mol_issues.append(f"MOL_ZERO_COORDS({n_zero})")

if not rec["mol_has_aromatic"] and mol.GetNumAtoms() >= 6:
    rec["flag_no_aromatics"] = True
    mol_issues.append("NO_AROMATICS")

if mol_issues:
    rec["issues"] += ("; " if rec["issues"] else "") + "; ".join(mol_issues)

# Set SDF properties
mol.SetProp("_Name", pdb_id.upper())
mol.SetProp("pdb_id", pdb_id.upper())

row = meta[meta["pdb_id"] == pdb_id] if not meta.empty else pd.DataFrame()
if not row.empty:
    for col, prop in [("ligand_code", "ligand_code"),
                      ("pIC50", "pIC50"),
                      ("conformation", "conformation"),
                      ("ac_rmsd_A", "ac_helix_rmsd"),
                      ("gmm_cluster", "gmm_cluster")]:
        if col in row.columns:
            val = row[col].values[0]
            if not pd.isna(val):
                mol.SetProp(prop, str(val) if col != "ac_rmsd_A"
                           else f"{float(val):.3f}")

if rec["pdb_mean_bfac"] is not None:
    mol.SetProp("mean_bfac", str(rec["pdb_mean_bfac"]))
if rec["pdb_n_atoms"] is not None:
    mol.SetProp("n_heavy_atoms", str(rec["pdb_n_atoms"]))
if rec["issues"]:
    mol.SetProp("quality_flags", rec["issues"])

# Write SDF
writer_all.write(mol)
ind = ROCS_DIR / f"{pdb_id}_ligand.sdf"
with Chem.SDWriter(str(ind)) as w:
    w.write(mol)

any_flag = any([rec["flag_high_bfac"], rec["flag_low_occ"],
                rec["flag_altloc"], rec["flag_isolated"],
                rec["flag_zero_coords"], rec["flag_no_aromatics"]])

if any_flag:
    print(f" {pdb_id.upper():<10} {rec['issues']}")
    n_warn += 1
else:
    n_ok += 1

records.append(rec)

writer_all.close()

# Save reports
df = pd.DataFrame(records)
df.to_csv(ROCS_DIR / "rocs_prep_report.csv", index=False)

flag_cols = [c for c in df.columns if c.startswith("flag_")]
issues_df = df[df[flag_cols].any(axis=1)]
issues_df.to_csv(ROCS_DIR / "rocs_prep_issues.csv", index=False)

# Summary

```

```

print(f"  ROCS prep complete")
print(f"  Clean:          {n_ok}")
print(f"  Warnings:         {n_warn}")
print(f"  Failed:           {n_fail}")
print(f"\n  Bond-order source breakdown:")
if "bond_source" in df.columns:
    for src, n in df["bond_source"].value_counts(dropna=False).items():
        label = src if src is not None else "n/a (parse failed before source check)"
        print(f"      {label:<35} {n}")
    n_fallback = int((df["bond_source"] == "pdb-geometry-fallback").sum())
    if n_fallback > 0:
        print(f"      {n_fallback} structure(s) used geometry-based bond "
              f"inference (no Step-1 SDF found) - bond orders/aromaticity "
              f"for these were NOT taken from the RCSB-SMILES template and "
              f"may disagree with the docking-stage PDBQT chemistry.")
print(f"\n  Flag breakdown:")
for col in flag_cols:
    n = df[col].sum()
    if n > 0:
        label = col.replace("flag_", "").replace("_", " ").title()
        print(f"      {label:<20} {n}")
print(f"\n  Outputs in: {ROCS_DIR}")
print(f"    all_ligands_crystal_poses.sdf")
print(f"    rocs_prep_report.csv    ({len(df)} structures)")
print(f"    rocs_prep_issues.csv    ({len(issues_df)} flagged)")

# MAIN

if __name__ == "__main__":
    cleanup()
    prepare_rocs()

```

7.4.4 ROCS Run Workflow for RCSB CDK2 Dataset

```

#!/usr/bin/env python3
"""
compute_similarity_matrices.py

Pairwise NxN similarity matrices for CDK2 crystal ligand poses.
requires an available openeye license, loads openeye as a module.

Computes ROCS Shape, Color, EON Tanimoto, ECFP4 Morgan Fingerprint Tanimoto
Uses 3D crystal structure derived in preparation steps previously as SDF file, no conformers or
3D poses
generated in the script

input instructions:
python compute_similarity_matrices.py

Dependencies to load:
module load openeye/2022.2.1    (provides rocs and eon binaries, you may need to change this to
fit
your ROCS location)
pip install rdkit openpyxl pandas numpy
"""

import sys
import logging
import subprocess
import tempfile
import shutil
from pathlib import Path

import numpy as np
import pandas as pd
from openpyxl import Workbook
from openpyxl.styles import Font, PatternFill, Alignment, Border, Side
from openpyxl.utils import get_column_letter
from openpyxl.formatting.rule import ColorScaleRule

```

```

from rdkit import Chem
from rdkit.Chem import AllChem, DataStructs

SDF_FILE = Path("all_ligands_crystal_poses.sdf")
OUTPUT_DIR = Path("similarity_output")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

ROCS_CMD = "rocs"
EON_CMD = "eon"

MORGAN_RADIUS = 2 # sets number of iterations morgan FP runs
MORGAN_NBITS = 2048 #number of bits of output FP

# Checkpoint files - pairs saved here as computed so runs can resume
ROCS_CHECKPOINT = OUTPUT_DIR / "checkpoint_rocs.csv"
EON_CHECKPOINT = OUTPUT_DIR / "checkpoint_eon_et_pb.csv"
# NOTE THIS SCRIPT RUNS ET_pb alone, not ET_pb+ET_coul WHICH IS THE EON STANDARD - WILL HAVE TO
# CHANGE THIS
# IF YOU WANT ET_pb
MORGAN_CHECKPOINT = OUTPUT_DIR / "checkpoint_morgan.csv"

# LOGGING - necessary bc this is quite a long run with 100 ligands, you can remove this if small
dataset

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s %(levelname)s: %(message)s",
    handlers=[
        logging.FileHandler(str(OUTPUT_DIR / "similarity_run.log")),
        logging.StreamHandler(sys.stdout),
    ],
)
log = logging.getLogger("similarity")

# CHECKPOINT HELPERS - allows the script to skip re-running everything if the run terminates in
the middle
# also necessary bc of the dataset size, and allows you to keep track of what has all run

def load_checkpoint(path):
    """Load existing pair results from checkpoint CSV.
    Returns dict {(id_i, id_j): (val1, val2)} or {(id_i, id_j): val}
    """
    if not Path(path).exists():
        return {}
    try:
        df = pd.read_csv(path)
        result = {}
        for _, row in df.iterrows():
            key = (row["id_i"], row["id_j"])
            if "val2" in df.columns:
                result[key] = (float(row["val1"]), float(row["val2"]))
            else:
                result[key] = float(row["val1"])
        log.info(f"Checkpoint loaded: {len(result)} pairs from {Path(path).name}")
        return result
    except Exception as e:
        log.warning(f"Could not load checkpoint {path}: {e}")
        return {}

def save_checkpoint_row(path, id_i, id_j, val1, val2=None):
    """Append one pair result to checkpoint CSV."""
    row = f"{id_i},{id_j},{val1}"
    if val2 is not None:
        row += f",{val2}"
    with open(path, "a") as f:
        f.write(row + "\n")

def init_checkpoint(path, has_val2=False):
    """Write header if checkpoint file does not exist yet."""

```

```

    if not Path(path).exists():
        header = "id_i,id_j,vals,vals2\n" if has_val2 else "id_i,id_j,vals\n"
        with open(path, "w") as f:
            f.write(header)

# STEP 1: LOAD MOLECULES FROM SDF - SDF best input bc allows you to preserve 3D conformation +
connectivity

def load_molecules(sdf_path):
    """
    Load all molecules from SDF file. SDF file written as one file for all ligands
    in previous script - this block also splits into indiv. ligands
    Returns list of (pdb_id, rdkit_mol, sdf_block) tuples.
    sdf_block is the raw SDF text for that molecule (needed to write
    individual SDF files for ROCS/EON).
    """
    molecules = []
    suppl = Chem.SDMolSupplier(str(sdf_path), removeHs=False, sanitize=True)

    # Also read raw SDF blocks for writing individual files
    sdf_text = sdf_path.read_text(encoding="utf-8", errors="replace")
    # Split on $$$ delimiter
    raw_blocks = [b.strip() for b in sdf_text.split("$$$") if b.strip()]

    for i, mol in enumerate(suppl):
        if mol is None:
            continue
        pdb_id = (mol.GetProp("pdb_id") if mol.HasProp("pdb_id")
                  else mol.GetProp("_Name")).strip().upper()
        raw = raw_blocks[i] + "\n$$$$\n" if i < len(raw_blocks) else ""
        molecules.append((pdb_id, mol, raw))

    log.info(f"Loaded {len(molecules)} molecules from {sdf_path.name}")
    return molecules

def write_individual_sdfs(molecules, out_dir):
    """Write one SDF file per molecule for ROCS/EON input."""
    out_dir.mkdir(parents=True, exist_ok=True)
    sdf_paths = {}
    for pdb_id, mol, raw in molecules:
        p = out_dir / f"{pdb_id}.sdf"
        p.write_text(raw, encoding="utf-8")
        sdf_paths[pdb_id] = p
    return sdf_paths

# STEP 2: ROCS SHAPE + COLOR TANIMOTO

def parse_rocs_rpt(rpt_path): # for my version, parsing .rpt file was best way to get Tanimoto
values
    """
    Parse ROCS 3.5.x .rpt file for ShapeTanimoto and ColorTanimoto.
    Format (tab-separated):
        Name ShapeQuery Rank TanimotoCombo ShapeTanimoto ColorTanimoto ...
    ShapeTanimoto = col index 4, ColorTanimoto = col index 5
    """
    try:
        lines = Path(rpt_path).read_text().splitlines()
        header = None
        for line in lines:
            line = line.strip()
            if not line:
                continue
            parts = line.split("\t")
            if parts[0] == "Name":
                header = parts
                continue
            if header is not None and len(parts) >= 6:
                try:

```

```

        shape_idx = header.index("ShapeTanimoto")
        color_idx = header.index("ColorTanimoto")
        shape = float(parts[shape_idx])
        color = float(parts[color_idx])
        return shape, color
    except (ValueError, IndexError):
        # fallback: cols 4 and 5
        try:
            return float(parts[4]), float(parts[5])
        except (ValueError, IndexError):
            continue
except Exception as e:
    log.debug(f"RPT parse error {rpt_path}: {e}")
return float("nan"), float("nan")

def run_rocs_pair(query_sdf, db_sdf, workdir):
    """
    Run ROCS with query vs db (both single-conformer SDF = crystal pose).
    Returns (shape_tanimoto, color_tanimoto, aligned_oeb_path) or (nan, nan, None).
    Uses a unique per-pair subdirectory so ROCS output files never collide.
    The caller is responsible for cleaning up pair_dir (oeb_path.parent) after
    EON has consumed the aligned OEB.
    Generates OEB file which is necessary to retain as input for EON run
    """
    import uuid
    pair_dir = Path(workdir) / uuid.uuid4().hex[:8]
    pair_dir.mkdir(parents=True, exist_ok=True)
    rpt_prefix = str(pair_dir / "rocs_out")

    cmd = [
        ROCS_CMD,
        "-query",      str(Path(query_sdf).resolve()),
        "-dbase",      str(Path(db_sdf).resolve()),
        "-prefix",     rpt_prefix,
        "-besthits",   "1",
        "-rankby",     "TanimotoCombo",
        "-oformat",    "oeb.gz",
    ]

    try:
        proc = subprocess.run(
            cmd,
            stdout=subprocess.PIPE, stderr=subprocess.PIPE,
            timeout=120
        )
        if proc.returncode != 0:
            log.debug(f"ROCS error: {proc.stderr.decode(errors='ignore')[:200]}")
            shutil.rmtree(pair_dir, ignore_errors=True)
            return float("nan"), float("nan"), None
    except Exception as e:
        log.debug(f"ROCS exception: {e}")
        shutil.rmtree(pair_dir, ignore_errors=True)
        return float("nan"), float("nan"), None

    # RPT
    rpt_found = pair_dir / "rocs_out_1.rpt"
    if not rpt_found.exists():
        rpts = list(pair_dir.glob("*.rpt"))
        rpt_found = rpts[0] if rpts else None
    shape, color = (parse_rocs_rpt(str(rpt_found))
                    if rpt_found else (float("nan"), float("nan")))

    # OEB - ROCS writes {prefix}_hits_1.oeb.gz
    oeb_path = pair_dir / "rocs_out_hits_1.oeb.gz"
    if not oeb_path.exists():
        hits = list(pair_dir.glob("*.oeb.gz")) or list(pair_dir.glob("*.oeb"))
        oeb_path = hits[0] if hits else None

    if oeb_path is None:
        log.debug(f"No OEB found in {pair_dir}. Files: {list(pair_dir.iterdir())}")

```

```

        shutil.rmtree(pair_dir, ignore_errors=True)
        return shape, color, None

# Return oeb_path - caller cleans up pair_dir after EON is done
return shape, color, oeb_path

def compute_rocs_matrices(pdb_ids, sdf_paths, aligned_oeb_dir):
    """
    Compute pairwise ROCS Shape and Color Tanimoto matrices.
    Aligned OEB files are saved to aligned_oeb_dir so EON can reuse them
    without re-running ROCS. Already-present OEBs are skipped.
    """
    n = len(pdb_ids)
    shape_mat = np.zeros((n, n)); np.fill_diagonal(shape_mat, 1.0)
    color_mat = np.zeros((n, n)); np.fill_diagonal(color_mat, 1.0)

    aligned_oeb_dir.mkdir(parents=True, exist_ok=True)

    # Load existing checkpoint
    init_checkpoint(ROCS_CHECKPOINT, has_val2=True)
    done_pairs = load_checkpoint(ROCS_CHECKPOINT)
    for (id_i, id_j), (st, ct) in done_pairs.items():
        if id_i in pdb_ids and id_j in pdb_ids:
            i = pdb_ids.index(id_i); j = pdb_ids.index(id_j)
            shape_mat[i, j] = shape_mat[j, i] = st
            color_mat[i, j] = color_mat[j, i] = ct

    n_pairs = n * (n - 1) // 2
    n_skip = len(done_pairs)
    log.info(f"Computing ROCS: {n_pairs} pairs ({n_skip} already done)...")

    # Use OUTPUT_DIR subdir - keeps everything on the same filesystem as the project,
    # avoiding node-local /tmp issues on the cluster.
    # might be different on your machine, mine gets overloaded with temp files
    workdir = OUTPUT_DIR / "rocs_scratch"
    workdir.mkdir(parents=True, exist_ok=True)
    done = n_skip

    try:
        for i in range(n):
            for j in range(i + 1, n):
                id_i, id_j = pdb_ids[i], pdb_ids[j]
                key = (id_i, id_j)
                if key in done_pairs:
                    continue

                st, ct, oeb_path = run_rocs_pair(
                    sdf_paths[id_i], sdf_paths[id_j], workdir
                )

                # Move aligned OEB to persistent directory for EON reuse
                if oeb_path is not None:
                    dest = aligned_oeb_dir / f"{id_i}_VS_{id_j}_hits_1.oeb.gz"
                    shutil.move(str(oeb_path), str(dest))
                    shutil.rmtree(oeb_path.parent, ignore_errors=True)

                st = max(0.0, min(1.0, st)) if not np.isnan(st) else 0.0
                ct = max(0.0, min(1.0, ct)) if not np.isnan(ct) else 0.0
                shape_mat[i, j] = shape_mat[j, i] = st
                color_mat[i, j] = color_mat[j, i] = ct
                save_checkpoint_row(ROCS_CHECKPOINT, id_i, id_j, st, ct)
                done += 1
                if done % 500 == 0:
                    log.info(f" ROCS {done}/{n_pairs} ({100*done/n_pairs:.1f}%)")
    finally:
        shutil.rmtree(str(workdir), ignore_errors=True)

    log.info("ROCS complete")
    return shape_mat, color_mat

```

```

# STEP 2b: Generate per-molecule self-ROCS OEB files (EON query input)

def generate_eon_query_oeb(pdb_ids, sdf_paths, oeb_dir):
    """
    Run ROCS in self-overlay mode for each molecule to produce a charge-bearing
    OEB that EON requires as its -query input. Mirrors the approach used in
    rocs_esp_morgan.py. re-runs skip already-run files. Necessary to run this
    so that database OEB (actual file of interest for EON) has a query comparison.
    For some reason EON requires a comparison OEB to calculate. However the OEB comparison
    file of interest from the Shape/Color run is the true file of interest
    Returns dict {pdb_id: Path to _hits_1.oeb.gz}
    """
    oeb_dir.mkdir(parents=True, exist_ok=True)
    eon_inputs = {}
    for pdb_id in pdb_ids:
        sdf = sdf_paths[pdb_id]
        prefix = str(oeb_dir / pdb_id)
        expected = oeb_dir / f"{pdb_id}_hits_1.oeb.gz"
        if expected.exists():
            eon_inputs[pdb_id] = expected
            continue
        cmd = [
            ROCS_CMD,
            "-query", str(sdf),
            "-dbase", str(sdf),
            "-prefix", prefix,
            "-oformat", "oeb.gz",
            "-besthits", "1",
        ]
        result = subprocess.run(cmd, stdout=subprocess.DEVNULL,
                                stderr=subprocess.DEVNULL)
        if result.returncode == 0 and expected.exists():
            eon_inputs[pdb_id] = expected
        else:
            log.warning(f"Self-ROCS OEB generation failed for {pdb_id}")
    log.info(f"Self-ROCS OEBs ready: {len(eon_inputs)}/{len(pdb_ids)}")
    return eon_inputs

# STEP 3: EON ESP TANIMOTO

def _parse_eon_rpt(rpt_path):
    """
    Parse EON RPT file (tab-separated).

    Column layout for output file:
    Name EON_Query EON_Rank EON_ET_pb EON_ET_coul EON_ET_combo EON_ShapeTanimoto

    EON writes the query molecule itself as rank 1 (self-score ET_combo ~ 2.0).
    skips any row where Name == EON_Query and return the first non-self row in the output file.

    Returns (et_combo, et_pb, et_coul) or None.
    """
    try:
        lines = Path(rpt_path).read_text().splitlines()
        header = next(
            (l.split("\t") for l in lines if "EON_ET_combo" in l),
            None,
        )
        if header is None:
            header = next(
                (l.split() for l in lines if "eon_et_combo" in l.lower()),
                None,
            )
        if header is None:
            log.debug(f"EON RPT has no recognised header: {rpt_path}")
            return None

        col = {h.strip().lower(): i for i, h in enumerate(header)}
        idx_combo = col.get("eon_et_combo", 5)
    
```

```

idx_pb    = col.get("eon_et_pb",    3)
idx_coul  = col.get("eon_et_coul",  4)
idx_name  = col.get("name",         0)
idx_query = col.get("eon_query",    1)

for line in lines[1:]:
    parts = line.split("\t") if "\t" in line else line.split()
    if not parts or len(parts) < 4:
        continue
    # Skip self-score row (Name == EON_Query, ET_combo ~ 2.0)
    name = parts[idx_name].strip()
    query = parts[idx_query].strip() if idx_query < len(parts) else ""
    if name == query:
        continue
    try:
        return (float(parts[idx_combo]),
                float(parts[idx_pb]),
                float(parts[idx_coul]))
    except (IndexError, ValueError):
        continue
except Exception as e:
    log.debug(f"EON rpt parse error {rpt_path}: {e}")
return None

def run_eon_pair(query_oeb, aligned_oeb, eon_workdir):
    """
    Run EON for ESP similarity.
    query_oeb - self-ROCS OEB for the query molecule (carries charge data)
    aligned_oeb - pairwise ROCS hitlist OEB (aligned db pose)
    Returns (et_combo, et_pb) or (nan, nan).

    NOTE: reports ET_pb alone as the primary electrostatic
    similarity metric, EON standard is to compute ET_coul + ET_pb.

    EON writes results to {prefix}_1.rpt or {prefix}.rpt depending on version.
    Each call uses a unique subdirectory to avoid file collisions.
    """
    import uuid
    pair_dir = Path(eon_workdir) / uuid.uuid4().hex[:8]
    pair_dir.mkdir(parents=True, exist_ok=True)
    prefix = str(pair_dir / "eon_out")

    cmd = [
        EON_CMD,
        "-query",    str(Path(query_oeb).resolve()),
        "-dbase",    str(Path(aligned_oeb).resolve()),
        "-prefix",   prefix,
        "-besthits", "2",
    ]

    et_combo = et_pb_val = float("nan")
    try:
        proc = subprocess.run(
            cmd,
            stdout=subprocess.PIPE, stderr=subprocess.PIPE,
            timeout=120
        )
        if proc.returncode != 0:
            log.debug("EON error: " + proc.stderr.decode(errors="ignore")[:200])
        else:
            # EON may write {prefix}_1.rpt or {prefix}.rpt depending on version
            rpt = next(
                (Path(f"{prefix}{s}") for s in ("_1.rpt", ".rpt"))
                if Path(f"{prefix}{s}").exists(),
                None,
            )
            if rpt is None:
                all_rpts = sorted(pair_dir.glob("*.rpt"))
                rpt = all_rpts[0] if all_rpts else None
            if rpt is None:

```

```

        log.debug(f"EON produced no RPT. Files: {list(pair_dir.iterdir())}")
    else:
        scores = _parse_eon_rpt(rpt)
        if scores is not None:
            et_combo_val, et_pb, et_coul = scores
            et_combo = et_combo_val
            et_pb_val = et_pb
            # et_coul is parsed (for potential future reference / a
            # separate sheet) but is no longer folded into the
            # reported electrostatic score.
except Exception as e:
    log.debug(f"EON exception: {e}")
finally:
    shutil.rmtree(pair_dir, ignore_errors=True)

return et_combo, et_pb_val

def compute_eon_matrix(pdb_ids, sdf_paths, eon_query_oeb, aligned_oeb_dir):
    """
    Compute pairwise EON matrices using pre-saved aligned OEBs.
    Returns two matrices:
    - et_pb_mat : EON_ET_pb (Poisson-Boltzmann electrostatic Tanimoto,
        range -1 to 1) - the primary electrostatic
        similarity metric, reported alone.
    - combo_mat : EON_ET_combo (electrostatic + shape, range -2 to 2)
    Both are stored in the checkpoint (val1=et_pb, val2=et_combo).
    """
    n = len(pdb_ids)
    et_pb_mat = np.full((n, n), np.nan); np.fill_diagonal(et_pb_mat, 2.0)
    combo_mat = np.full((n, n), np.nan); np.fill_diagonal(combo_mat, 2.0)

    n_pairs = n * (n - 1) // 2
    init_checkpoint(EON_CHECKPOINT, has_val2=True)
    done_pairs = load_checkpoint(EON_CHECKPOINT)
    for (id_i, id_j), (et_pb, combo) in done_pairs.items():
        if id_i in pdb_ids and id_j in pdb_ids:
            i = pdb_ids.index(id_i); j = pdb_ids.index(id_j)
            et_pb_mat[i, j] = et_pb_mat[j, i] = et_pb
            combo_mat[i, j] = combo_mat[j, i] = combo

    n_skip = len(done_pairs)
    log.info(f"Computing EON (ET_pb only): {n_pairs} pairs ({n_skip} already done)...")

    eon_workdir = OUTPUT_DIR / "eon_scratch"
    eon_workdir.mkdir(parents=True, exist_ok=True)
    done = n_skip

    try:
        for i in range(n):
            for j in range(i + 1, n):
                id_i, id_j = pdb_ids[i], pdb_ids[j]
                key = (id_i, id_j)
                if key in done_pairs:
                    continue

                if id_i not in eon_query_oeb or id_j not in eon_query_oeb:
                    log.debug(f"Missing self-OEB for {id_i} or {id_j}, skipping")
                    et_pb, combo = float("nan"), float("nan")
                else:
                    aligned_oeb = aligned_oeb_dir / f"{id_i}_VS_{id_j}_hits_1.oeb.gz"
                    if not aligned_oeb.exists():
                        log.debug(f"No aligned OEB for {id_i}/{id_j}, skipping")
                        et_pb, combo = float("nan"), float("nan")
                    else:
                        combo, et_pb = run_eon_pair(
                            eon_query_oeb[id_i], aligned_oeb, eon_workdir
                        )

                # No clamping - both metrics range -2 to 2; use 0.0 for failures
                et_pb = et_pb if not np.isnan(et_pb) else 0.0
                combo = combo if not np.isnan(combo) else 0.0

```

```

        et_pb_mat[i, j] = et_pb_mat[j, i] = et_pb
        combo_mat[i, j] = combo_mat[j, i] = combo
        save_checkpoint_row(EON_CHECKPOINT, id_i, id_j, et_pb, combo)
        done += 1
        if done % 500 == 0:
            log.info(f"  EON {done}/{n_pairs} ({100*done/n_pairs:.1f}%)")
    finally:
        shutil.rmtree(str(eon_workdir), ignore_errors=True)

    log.info("EON complete")
    return et_pb_mat, combo_mat

# STEP 4: MORGAN ECFP4 TANIMOTO
def compute_morgan_matrix(pdb_ids, molecules):
    #Pairwise Morgan ECFP4 Tanimoto, uses RDKit to calculate
    mol_map = {pid: mol for pid, mol, _ in molecules}
    n = len(pdb_ids)
    mat = np.zeros((n, n)); np.fill_diagonal(mat, 1.0)

    fps = []
    for pid in pdb_ids:
        mol = mol_map[pid]
        fp = AllChem.GetMorganFingerprintAsBitVect(
            mol, MORGAN_RADIUS, MORGAN_NBITS)
        fps.append(fp)

    n_pairs = n * (n - 1) // 2
    log.info(f"Computing Morgan ECFP4: {n_pairs} pairs...")

    for i in range(n):
        for j in range(i + 1, n):
            sim = DataStructs.TanimotoSimilarity(fps[i], fps[j])
            mat[i, j] = mat[j, i] = sim

    log.info("Morgan complete")
    return mat

# STEP 5: WRITE EXCEL WORKBOOK
def write_matrix_sheet(wb, sheet_name, matrix, pdb_ids, description):
    ws = wb.create_sheet(title=sheet_name)
    n = len(pdb_ids)
    hf = Font(name="Arial", bold=True, color="FFFFFF", size=9)
    hfill = PatternFill("solid", fgColor="1F4E79")
    ha = Alignment(horizontal="center", vertical="center",
                    wrap_text=True, text_rotation=90)
    ctr = Alignment(horizontal="center", vertical="center")
    dgrey = PatternFill("solid", fgColor="EFEFEF")
    df = Font(name="Arial", size=8)
    thin = Side(style="thin", color="D9D9D9")
    bdr = Border(left=thin, right=thin, top=thin, bottom=thin)

    # Description row
    ws["A1"] = description
    ws["A1"].font = Font(name="Arial", italic=True, size=9, color="595959")
    ws.merge_cells(start_row=1, start_column=1, end_row=1, end_column=n + 1)

    # Column headers
    c = ws.cell(row=2, column=1, value="PDB ID")
    c.font = hf; c.fill = hfill
    c.alignment = Alignment(horizontal="center", vertical="center")

    for j, pid in enumerate(pdb_ids):
        c = ws.cell(row=2, column=j + 2, value=pid)
        c.font = hf; c.fill = hfill; c.alignment = ha

    # Data
    for i, pid in enumerate(pdb_ids):

```

```

        rh = ws.cell(row=i + 3, column=1, value=pid)
        rh.font = hf; rh.fill = hfill
        rh.alignment = Alignment(horizontal="left", vertical="center")
        for j in range(n):
            c = ws.cell(row=i + 3, column=j + 2,
                        value=round(float(matrix[i, j]), 4))
            c.font = df; c.alignment = ctr; c.border = bdr
            c.number_format = "0.0000"
            if i == j:
                c.fill = dgrey

        ws.column_dimensions["A"].width = 9
        for j in range(n):
            ws.column_dimensions[get_column_letter(j + 2)].width = 6.5
        ws.row_dimensions[2].height = 55
        for i in range(n):
            ws.row_dimensions[i + 3].height = 13

        ws.conditional_formatting.add(
            f"B3:{get_column_letter(n + 1)}:{n + 2}",
            ColorScaleRule(
                start_type="num", start_value=0, start_color="FFFFFF",
                mid_type="num", mid_value=0.5, mid_color="BDD7EE",
                end_type="num", end_value=1.0, end_color="1F4E79",
            ),
        )
        ws.freeze_panes = "B3"
        log.info(f" Sheet '{sheet_name}' written ({n}x{n})")

def write_workbook(shape_mat, color_mat, eon_et_pb_mat, eon_combo_mat, morgan_mat, pdb_ids):
    wb = Workbook()
    wb.remove(wb.active)
    sheets = [
        ("ROCS_Shape", shape_mat,
         "ROCS Shape Tanimoto - 3D shape similarity, crystal poses, no conformer search"),
        ("ROCS_Color", color_mat,
         "ROCS Color Tanimoto - pharmacophore feature overlay, crystal poses"),
        ("EON_ET_pb", eon_et_pb_mat,
         "EON Electrostatic Tanimoto (ET_pb, Poisson-Boltzmann) - primary "
         "electrostatic similarity metric, range -2 to 2. (Previously this "
         "sheet reported ET_pb + ET_coul; see methods notes - ET_pb alone "
         "correlates with that combined score at r > 0.99 in this dataset.)"),
        ("EON_ET_combo", eon_combo_mat,
         "EON ET_combo - electrostatic (ET_pb) + shape combined Tanimoto, range -2 to 2"),
        ("Morgan_ECFP4", morgan_mat,
         f"Morgan Fingerprint Tanimoto (radius={MORGAN_RADIUS}, {MORGAN_NBITS} bits)"),
    ]
    for name, mat, desc in sheets:
        write_matrix_sheet(wb, name, mat, pdb_ids, desc)

    out = OUTPUT_DIR / "similarity_matrices.xlsx"
    wb.save(str(out))
    log.info(f"Workbook saved -> {out}")

def write_csvs(shape_mat, color_mat, eon_et_pb_mat, eon_combo_mat, morgan_mat, pdb_ids):
    for name, mat in [("rocs_shape", shape_mat),
                     ("rocs_color", color_mat),
                     ("eon_et_pb", eon_et_pb_mat),
                     ("eon_et_combo", eon_combo_mat),
                     ("morgan_ecfp4", morgan_mat)]:
        df = pd.DataFrame(mat.round(4), index=pdb_ids, columns=pdb_ids)
        df.index.name = "pdb_id"
        df.to_csv(OUTPUT_DIR / f"similarity_{name}.csv")
        log.info(f" CSV -> similarity_{name}.csv")

# MAIN

def main():

```

```

log.info("CDK2 CRYSTAL POSE SIMILARITY MATRICES")

# Check binaries are available
for binary, name in [(ROCS_CMD, "ROCS"), (EON_CMD, "EON")]:
    result = subprocess.run(["which", binary],
                             capture_output=True, text=True)
    if result.returncode != 0:
        log.error(f"{name} binary '{binary}' not found in PATH.")
        log.error("Make sure you have run: module load openeye/2022.2.1")
        sys.exit(1)
    log.info(f"{name} found: {result.stdout.strip()}")

# Load molecules
log.info(f"\nLoading molecules from {SDF_FILE}")
molecules = load_molecules(SDF_FILE)
pdb_ids = [pid for pid, _, _ in molecules]
n = len(pdb_ids)
log.info(f"{n} molecules, {n*(n-1)//2} pairs per metric")

# Write individual SDF files
sdf_dir = OUTPUT_DIR / "individual_sdfs"
sdf_paths = write_individual_sdfs(molecules, sdf_dir)
log.info(f"Individual SDFs written to {sdf_dir}")

# Placeholders so write functions always have all four matrices
nan_mat = np.full((n, n), float("nan"))
np.fill_diagonal(nan_mat, 1.0)
shape_mat = color_mat = eon_et_pb_mat = eon_combo_mat = morgan_mat = nan_mat.copy()

def save_intermediate():
    """Write Excel + CSVs with whatever matrices are complete so far.
    Sheets with nan_mat placeholders are included as empty
    so the file is always a valid workbook.
    """
    log.info(" Writing intermediate Excel + CSVs")
    write_workbook(shape_mat, color_mat, eon_et_pb_mat, eon_combo_mat, morgan_mat, pdb_ids)
    write_csvs(shape_mat, color_mat, eon_et_pb_mat, eon_combo_mat, morgan_mat, pdb_ids)

# ROCS
log.info("\nRocs Shape + Color")
aligned_oeb_dir = OUTPUT_DIR / "aligned_oeb"
shape_mat, color_mat = compute_rocs_matrices(pdb_ids, sdf_paths, aligned_oeb_dir)
np.save(str(OUTPUT_DIR / "sim_shape.npy"), shape_mat)
np.save(str(OUTPUT_DIR / "sim_color.npy"), color_mat)
log.info("ROCS matrices saved - writing intermediate Excel")
save_intermediate()

# Self-ROCS OEBs (EON query input)
log.info("\nGenerating self-ROCS OEB files for EON queries")
eon_oeb_dir = OUTPUT_DIR / "eon_query_oeb"
eon_query_oeb = generate_eon_query_oeb(pdb_ids, sdf_paths, eon_oeb_dir)

# EON
log.info("\nEON ET_pb (Poisson-Boltzmann electrostatic Tanimoto)")
eon_et_pb_mat, eon_combo_mat = compute_eon_matrix(pdb_ids, sdf_paths, eon_query_oeb,
aligned_oeb_dir)
np.save(str(OUTPUT_DIR / "sim_eon_et_pb.npy"), eon_et_pb_mat)
np.save(str(OUTPUT_DIR / "sim_eon_combo.npy"), eon_combo_mat)
log.info("EON matrices saved - writing intermediate Excel")
save_intermediate()

# Morgan
log.info("\nMorgan ECFP4")
morgan_mat = compute_morgan_matrix(pdb_ids, molecules)
np.save(str(OUTPUT_DIR / "sim_morgan.npy"), morgan_mat)
log.info("All matrices complete - writing final Excel")
save_intermediate()

# Summary
log.info("Summary (upper triangle, diagonal excluded):")
for name, mat in [("ROCS Shape", shape_mat),

```

```

        ("ROCS Color",    color_mat),
        ("EON ET_pb",    eon_et_pb_mat),
        ("EON ET_combo",  eon_combo_mat),
        ("Morgan ECFP4",  morgan_mat)]:
    upper = mat[np.triu_indices(n, k=1)]
    valid = upper[~np.isnan(upper)]
    log.info(f" {name:<15} mean={valid.mean():.3f} "
             f"median={np.median(valid):.3f} "
             f"max={valid.max():.3f} min={valid.min():.3f} "
             f"({len(valid)}/{len(upper)} valid pairs)")
    log.info(f"All outputs -> {OUTPUT_DIR}")

    # Clean up individual SDFs
    shutil.rmtree(sdf_dir, ignore_errors=True)
    log.info("Individual SDF files cleaned up")

if __name__ == "__main__":
    main()

```

7.4.5 Vina and GNINA Docking Workflow for RCSB CDK2 Dataset

1) Scoring the native ligand-receptor complex via Vina

```

#!/bin/bash
#
# 01_vina_score_only.sh - first bash script in the workflow
# requires that you have Autodock Vina installed or present as a module to be called
#
# Score each crystal-pose ligand in its receptor using AutoDock Vina 1.2.7
# (--score_only, no docking search bc want to preserve ligand crystal pose for this)
#
# Each job writes its own .result file
# Run 06_collect_all.py after all jobs finish.
#
# BOX_MODE - Vina requires a valid grid box even in --score_only mode.
# The box does not affect the score but dimensions must be > 0. I created this so
# you can run this with a global .json file or .json files for each structure,
# or just input global box coordinates
#
# BOX SETTINGS:
# "auto" - uses ???_vina_box.json if present, global box as fallback
# "per_complex" - requires ???_vina_box.json for every complex
# "global" - uses the GLOBAL_* variables below for every complex
#
# Output per complex: results/01_vina_scoreonly/???_vina_scoreonly.result
# (single line containing the score in kcal/mol)
#

set -euo pipefail

BOX_MODE="global" # auto, per_complex, or global can be set here

# Global box - used when BOX_MODE=global or no JSON found in auto mode.
# Set these to your binding site coordinates.
GLOBAL_CENTER_X=-1.788
GLOBAL_CENTER_Y=30.962
GLOBAL_CENTER_Z=8.996
GLOBAL_SIZE_X=30.0
GLOBAL_SIZE_Y=30.0
GLOBAL_SIZE_Z=30.0

# SLURM
SLURM_MEM="2GB"
SLURM_TIME="0-0:05:00"
SLURM_CPUS=4
SLURM_PARTITION="XXX"

# Vina

```

```

CPU=4

# Setup

JOBS_DIR="jobs/01_vina_scoreonly"
RESULTS_DIR="results/01_vina_scoreonly"
mkdir -p "$JOBS_DIR" "$RESULTS_DIR"

submitted=0
skipped=0
n_missing=0

# Main loop with troubleshooting outputs

for ligand in ???_ligand.pdbqt; do
    [ -e "$ligand" ] || { echo " No ligand files found (???_ligand_clean.pdbqt)."; exit 1; }

    pdbid="${ligand:0:4}"
    protein="${pdbid}_receptor.pdbqt"

    for f in "$protein" "$ligand"; do
        [ -f "$f" ] || { echo " Skipping $pdbid - missing $f"; n_missing=$((n_missing+1));
    continue 2; }
    done

    RESULT="${RESULTS_DIR}/${pdbid}_vina_scoreonly.result"
    if [ -f "$RESULT" ]; then
        echo " $pdbid already scored, skipping."
        skipped=$((skipped+1))
        continue
    fi

    # Resolve box coordinates - json stuff is skipped if "box mode" set to global
    boxjson="${pdbid}_vina_box.json"
    box_source=""

    case "$BOX_MODE" in
        per_complex)
            if [ ! -f "$boxjson" ]; then
                echo " Skipping $pdbid - BOX_MODE=per_complex but $boxjson not found"
                n_missing=$((n_missing+1))
                continue
            fi
            CENTER_X=$(jq '.center[0]' "$boxjson")
            CENTER_Y=$(jq '.center[1]' "$boxjson")
            CENTER_Z=$(jq '.center[2]' "$boxjson")
            SIZE_X=$(jq '.size[0]' "$boxjson")
            SIZE_Y=$(jq '.size[1]' "$boxjson")
            SIZE_Z=$(jq '.size[2]' "$boxjson")
            box_source="per-complex JSON"
            ;;
        global)
            CENTER_X=$GLOBAL_CENTER_X; CENTER_Y=$GLOBAL_CENTER_Y; CENTER_Z=$GLOBAL_CENTER_Z
            SIZE_X=$GLOBAL_SIZE_X; SIZE_Y=$GLOBAL_SIZE_Y; SIZE_Z=$GLOBAL_SIZE_Z
            box_source="global box"
            ;;
        auto)
            if [ -f "$boxjson" ]; then
                CENTER_X=$(jq '.center[0]' "$boxjson")
                CENTER_Y=$(jq '.center[1]' "$boxjson")
                CENTER_Z=$(jq '.center[2]' "$boxjson")
                SIZE_X=$(jq '.size[0]' "$boxjson")
                SIZE_Y=$(jq '.size[1]' "$boxjson")
                SIZE_Z=$(jq '.size[2]' "$boxjson")
                box_source="per-complex JSON"
            else
                CENTER_X=$GLOBAL_CENTER_X; CENTER_Y=$GLOBAL_CENTER_Y; CENTER_Z=$GLOBAL_CENTER_Z
                SIZE_X=$GLOBAL_SIZE_X; SIZE_Y=$GLOBAL_SIZE_Y; SIZE_Z=$GLOBAL_SIZE_Z
                box_source="global box (no JSON found)"
            fi
            ;;
    esac
done

```

```

*)
    echo " Unknown BOX_MODE: $BOX_MODE. Use auto, per_complex, or global."; exit 1 ;;
esac

LOG="${RESULTS_DIR}/${pdbid}_vina_scoreonly.log"
JOB="${JOBS_DIR}/${pdbid}_vina_scoreonly.job"

cat > "$JOB" <<EOF
#!/bin/bash
#SBATCH --job-name=${pdbid}_vscore
#SBATCH --mem=${SLURM_MEM}
#SBATCH --time=${SLURM_TIME}
#SBATCH --cpus-per-task=${SLURM_CPUS}
#SBATCH --partition=${SLURM_PARTITION}
#SBATCH --output=${RESULTS_DIR}/${pdbid}_vina_scoreonly.out
#SBATCH --error=${RESULTS_DIR}/${pdbid}_vina_scoreonly.err

set -euo pipefail

module purge
module load arch/avx512
module load gcc/13.2.0
module load autodock-vina/1.2.7
module load jq/1.8.1

cd \${SLURM_SUBMIT_DIR}

echo " Vina score-only: ${pdbid} [box: ${box_source}]"

# Box coordinates are required by Vina even in --score_only mode.
# They do not affect the score - Vina just needs non-zero dimensions.
vina \\\
    --receptor    ${protein} \\\
    --ligand      ${ligand} \\\
    --score_only \\\
    --center_x    ${CENTER_X} \\\
    --center_y    ${CENTER_Y} \\\
    --center_z    ${CENTER_Z} \\\
    --size_x      ${SIZE_X} \\\
    --size_y      ${SIZE_Y} \\\
    --size_z      ${SIZE_Z} \\\
    --cpu         ${CPU} \\\
    > ${LOG} 2>&1

# Vina --score_only output format:
#   Estimated Free Energy of Binding    : -X.XXX (kcal/mol) [(1)+(2)+(3)-(4)]
#
# NOTE: "Affinity:" does NOT appear in Vina score_only output
vina_score=$(grep -ml "Estimated Free Energy of Binding" "${LOG}" \\\
    | awk -F: '{print $2}' | awk '{print $1}')

if [ -z "\${vina_score}" ]; then
    echo " Could not extract score for ${pdbid}."
    echo "   Expected line: 'Estimated Free Energy of Binding    : -X.XXX'"
    echo "   Check log: ${LOG}"
    exit 1
fi

echo "\${vina_score}" > "${RESULT}"
echo " ${pdbid}: \${vina_score} kcal/mol"
EOF

sbatch "$JOB"
echo " Submitted $pdbid [${box_source}]"
submitted=$((submitted+1))
done

echo ""
echo " Submitted $submitted Vina score-only jobs | Skipped $skipped | Missing: $n_missing"
echo " Results output to $RESULTS_DIR"
echo " Run 06_collect_all.py after all jobs complete."

```

2) Re-scoring the native ligand-receptor complex via GNINA

```
#!/bin/bash
#
# 02_gnina_score_only.sh
#
# Re-score each crystal-pose ligand in its receptor using GNINA (score_only),
# preserves native ligand-receptor coordinates
#
# NOTE: Like Vina, GNINA requires a valid grid box even in --score_only mode.
#       Box coordinates are passed but do not affect the CNN score (as long as box
#       coords set properly)
#
# BOX_MODE - same logic as 01:
#   "per_complex" - reads ???_vina_box.json per complex
#   "global"      - uses the GLOBAL_* variables below for every complex
#   "auto"        - per-complex JSON if present, global box as fallback
#
# Output per complex: results/02_gnina_scoreonly/???_gnina_scoreonly.result
# Format (space-separated): CNNscore CNNaffinity vinascore
#

set -euo pipefail

BOX_MODE="global" #SET BOX MODE HERE

GLOBAL_CENTER_X=-1.788
GLOBAL_CENTER_Y=30.962
GLOBAL_CENTER_Z=8.996
GLOBAL_SIZE_X=30.0
GLOBAL_SIZE_Y=30.0
GLOBAL_SIZE_Z=30.0

# SLURM
SLURM_MEM="4GB"
SLURM_TIME="0-0:10:00"
SLURM_CPUS=4
SLURM_PARTITION="XXXX"

# GNINA
GNINA_MODULE="gnina" # adjust this to your module name or GNINA path

# Setup

JOBS_DIR="jobs/02_gnina_scoreonly"
RESULTS_DIR="results/02_gnina_scoreonly"
mkdir -p "$JOBS_DIR" "$RESULTS_DIR"

submitted=0
skipped=0
n_missing=0

# Main loop
for ligand in ???_ligand.pdbqt; do
    [ -e "$ligand" ] || { echo " No ligand files found."; exit 1; }

    pdbid="${ligand:0:4}" #assumes file format is {4-digit PDBID}_ligand.pdbqt
    protein="${pdbid}_receptor.pdbqt"

    for f in "$protein" "$ligand"; do
        [ -f "$f" ] || { echo " Skipping $pdbid - missing $f"; n_missing=$((n_missing+1));
        continue 2; }
    done

    RESULT="${RESULTS_DIR}/${pdbid}_gnina_scoreonly.result"
    if [ -f "$RESULT" ]; then
        echo " $pdbid already scored, skipping."
        skipped=$((skipped+1))
        continue
    fi
done
```

```

fi

# Resolve box coordinates - if there is no .json files json stuff is skipped
boxjson="{pdbname}_vina_box.json"
box_source=""

case "$BOX_MODE" in
  per_complex)
    if [ ! -f "$boxjson" ]; then
      echo " Skipping $pdbname - BOX_MODE=per_complex but $boxjson not found"
      n_missing=$((n_missing+1))
      continue
    fi
    CENTER_X=$(jq '.center[0]' "$boxjson")
    CENTER_Y=$(jq '.center[1]' "$boxjson")
    CENTER_Z=$(jq '.center[2]' "$boxjson")
    SIZE_X=$(jq '.size[0]' "$boxjson")
    SIZE_Y=$(jq '.size[1]' "$boxjson")
    SIZE_Z=$(jq '.size[2]' "$boxjson")
    box_source="per-complex JSON"
    ;;
  global)
    CENTER_X=$GLOBAL_CENTER_X; CENTER_Y=$GLOBAL_CENTER_Y; CENTER_Z=$GLOBAL_CENTER_Z
    SIZE_X=$GLOBAL_SIZE_X; SIZE_Y=$GLOBAL_SIZE_Y; SIZE_Z=$GLOBAL_SIZE_Z
    box_source="global box"
    ;;
  auto)
    if [ -f "$boxjson" ]; then
      CENTER_X=$(jq '.center[0]' "$boxjson")
      CENTER_Y=$(jq '.center[1]' "$boxjson")
      CENTER_Z=$(jq '.center[2]' "$boxjson")
      SIZE_X=$(jq '.size[0]' "$boxjson")
      SIZE_Y=$(jq '.size[1]' "$boxjson")
      SIZE_Z=$(jq '.size[2]' "$boxjson")
      box_source="per-complex JSON"
    else
      CENTER_X=$GLOBAL_CENTER_X; CENTER_Y=$GLOBAL_CENTER_Y; CENTER_Z=$GLOBAL_CENTER_Z
      SIZE_X=$GLOBAL_SIZE_X; SIZE_Y=$GLOBAL_SIZE_Y; SIZE_Z=$GLOBAL_SIZE_Z
      box_source="global box (no JSON found)"
    fi
    ;;
  *)
    echo " Unknown BOX_MODE: $BOX_MODE"; exit 1 ;;
esac

LOG="{RESULTS_DIR}/${pdbname}_gnina_scoreonly.log"
JOB="{JOBS_DIR}/${pdbname}_gnina_scoreonly.job"

cat > "$JOB" <<EOF
#!/bin/bash
#SBATCH --job-name=${pdbname}_gscore
#SBATCH --mem=${SLURM_MEM}
#SBATCH --time=${SLURM_TIME}
#SBATCH --cpus-per-task=${SLURM_CPUS}
#SBATCH --partition=${SLURM_PARTITION}
#SBATCH --output=${RESULTS_DIR}/${pdbname}_gnina_scoreonly.out
#SBATCH --error=${RESULTS_DIR}/${pdbname}_gnina_scoreonly.err

set -euo pipefail

module purge
module load arch/avx512
module load gcc/13.2.0
module load jq/1.8.1
module load ${GNINA_MODULE}

cd \${SLURM_SUBMIT_DIR}

echo " GNINA score-only: ${pdbname} [box: ${box_source}]"

# Box coordinates required even in score_only mode.

```

```

gnina \
  --receptor    ${protein} \
  --ligand      ${ligand} \
  --score_only \
  --center_x    ${CENTER_X} \
  --center_y    ${CENTER_Y} \
  --center_z    ${CENTER_Z} \
  --size_x      ${SIZE_X} \
  --size_y      ${SIZE_Y} \
  --size_z      ${SIZE_Z} \
  --cpu         ${SLURM_CPUS} \
  > "${LOG}" 2>&1

# GNINA --score_only prints individual labeled lines (not a mode table):
#   Affinity: -X.XXX (kcal/mol)
#   CNNscore: X.XXX
#   CNNaffinity: X.XXX
vina_score=$(grep -m1 "^Affinity:" "${LOG}" | awk '{print $2}')
cnn_score=$(grep -m1 "^CNNscore:" "${LOG}" | awk '{print $2}')
cnn_affinity=$(grep -m1 "^CNNaffinity:" "${LOG}" | awk '{print $2}')

[ -z "${cnn_score}" ]      && cnn_score="N/A"
[ -z "${cnn_affinity}" ]  && cnn_affinity="N/A"

if [ -z "${vina_score}" ]; then
  echo " Could not parse GNINA output for ${pdbid}. Check ${LOG}."
  exit 1
fi

echo "${cnn_score} ${cnn_affinity} ${vina_score}" > "${RESULT}"
echo " ${pdbid}: CNN=${cnn_score}  CNNaff=${cnn_affinity}  Vina=${vina_score}"
EOF

  sbatch "$JOB"
  echo " Submitted $pdbid [$box_source]"
  submitted=$((submitted+1))
done

echo ""
echo " Submitted $submitted GNINA score-only jobs | Skipped $skipped | Missing: $n_missing"
echo " Results written to $RESULTS_DIR"

```

3) Self-docking the native ligand into its receptor using Vina

```

#!/bin/bash
#
# 03_vina_selfdock.sh
#
# 3rd script in the docking workflow
#
# Self-docks each ligand into its native receptor using AutoDock Vina 1.2.7.
# Requests 3 poses (num_modes=3), don't need this unless ya care
#
# BOX MODE - set BOX_MODE to one of the following:
#
# "per_complex" - reads ???_vina_box.json for each complex (default)
#
# "global" - uses a single box defined by the GLOBAL_* variables below.
#
# "auto" - uses per-complex JSON if it exists for that PDB ID,
#          falls back to the global box if it does not.
#
set -euo pipefail

# Box mode: "per_complex" | "global" | "auto"
BOX_MODE="global"

# Global box (used when BOX_MODE=global or BOX_MODE=auto fallback)
# Set these to your binding site coordinates if using a global box.
GLOBAL_CENTER_X=-1.788

```

```

GLOBAL_CENTER_Y=30.962
GLOBAL_CENTER_Z=8.996
GLOBAL_SIZE_X=30.0
GLOBAL_SIZE_Y=30.0
GLOBAL_SIZE_Z=30.0

# SLURM settings
SLURM_MEM="4GB"
SLURM_TIME="0-0:15:00"
SLURM_CPUS=12
SLURM_PARTITION="XXXX"

# Vina settings
EXHAUSTIVENESS=8
CPU=4
NUM_MODES=3
MIN_RMSD=2

# Setup
JOBS_DIR="jobs/03_vina_selfdock"
RESULTS_DIR="results/03_vina_selfdock"
mkdir -p "$JOBS_DIR" "$RESULTS_DIR"

submitted=0
skipped=0
missing=0

# Main loop
for ligand in ???_ligand.pdbqt; do
    [ -e "$ligand" ] || { echo " No ligand files found (???_ligand_clean.pdbqt)."; exit 1; }

    pdbid="${ligand:0:4}"
    protein="${pdbid}_receptor.pdbqt"

    for f in "$protein" "$ligand"; do
        [ -f "$f" ] || { echo " Skipping $pdbid - missing $f"; missing=$((missing+1)); continue
2; }
    done

    OUT="${RESULTS_DIR}/${pdbid}_dock.pdbqt"
    if [ -f "$OUT" ] && [ -s "$OUT" ]; then
        echo " $pdbid already docked, skipping."
        skipped=$((skipped+1))
        continue
    fi

    # Resolve json/global box coordinates if necessary
    boxjson="${pdbid}_vina_box.json"
    box_source=""

    case "$BOX_MODE" in
        per_complex)
            if [ ! -f "$boxjson" ]; then
                echo " Skipping $pdbid - BOX_MODE=per_complex but $boxjson not found"
                missing=$((missing+1))
                continue
            fi
            CENTER_X=$(jq '.center[0]' "$boxjson")
            CENTER_Y=$(jq '.center[1]' "$boxjson")
            CENTER_Z=$(jq '.center[2]' "$boxjson")
            SIZE_X=$(jq '.size[0]' "$boxjson")
            SIZE_Y=$(jq '.size[1]' "$boxjson")
            SIZE_Z=$(jq '.size[2]' "$boxjson")
            box_source="per-complex JSON"
            ;;
        global)
            CENTER_X=$GLOBAL_CENTER_X
            CENTER_Y=$GLOBAL_CENTER_Y
            CENTER_Z=$GLOBAL_CENTER_Z
    esac
done

```

```

        SIZE_X=$GLOBAL_SIZE_X
        SIZE_Y=$GLOBAL_SIZE_Y
        SIZE_Z=$GLOBAL_SIZE_Z
        box_source="global box"
        ;;
    auto)
        if [ -f "$boxjson" ]; then
            CENTER_X=$(jq '.center[0]' "$boxjson")
            CENTER_Y=$(jq '.center[1]' "$boxjson")
            CENTER_Z=$(jq '.center[2]' "$boxjson")
            SIZE_X=$(jq '.size[0]' "$boxjson")
            SIZE_Y=$(jq '.size[1]' "$boxjson")
            SIZE_Z=$(jq '.size[2]' "$boxjson")
            box_source="per-complex JSON"
        else
            CENTER_X=$GLOBAL_CENTER_X
            CENTER_Y=$GLOBAL_CENTER_Y
            CENTER_Z=$GLOBAL_CENTER_Z
            SIZE_X=$GLOBAL_SIZE_X
            SIZE_Y=$GLOBAL_SIZE_Y
            SIZE_Z=$GLOBAL_SIZE_Z
            box_source="global box (no JSON found)"
        fi
        ;;
    *)
        echo " Unknown BOX_MODE: $BOX_MODE. Use per_complex, global, or auto."
        exit 1
        ;;
esac

CONF="${JOBS_DIR}/${pdbid}.conf"
JOB="${JOBS_DIR}/${pdbid}_selfdock.job"
LOG="${RESULTS_DIR}/${pdbid}_vina.log"

# Write Vina config file
cat > "$CONF" <<EOF
receptor      = $protein
ligand         = $ligand
center_x       = $CENTER_X
center_y       = $CENTER_Y
center_z       = $CENTER_Z
size_x         = $SIZE_X
size_y         = $SIZE_Y
size_z         = $SIZE_Z
exhaustiveness = $EXHAUSTIVENESS
cpu            = $CPU
num_modes      = $NUM_MODES
min_rmsd       = $MIN_RMSD
EOF

# Write SLURM job files
cat > "$JOB" <<EOF
#!/bin/bash
#SBATCH --job-name=${pdbid}_dock
#SBATCH --mem=${SLURM_MEM}
#SBATCH --time=${SLURM_TIME}
#SBATCH --cpus-per-task=${SLURM_CPUS}
#SBATCH --partition=${SLURM_PARTITION}
#SBATCH --output=${RESULTS_DIR}/${pdbid}_dock.out
#SBATCH --error=${RESULTS_DIR}/${pdbid}_dock.err

set -euo pipefail

module purge
module load arch/avx512
module load gcc/13.2.0
module load autodock-vina/1.2.7
module load jq/1.8.1

cd \${SLURM_SUBMIT_DIR}

```

```

echo " Self-docking ${pdbid} [box: ${box_source}]"

vina --config ${CONF} --out ${OUT} > ${LOG} 2>&1

if [ ! -s "${OUT}" ]; then
    echo " Vina produced no output for ${pdbid}. Check ${LOG}."
    exit 1
fi

n_poses=$(grep -c "^REMARK VINA RESULT:" "${OUT}" || true)
echo " ${pdbid}: ${n_poses} pose(s) written to ${OUT}"
EOF

    sbatch "$JOB"
    echo " Submitted $pdbid [$box_source]"
    submitted=$((submitted+1))
done

echo ""
echo " Submitted $submitted jobs | Skipped $skipped | Missing/failed: $missing"
echo " Outputs written to $RESULTS_DIR"
echo " After all jobs finish: bash 04_collect_vina_selfdock.sh"

```

4) Collect Vina results and calculate RMSD of top pose and native ligand pose, identify which ligand top scoring poses are not the closest to native ligand pose in RMSD

```

#!/bin/bash
#
# 04_collect_vina_selfdock.sh
#
# 4th script in workflow - Run AFTER all 03_vina_selfdock.sh jobs have finished.
#
# For each protein-ligand complex this script extracts energies for all 3 poses from
# the output PDBQT, splits poses into individual PDBQT files, calculates RMSD of each
# docked pose vs the native crystal pose (obrms), reports the best-scoring pose (always
# pose 1 from Vina) and its RMSD, reports which pose has the best RMSD across all 3 poses,
# flags complexes where pose 1 does NOT have the best RMSD
#
# The "selected pose" for downstream re-scoring is always pose 1 (top Vina rank).
# This script and the discordance flag is for graphing/analyzing whether
# Vina's top-ranked pose is also the closest to the native crystal pose.
#
# Output:
# results/04_selfdock_collect/poses/      - individual pose PDBQT files
# results/04_selfdock_collect/????_selfdock.result
# vina_selfdock_results.csv
#
# Dependencies: obrms (Open Babel), awk, python3
#

set -euo pipefail

RMSD_THRESHOLD=2.0    # Å, flags "successful" redocks in summary

# Setup

DOCK_DIR="results/03_vina_selfdock"
SPLIT_DIR="results/04_selfdock_collect/poses"
RESULT_DIR="results/04_selfdock_collect"
CSV_OUT="vina_selfdock_results.csv"

mkdir -p "$SPLIT_DIR" "$RESULT_DIR"

module purge
module load arch/avx512 b#dependencies on my machine to run openbabel
module load gcc/13.2.0 #dependencies on my machine to run openbabel
module load openbabel/3.1.1 # provides obrms; adjust module name if needed

# CSV header
# pose energies: all 3 Vina-ranked poses

```

```

# rmsd: RMSD of each pose vs the native crystal ligand
# top_pose_rmsd: RMSD of pose 1 (the top-ranked pose used for all scoring)
# best_rmsd_pose: which of the 3 poses is geometrically closest to native
# rmsd_discordant: TRUE when pose 1 is NOT the best-RMSD pose
# (useful for graphing: how often does Vina's top energy ≠ best geometry?)
echo "pdb_id,\
vina_pose1_energy,vina_pose2_energy,vina_pose3_energy,\
rmsd_pose1,rmsd_pose2,rmsd_pose3,\
top_pose_rmsd,\
best_rmsd_pose,best_rmsd_value,best_rmsd_pose_energy,\
rmsd_discordant" \
> "$CSV_OUT"

# Helper: split a multi-MODEL PDBQT into individual pose files
# Echoes the number of poses found

split_poses() {
    local pdbqt="$1"
    local prefix="$2"
    local pose=0
    local in_model=0
    local outfile=""

    while IFS= read -r line; do
        if [[ "$line" == MODEL* ]]; then
            pose=$((pose+1))
            outfile="{prefix}_pose${pose}.pdbqt"
            in_model=1
            # Do NOT write MODEL line - GNINA score_only rejects multi-model PDBQT
            : > "$outfile"
        elif [[ "$line" == ENDMDL* ]] && [ "$in_model" -eq 1 ]; then
            # Do NOT write ENDMDL either
            in_model=0
        elif [ "$in_model" -eq 1 ]; then
            printf '%s\n' "$line" >> "$outfile"
        fi
    done < "$pdbqt"

    echo "$pose"
}

# Helper: safe float comparison via python3
# Returns 0 (true) if $1 < $2

float_lt() {
    python3 -c "import sys; sys.exit(0 if float('$1') < float('$2') else 1)" 2>/dev/null
}

# Main loop

processed=0
n_missing=0
n_success=0    # pose1 RMSD <= RMSD_THRESHOLD
n_discordant=0

for ligand in ???_ligand.pdbqt; do
    [ -e "$ligand" ] || { echo " No ligand files found."; exit 1; }

    pdbid="{ligand:0:4}"
    dock_out="{DOCK_DIR}/${pdbid}_dock.pdbqt"

    if [ ! -s "$dock_out" ]; then
        echo " $pdbid - docked output missing or empty: $dock_out"
        n_missing=$((n_missing+1))
        continue
    fi

    echo " $pdbid"
    PREFIX="{SPLIT_DIR}/${pdbid}"

    # Split into individual pose files

```

```

n_poses=$(split_poses "$dock_out" "$PREFIX")

if [ "$n_poses" -eq 0 ]; then
    echo "    No MODEL blocks found in $dock_out"
    n_missing=$((n_missing+1))
    continue
fi

# Extract all pose energies
# REMARK VINA RESULT:  energy  rmsd_lb  rmsd_ub
mapfile -t energies < <(grep "^REMARK VINA RESULT:" "$dock_out" | awk '{print $4}')

e1="${energies[0]:-}"
e2="${energies[1]:-}"
e3="${energies[2]:-}"

if [ -z "$e1" ]; then
    echo "    Could not extract pose energies from $dock_out"
    n_missing=$((n_missing+1))
    continue
fi

# Calculate RMSD for each pose vs native crystal ligand
rmsd1="" rmsd2="" rmsd3=""
for i in $(seq 1 "$n_poses"); do
    pose_file="${PREFIX}_pose${i}.pdbqt"
    [ -f "$pose_file" ] || continue

    # obrms <reference> <query> - outputs a line with the RMSD value
    rmsd_raw=$(obrms "$ligand" "$pose_file" 2>/dev/null || true)

    # obrms output varies by version so format this for yours
    rmsd_val=$(echo "$rmsd_raw" | grep -oP '[-+]?[0-9]*\.[0-9]+' | tail -n1)

    eval "rmsd${i}=\"\${rmsd_val}\""
done

# Top pose (pose 1) RMSD - the primary metric
top_pose_rmsd="${rmsd1:-}"

# Best-RMSD pose across all poses
best_rmsd_pose=1
best_rmsd_value="${rmsd1:-999}"

for i in 2 3; do
    eval "ri=\"\${rmsd${i}}:-\""
    [ -z "$ri" ] && break
    if float_lt "$ri" "$best_rmsd_value"; then
        best_rmsd_pose=$i
        best_rmsd_value="$ri"
    fi
done

# Energy of the best-RMSD pose
eval "best_rmsd_pose_energy=\"\${e${best_rmsd_pose}}:-\""

# Discordance: is pose 1 NOT the geometrically best pose?
if [ "$best_rmsd_pose" -ne 1 ]; then
    discordant="TRUE"
    n_discordant=$((n_discordant+1))
else
    discordant="FALSE"
fi

# Flag successful redock
if [ -n "$top_pose_rmsd" ]; then
    if float_lt "$top_pose_rmsd" "$RMSD_THRESHOLD" || \
        python3 -c "import sys; sys.exit(0 if float('$top_pose_rmsd') ==
float('$RMSD_THRESHOLD') else 1)" 2>/dev/null; then
        n_success=$((n_success+1))
    fi
fi

```

```

fi

# Write per-complex result file
# (used by 05_gnina_rescore_docked.sh to find pose 1 file)
RESULT="${RESULT_DIR}/${pdbid}_selfdock.result"
cat > "$RESULT" <<RESEOF
pdbid=${pdbid}
n_poses=${n_poses}
e1=${e1}
e2=${e2:-}
e3=${e3:-}
rmsd1=${rmsd1:-}
rmsd2=${rmsd2:-}
rmsd3=${rmsd3:-}
top_pose_rmsd=${top_pose_rmsd:-}
best_rmsd_pose=${best_rmsd_pose}
best_rmsd_value=${best_rmsd_value}
best_rmsd_pose_energy=${best_rmsd_pose_energy:-}
discordant=${discordant}
pose1_file=${PREFIX}_pose1.pdbqt
RESEOF

# Append row to CSV
echo "${pdbid},${e1},${e2:-},${e3:-},\
${rmsd1:-},${rmsd2:-},${rmsd3:-},\
${top_pose_rmsd:-},\
${best_rmsd_pose},${best_rmsd_value},${best_rmsd_pose_energy:-},\
${discordant}" >> "$CSV_OUT"

echo "    E1=${e1} kcal/mol | RMSD_pose1=${rmsd1:-N/A} Å |
BestRMSD=pose${best_rmsd_pose} (${best_rmsd_value} Å, ${best_rmsd_pose_energy:-N/A} kcal/mol) |
Discordant=${discordant}"
processed=$((processed+1))
done

# Summary
echo ""
echo " Self-dock collection complete"
echo " Processed:                $processed"
echo " Missing/failed:            $n_missing"
echo " Pose 1 RMSD ≤ ${RMSD_THRESHOLD} Å:      $n_success / $processed"
echo " Discordant (pose1 ≠ best-RMSD): $n_discordant / $processed"
echo " CSV: $CSV_OUT"
echo ""

if [ "$n_discordant" -gt 0 ]; then
echo "Discordant complexes (pose 1 energy rank ≠ best RMSD pose):"
awk -F',' ' $11=="TRUE" {printf "    %-6s E1=%-8s RMSD_pose1=%-6s BestRMSD=pose%s (%-6s
Å)\n", $1,$2,$5,$9,$10}' "$CSV_OUT"
echo ""
fi

echo "Next step: bash 05_gnina_rescore_docked.sh"

```

5) Rescoring the self-docked native ligand into its receptor Vina result using GNINA

```

#!/bin/bash
#
# 05_gnina_rescore_docked.sh
# 5th script in the workflow
#
# Run AFTER 04_collect_vina_selfdock.sh has finished.
#
# Re-scores the TOP-RANKED Vina pose (pose 1) for each complex using GNINA
# score_only. Pose 1 is always Vina's best-energy pose.
#
# NOTE: Box coordinates are required by GNINA even in --score_only mode,
#       same as Vina. Uses the same BOX_MODE logic as scripts 01-03.
#
# Output per complex:

```

```

# results/05_gnina_rescore/????_gnina_rescore.result
# Format (space-separated): CNNscore CNNaffinity vinascore
#

set -euo pipefail

BOX_MODE="global"

GLOBAL_CENTER_X=-1.789
GLOBAL_CENTER_Y=30.962
GLOBAL_CENTER_Z=8.996
GLOBAL_SIZE_X=30.0
GLOBAL_SIZE_Y=30.0
GLOBAL_SIZE_Z=30.0

GNINA_MODULE="gnina"

# SLURM
SLURM_MEM="4GB"
SLURM_TIME="0-0:10:00"
SLURM_CPUS=4
SLURM_PARTITION="genoacpu-b"

# Setup

COLLECT_DIR="results/04_selfdock_collect"
JOBS_DIR="jobs/05_gnina_rescore"
RESULTS_DIR="results/05_gnina_rescore"
mkdir -p "$JOBS_DIR" "$RESULTS_DIR"

submitted=0
skipped=0
n_missing=0

# Main loop
for ligand in ????.ligand.pdbqt; do
    [ -e "$ligand" ] || { echo " No ligand files found."; exit 1; }

    pdbid="${ligand:0:4}"
    protein="${pdbid}_receptor.pdbqt"
    result_file="${COLLECT_DIR}/${pdbid}_selfdock.result"

    if [ ! -f "$result_file" ]; then
        echo " $pdbid - no selfdock result file (run Step 4 first)"
        n_missing=$((n_missing+1))
        continue
    fi

    pose1_file=$(grep "^pose1_file=" "$result_file" | cut -d'=' -f2)

    if [ ! -f "$pose1_file" ]; then
        echo " $pdbid - pose 1 file not found: $pose1_file"
        n_missing=$((n_missing+1))
        continue
    fi

    RESULT="${RESULTS_DIR}/${pdbid}_gnina_rescore.result"
    if [ -f "$RESULT" ]; then
        echo " $pdbid already re-scored, skipping."
        skipped=$((skipped+1))
        continue
    fi

    # Resolve box coordinates (same logic as scripts 01-03)
    boxjson="${pdbid}_vina_box.json"
    box_source=""

    case "$BOX_MODE" in
        per_complex)

```

```

        if [ ! -f "$boxjson" ]; then
            echo " Skipping $pdbid - BOX_MODE=per_complex but $boxjson not found"
            n_missing=$((n_missing+1))
            continue
        fi
        CENTER_X=$(jq '.center[0]' "$boxjson")
        CENTER_Y=$(jq '.center[1]' "$boxjson")
        CENTER_Z=$(jq '.center[2]' "$boxjson")
        SIZE_X=$(jq '.size[0]' "$boxjson")
        SIZE_Y=$(jq '.size[1]' "$boxjson")
        SIZE_Z=$(jq '.size[2]' "$boxjson")
        box_source="per-complex JSON"
        ;;
    global)
        CENTER_X=$GLOBAL_CENTER_X; CENTER_Y=$GLOBAL_CENTER_Y; CENTER_Z=$GLOBAL_CENTER_Z
        SIZE_X=$GLOBAL_SIZE_X; SIZE_Y=$GLOBAL_SIZE_Y; SIZE_Z=$GLOBAL_SIZE_Z
        box_source="global box"
        ;;
    auto)
        if [ -f "$boxjson" ]; then
            CENTER_X=$(jq '.center[0]' "$boxjson")
            CENTER_Y=$(jq '.center[1]' "$boxjson")
            CENTER_Z=$(jq '.center[2]' "$boxjson")
            SIZE_X=$(jq '.size[0]' "$boxjson")
            SIZE_Y=$(jq '.size[1]' "$boxjson")
            SIZE_Z=$(jq '.size[2]' "$boxjson")
            box_source="per-complex JSON"
        else
            CENTER_X=$GLOBAL_CENTER_X; CENTER_Y=$GLOBAL_CENTER_Y; CENTER_Z=$GLOBAL_CENTER_Z
            SIZE_X=$GLOBAL_SIZE_X; SIZE_Y=$GLOBAL_SIZE_Y; SIZE_Z=$GLOBAL_SIZE_Z
            box_source="global box (no JSON found)"
        fi
        ;;
    *)
        echo " Unknown BOX_MODE: $BOX_MODE"; exit 1 ;;
esac

LOG="${RESULTS_DIR}/${pdbid}_gnina_rescore.log"
JOB="${JOBS_DIR}/${pdbid}_gnina_rescore.job"

cat > "$JOB" <<EOF
#!/bin/bash
#SBATCH --job-name=${pdbid}_grescore
#SBATCH --mem=${SLURM_MEM}
#SBATCH --time=${SLURM_TIME}
#SBATCH --cpus-per-task=${SLURM_CPUS}
#SBATCH --partition=${SLURM_PARTITION}
#SBATCH --output=${RESULTS_DIR}/${pdbid}_gnina_rescore.out
#SBATCH --error=${RESULTS_DIR}/${pdbid}_gnina_rescore.err

set -euo pipefail

module purge
module load arch/avx512
module load gcc/13.2.0
module load jq/1.8.1
module load ${GNINA_MODULE}

cd \${SLURM_SUBMIT_DIR}

echo " GNINA re-score (pose 1): ${pdbid} [box: ${box_source}]"

# Pose file has MODEL/ENDMDL stripped (done in Step 4) so GNINA can read it.
# Box coordinates required even in score_only mode.
gnina \\\
    --receptor ${protein} \\\
    --ligand "${pose1_file}" \\\
    --score_only \\\
    --center_x ${CENTER_X} \\\
    --center_y ${CENTER_Y} \\\
    --center_z ${CENTER_Z} \\\

```

```

--size_x      ${SIZE_X} \\
--size_y      ${SIZE_Y} \\
--size_z      ${SIZE_Z} \\
--cpu         ${SLURM_CPUS} \\
> "${LOG}" 2>&1

# GNINA --score_only prints individual labeled lines (not a mode table).
# Format:
#   Affinity: -X.XXX (kcal/mol)
#   CNNscore: X.XXX
#   CNNaffinity: X.XXX
vina_score=$(grep -m1 "^Affinity:" "${LOG}" | awk '{print \2}')
cnn_score=$(grep -m1 "^CNNscore:" "${LOG}" | awk '{print \2}')
cnn_affinity=$(grep -m1 "^CNNaffinity:" "${LOG}" | awk '{print \2}')

[ -z "${cnn_score}" ]      && cnn_score="N/A"
[ -z "${cnn_affinity}" ]  && cnn_affinity="N/A"

if [ -z "${vina_score}" ]; then
    echo " Could not parse GNINA output for ${pdbid}. Check ${LOG}."
    exit 1
fi

echo "${cnn_score} ${cnn_affinity} ${vina_score}" > "${RESULT}"
echo " ${pdbid}: CNN=${cnn_score}  CNNaff=${cnn_affinity}  Vina=${vina_score}"
EOF

    sbatch "$JOB"
    echo " Submitted $pdbid  [$box_source]"
    submitted=$((submitted+1))
done

echo ""
echo " Submitted $submitted GNINA re-score jobs | Skipped $skipped | Missing: $n_missing"
echo " Results written to $RESULTS_DIR"
echo " After all jobs finish: python3 06_collect_all.py"

```

7.4.6 Cross-Docked Workflow for RCSB CDK2 Dataset

1) Cross-dock all ligands into all receptors using Vina - needs to chunk into submission arrays

```

#!/bin/bash
#
# 07_vina_crossdock.sh
#
# Cross-dock every ligand into every receptor using AutoDock Vina FULL
# SEARCH (not --score_only) - the ligand is allowed to move within the
# active site, exactly as in 03_vina_selfdock.sh.
#
# Search settings (EXHAUSTIVENESS, NUM_MODES, MIN_RMSD) are matched to
# 03_vina_selfdock.sh so cross-dock and self-dock results are produced
# under identical search conditions.
#
# Output per pair: results/07_crossdock/{lig}_lig_{rec}_rec_dock.pdbqt
#
# Box settings load from global_docking_box.json (the bounding box over
# ALL aligned ligand poses) - this is intentionally larger than the
# per-structure self-dock box, since the ligand must be able to explore
# the active site of a receptor it was not originally crystallized with.
#
# Handles cluster limits by detecting MaxArraySize (array index limit per job),
# using CHUNK_SIZE <= MaxArraySize tasks per submission, submitting ONLY the
# first chunk immediately, each chunk's final task self-resubmits the next chunk,
# so only 1 chunk (CHUNK_SIZE tasks) is ever in the queue
# not necessary with a smaller dataset/larger number of job limit, although arrays
# can stop you from flooding your cluster

set -euo pipefail

BOX_JSON="global_docking_box.json"

```

```

GLOBAL_CENTER_X=-1.12
GLOBAL_CENTER_Y=30.54
GLOBAL_CENTER_Z=17.87
GLOBAL_SIZE_X=56.49
GLOBAL_SIZE_Y=44.17
GLOBAL_SIZE_Z=60.50

# Vina settings
EXHAUSTIVENESS=8
CPU=4
NUM_MODES=3
MIN_RMSD=2

# SLURM settings

SLURM_MEM="4GB"
SLURM_TIME="0-0:20:00"
SLURM_CPUS=12
SLURM_PARTITION="genoacpu-b"
SLURM_ACCOUNT="def-davisrl"

# Max tasks running simultaneously
MAX_RUNNING=200

# Tasks per chunk
CHUNK_SIZE=500

RESULTS_DIR="results/07_crossdock"
JOBS_DIR="jobs/07_crossdock"
PAIRS_FILE="jobs/07_crossdock_pairs.txt"
mkdir -p "$RESULTS_DIR" "$JOBS_DIR"

if [ -f "$BOX_JSON" ]; then
    GLOBAL_CENTER_X=$(jq -r '.center[0]' "$BOX_JSON")
    GLOBAL_CENTER_Y=$(jq -r '.center[1]' "$BOX_JSON")
    GLOBAL_CENTER_Z=$(jq -r '.center[2]' "$BOX_JSON")
    GLOBAL_SIZE_X=$(jq -r '.size[0]' "$BOX_JSON")
    GLOBAL_SIZE_Y=$(jq -r '.size[1]' "$BOX_JSON")
    GLOBAL_SIZE_Z=$(jq -r '.size[2]' "$BOX_JSON")
    echo "Box loaded from $BOX_JSON"
    echo "  Centre: ($GLOBAL_CENTER_X, $GLOBAL_CENTER_Y, $GLOBAL_CENTER_Z)"
    echo "  Size:   ($GLOBAL_SIZE_X, $GLOBAL_SIZE_Y, $GLOBAL_SIZE_Z)"
fi

# Build pairs list to do cross-docking
# Within this script a pair counts as "done" only if its output PDBQT exists, is non-empty,
# AND contains at least one "REMARK VINA RESULT:" line (i.e. at least
# one valid pose was produced)

echo ""
echo "Step 1: Building pairs list"

> "$PAIRS_FILE"
n_pairs=0; n_skip=0; n_invalid=0

for ligand in ????.ligand.pdbqt; do
    [ -e "$ligand" ] || { echo " No ????.ligand.pdbqt files found"; exit 1; }
    lig_id="${ligand:0:4}"
    for receptor in ????.receptor.pdbqt; do
        [ -e "$receptor" ] || continue
        rec_id="${receptor:0:4}"
        OUT="${RESULTS_DIR}/${lig_id}_lig_${rec_id}_rec_dock.pdbqt"
        if [ -s "$OUT" ] && grep -q "^REMARK VINA RESULT:" "$OUT" 2>/dev/null; then
            n_skip=$((n_skip+1)); continue
        elif [ -f "$OUT" ]; then
            echo "  Invalid/empty output ${lig_id}-${rec_id} - requeuing"
            rm -f "$OUT"; n_invalid=$((n_invalid+1))
        fi
        echo "${lig_id} ${rec_id} ${ligand} ${receptor}" >> "$PAIRS_FILE"
        n_pairs=$((n_pairs+1))
    done
done

```

```

done
done

echo " Pairs to run:      $n_pairs"
echo " Already done:     $n_skip"
[ $n_invalid -gt 0 ] && echo " Invalid (requeued): $n_invalid"

if [ "$n_pairs" -eq 0 ]; then
    echo " All pairs complete. Run: python3 08_collect_crossdock.py"
    exit 0
fi

# Write resubmit helper
# Called by the last task of each chunk to submit the next chunk.

RESUBMIT="${JOBS_DIR}/resubmit_next_chunk.sh"

cat > "$RESUBMIT" << RESUBEOF
#!/bin/bash
# resubmit_next_chunk.sh <next_offset>
# Submits the next chunk of the cross-dock array.
# Called automatically by the last task of the previous chunk.

set -euo pipefail

NEXT_OFFSET=\$1
PAIRS_FILE="${PAIRS_FILE}"
WORKER="${JOBS_DIR}/crossdock_worker.sh"
CHUNK_SIZE=${CHUNK_SIZE}
MAX_RUNNING=${MAX_RUNNING}
RESULTS_DIR="${RESULTS_DIR}"

n_pairs=\$(wc -l < "\$PAIRS_FILE")
[ "\$NEXT_OFFSET" -ge "\$n_pairs" ] && { echo "All chunks submitted."; exit 0; }

chunk_end=\$(( NEXT_OFFSET + CHUNK_SIZE - 1 ))
[ \$chunk_end -ge \$n_pairs ] && chunk_end=\$(( n_pairs - 1 ))
chunk_size=\$(( chunk_end - NEXT_OFFSET + 1 ))

JOB_ID=\$(sbatch \\\
    --array="0-\$((chunk_size - 1))%\${MAX_RUNNING}" \\\
    --export=ALL,PAIR_OFFSET=\${NEXT_OFFSET} \\\
    "\$WORKER" | awk '{print \$NF}')

echo "\$(date): Submitted chunk offset=\${NEXT_OFFSET}, \${chunk_size} tasks → job \${JOB_ID}" \\\
    >> "\${RESULTS_DIR}/resubmit.log"
echo " → Next chunk submitted: offset=\${NEXT_OFFSET}, \${chunk_size} tasks, job \${JOB_ID}"
RESUBEOF

chmod +x "$RESUBMIT"

# Write worker script

WORKER="${JOBS_DIR}/crossdock_worker.sh"

cat > "$WORKER" << EOF
#!/bin/bash
#SBATCH --job-name=cdk2_xdock
#SBATCH --mem=${SLURM_MEM}
#SBATCH --time=${SLURM_TIME}
#SBATCH --cpus-per-task=${SLURM_CPUS}
#SBATCH --partition=${SLURM_PARTITION}
#SBATCH --account=${SLURM_ACCOUNT}
#SBATCH --output=${RESULTS_DIR}/xdock_%A_%a.out
#SBATCH --error=${RESULTS_DIR}/xdock_%A_%a.err

set -euo pipefail

module purge
module load arch/avx512
module load gcc/13.2.0

```

```

module load autodock-vina/1.2.7
module load jq/1.8.1

cd \${SLURM_SUBMIT_DIR}

TASK_LINE=\$(( SLURM_ARRAY_TASK_ID + PAIR_OFFSET + 1 ))
LINE=\$(sed -n "\${TASK_LINE}p" "\${PAIRS_FILE}")
[ -z "\$LINE" ] && exit 0

LIG_ID=\$(echo "\$LINE" | awk '{print \$1}')
REC_ID=\$(echo "\$LINE" | awk '{print \$2}')
LIGAND=\$(echo "\$LINE" | awk '{print \$3}')
RECEPTOR=\$(echo "\$LINE" | awk '{print \$4}')
```

OUT="\\${RESULTS_DIR}/\\${LIG_ID}_lig_\\${REC_ID}_rec_dock.pdbqt"

LOG="\\${RESULTS_DIR}/\\${LIG_ID}_lig_\\${REC_ID}_rec_dock.log"

CONF="\\${JOBS_DIR}/\\${LIG_ID}_lig_\\${REC_ID}_rec.conf"

```

if [ -s "\$OUT" ] && grep -q "^REMARK VINA RESULT:" "\$OUT" 2>/dev/null; then
    exit 0
fi
rm -f "\$OUT"
```

Write per-pair Vina config files

```

cat > "\$CONF" <<CONFEOF
receptor      = \$RECEPTOR
ligand        = \$LIGAND
center_x      = ${GLOBAL_CENTER_X}
center_y      = ${GLOBAL_CENTER_Y}
center_z      = ${GLOBAL_CENTER_Z}
size_x        = ${GLOBAL_SIZE_X}
size_y        = ${GLOBAL_SIZE_Y}
size_z        = ${GLOBAL_SIZE_Z}
exhaustiveness = ${EXHAUSTIVENESS}
cpu           = ${CPU}
num_modes     = ${NUM_MODES}
min_rmsd      = ${MIN_RMSD}
CONFEOF
```

vina --config "\\$CONF" --out "\\$OUT" > "\\$LOG" 2>&1

```

if [ ! -s "\$OUT" ] || ! grep -q "^REMARK VINA RESULT:" "\$OUT"; then
    echo "Vina produced no valid poses for \${LIG_ID} to \${REC_ID}. Check \$LOG."
    exit 1
fi
```

```

rm -f "\$CONF"
n_poses=\$(grep -c "^REMARK VINA RESULT:" "\$OUT" || true)
echo " \${LIG_ID} to \${REC_ID}: \${n_poses} pose(s) written to \$OUT"
```

Self-resubmit: last task in this chunk submits the next chunk

```

n_pairs=\$(wc -l < "\${PAIRS_FILE}")
chunk_last=\$(( PAIR_OFFSET + ${CHUNK_SIZE} - 1 ))
[ \${chunk_last} -ge \${n_pairs} ] && chunk_last=\$(( n_pairs - 1 ))
last_task_id=\$(( chunk_last - PAIR_OFFSET ))
```

```

if [ "\$SLURM_ARRAY_TASK_ID" -eq "\$last_task_id" ]; then
    NEXT_OFFSET=\$(( PAIR_OFFSET + ${CHUNK_SIZE} ))
    if [ "\$NEXT_OFFSET" -lt "\${n_pairs}" ]; then
        bash "${RESUBMIT}" "\$NEXT_OFFSET"
    else
        echo "Final chunk complete. Run: python3 08_collect_crossdock.py"
    fi
fi
```

EOF

```

chmod +x "$WORKER"
```

Step 4: Detect MaxArraySize and validate CHUNK_SIZE

```

MAX_ARRAY_SIZE=$(scontrol show config 2>/dev/null \
```

```

| grep -i "MaxArraySize" \
| awk -F= '{print $2}' \
| tr -d ' ' | head -1)

if [[ -z "$MAX_ARRAY_SIZE" || "$MAX_ARRAY_SIZE" -lt 1 ]]; then
    MAX_ARRAY_SIZE=1000
fi

if [ "$CHUNK_SIZE" -gt "$MAX_ARRAY_SIZE" ]; then
    echo " CHUNK_SIZE ($CHUNK_SIZE) > MaxArraySize ($MAX_ARRAY_SIZE) - capping to
$MAX_ARRAY_SIZE"
    CHUNK_SIZE=$MAX_ARRAY_SIZE
fi

n_chunks=$(( (n_pairs + CHUNK_SIZE - 1) / CHUNK_SIZE ))

echo ""
echo "  MaxArraySize = $MAX_ARRAY_SIZE"
echo "  Chunk size   = $CHUNK_SIZE"
echo "  Total chunks = $n_chunks (only 1 submitted at a time - self-resubmitting)"
echo "  Max running  = $MAX_RUNNING"
echo "  Search:      exhaustiveness=${EXHAUSTIVENESS}, num_modes=${NUM_MODES},
min_rmsd=${MIN_RMSD} (matches self-dock)"
echo ""

# Step 5: Submit ONLY the first chunk

first_chunk_size=$(( CHUNK_SIZE < n_pairs ? CHUNK_SIZE : n_pairs ))

JOB_ID=$(sbatch \
  --array="0-$(first_chunk_size - 1)">${MAX_RUNNING}" \
  --export=ALL,PAIR_OFFSET=0 \
  "$WORKER" | awk '{print $NF}')

echo "  Chunk 1/$n_chunks: ${first_chunk_size} tasks (pairs 0-$(first_chunk_size-1)) - job
${JOB_ID} [starts immediately]"
echo "  Chunks 2-$n_chunks will self-submit as each chunk's last task completes."
echo ""
echo "Monitor:  squeue -u \${USER}"
echo "Progress:  ls ${RESULTS_DIR}/*_dock.pdbqt 2>/dev/null | wc -l (target: $(n_skip +
n_pairs))"
echo "Resubmit log: ${RESULTS_DIR}/resubmit.log"
echo ""
echo "After all tasks finish:"
echo "  python3 08_collect_crossdock.py"

```

2) Re-score all cross-docked ligands into all receptors using GNINA - needs to chunk into submission arrays

```

#!/bin/bash
#
# 09_gnina_crossdock.sh
#
# Re-scores the top-ranked Vina pose (pose 1) from each cross-docked pair
# using GNINA --score_only. Pose 1 is always Vina's best-energy pose,
# taken from the full-search output 07_vina_crossdock.sh now produces.
#
# GNINA piggybacks off of Vina's conformational search here and
# and just scores the final docked pose
#
# Needs to be run AFTER 07_vina_crossdock.sh has finished producing
# results/07_crossdock/????_lig????_rec_dock.pdbqt files.
#
# Output per pair:
#   results/09_gnina_crossdock/????_lig????_rec_gnina.result
#   Format (space-separated): CNNscore CNNaffinity vinascore
#
# Handles cluster limits by detecting MaxArraySize (array index limit per job)
# using MaxArraySize tasks per submission submitting ONLY the first chunk immediately

```

```

# Each chunk's final task self-resubmits the next chunk, so only 1 chunk is ever in the queue
#

set -euo pipefail

# USER CONFIG

BOX_JSON="global_docking_box.json"

GLOBAL_CENTER_X=-1.12
GLOBAL_CENTER_Y=30.54
GLOBAL_CENTER_Z=17.87
GLOBAL_SIZE_X=56.49
GLOBAL_SIZE_Y=44.17
GLOBAL_SIZE_Z=60.50

GNINA_MODULE="gnina"
SLURM_MEM="4GB"
SLURM_TIME="0-0:10:00"
SLURM_CPUS=4
SLURM_PARTITION="genoacpu-b"
SLURM_ACCOUNT="def-davisrl"

# Max tasks running simultaneously
MAX_RUNNING=200

# Tasks per chunk - keep well below MaxJobsPU (1000) and MaxArraySize
CHUNK_SIZE=500

VINA_DIR="results/07_crossdock"
RESULTS_DIR="results/09_gnina_crossdock"
POSE1_DIR="results/09_gnina_crossdock/pose1"
JOBS_DIR="jobs/09_gnina_crossdock"
PAIRS_FILE="jobs/09_gnina_crossdock_pairs.txt"
mkdir -p "$RESULTS_DIR" "$POSE1_DIR" "$JOBS_DIR"

if [ -f "$BOX_JSON" ]; then
    GLOBAL_CENTER_X=$(jq -r '.center[0]' "$BOX_JSON")
    GLOBAL_CENTER_Y=$(jq -r '.center[1]' "$BOX_JSON")
    GLOBAL_CENTER_Z=$(jq -r '.center[2]' "$BOX_JSON")
    GLOBAL_SIZE_X=$(jq -r '.size[0]' "$BOX_JSON")
    GLOBAL_SIZE_Y=$(jq -r '.size[1]' "$BOX_JSON")
    GLOBAL_SIZE_Z=$(jq -r '.size[2]' "$BOX_JSON")
    echo "Box loaded from $BOX_JSON"
    echo "  Centre: ($GLOBAL_CENTER_X, $GLOBAL_CENTER_Y, $GLOBAL_CENTER_Z)"
    echo "  Size: ($GLOBAL_SIZE_X, $GLOBAL_SIZE_Y, $GLOBAL_SIZE_Z)"
fi

# Build pairs list
# Only pairs with a completed Vina cross-dock output added to list

echo ""
echo "Building pairs list"
> "$PAIRS_FILE"
n_pairs=0; n_skip=0; n_invalid=0; n_no_vina=0

for ligand in ???_ligand.pdbqt; do
    [ -e "$ligand" ] || { echo " No ???_ligand.pdbqt files found"; exit 1; }
    lig_id="${ligand:0:4}"
    for receptor in ???_receptor.pdbqt; do
        [ -e "$receptor" ] || continue
        rec_id="${receptor:0:4}"

        VINA_OUT="${VINA_DIR}/${lig_id}_lig_${rec_id}_rec_dock.pdbqt"
        if [ ! -s "$VINA_OUT" ] || ! grep -q "^REMARK VINA RESULT:" "$VINA_OUT" 2>/dev/null; then
            n_no_vina=$((n_no_vina+1))
            continue
        fi

        RESULT="${RESULTS_DIR}/${lig_id}_lig_${rec_id}_rec_gnina.result"
    done
done

```

```

        if [ -f "$RESULT" ]; then
            last=$(cat "$RESULT" 2>/dev/null | awk '{print $NF}')
            n_tok=$(cat "$RESULT" 2>/dev/null | wc -w)
            if [ "$n_tok" -ge 1 ] && [[ "$last" =~ ^-?[0-9]+\.[0-9]*$ ]]; then
                n_skip=$((n_skip+1)); continue
            else
                echo "    Invalid result ${lig_id}→${rec_id} - requeuing"
                rm -f "$RESULT"; n_invalid=$((n_invalid+1))
            fi
        fi
        echo "${lig_id} ${rec_id} ${receptor} ${VINA_OUT}" >> "$PAIRS_FILE"
        n_pairs=$((n_pairs+1))
    done
done

echo " Pairs to run:          $n_pairs"
echo " Already done:         $n_skip"
echo " No Vina dock yet:      $n_no_vina (run 07_vina_crossdock.sh first)"
[ $n_invalid -gt 0 ] && echo " Invalid (requeued):  $n_invalid"

if [ "$n_pairs" -eq 0 ]; then
    echo " All eligible pairs done. Run: python3 08_collect_crossdock.py"
    exit 0
fi

# Write resubmit helper

RESUBMIT="${JOBS_DIR}/resubmit_next_chunk.sh"

cat > "$RESUBMIT" << RESUBEOF
#!/bin/bash
set -euo pipefail

NEXT_OFFSET=\$1
PAIRS_FILE="\${PAIRS_FILE}"
WORKER="\${JOBS_DIR}/gnina_crossdock_worker.sh"
CHUNK_SIZE=\${CHUNK_SIZE}
MAX_RUNNING=\${MAX_RUNNING}
RESULTS_DIR="\${RESULTS_DIR}"

n_pairs=\$(wc -l < "\$PAIRS_FILE")
[ "\$NEXT_OFFSET" -ge "\$n_pairs" ] && { echo "All chunks submitted."; exit 0; }

chunk_end=\$(( NEXT_OFFSET + CHUNK_SIZE - 1 ))
[ \$chunk_end -ge \$n_pairs ] && chunk_end=\$(( n_pairs - 1 ))
chunk_size=\$(( chunk_end - NEXT_OFFSET + 1 ))

JOB_ID=\$(sbatch \\\
    --array="0-\$((chunk_size - 1))%\${MAX_RUNNING}" \\\
    --export=ALL,PAIR_OFFSET=\${NEXT_OFFSET} \\\
    "\$WORKER" | awk '{print \$NF}')

echo "\$(date): Submitted chunk offset=\${NEXT_OFFSET}, \${chunk_size} tasks → job \${JOB_ID}" \\\
    >> "\${RESULTS_DIR}/resubmit.log"
echo " → Next chunk submitted: offset=\${NEXT_OFFSET}, \${chunk_size} tasks, job \${JOB_ID}"
RESUBEOF

chmod +x "$RESUBMIT"

# Write worker script
# Extracts pose 1 from the Vina cross-dock PDBQT (MODEL/ENDMDL stripped,
# same approach as 04_collect_vina_selfdock.sh's split_poses), then
# rescores ONLY that pose with GNINA --score_only.

WORKER="\${JOBS_DIR}/gnina_crossdock_worker.sh"

cat > "$WORKER" << EOF
#!/bin/bash
#SBATCH --job-name=cdk2_gnina_xdock
#SBATCH --mem=\${SLURM_MEM}
#SBATCH --time=\${SLURM_TIME}

```

```

#SBATCH --cpus-per-task=${SLURM_CPUS}
#SBATCH --partition=${SLURM_PARTITION}
#SBATCH --account=${SLURM_ACCOUNT}
#SBATCH --output=${RESULTS_DIR}/gnina_xdock_%A_%a.out
#SBATCH --error=${RESULTS_DIR}/gnina_xdock_%A_%a.err

set -euo pipefail

module purge
module load arch/avx512
module load gcc/13.2.0
module load jq/1.8.1
module load ${GNINA_MODULE}

cd \${SLURM_SUBMIT_DIR}

TASK_LINE=\${( SLURM_ARRAY_TASK_ID + PAIR_OFFSET + 1 )}
LINE=\$(sed -n "\${TASK_LINE}p" "\${PAIRS_FILE}")
[ -z "\$LINE" ] && exit 0

LIG_ID=\$(echo "\$LINE" | awk '{print \$1}')
REC_ID=\$(echo "\$LINE" | awk '{print \$2}')
RECEPTOR=\$(echo "\$LINE" | awk '{print \$3}')
VINA_OUT=\$(echo "\$LINE" | awk '{print \$4}')

RESULT="\${RESULTS_DIR}/\${LIG_ID}_lig_\${REC_ID}_rec_gnina.result"
LOG="\${RESULTS_DIR}/\${LIG_ID}_lig_\${REC_ID}_rec_gnina.log"
POSE1="\${POSE1_DIR}/\${LIG_ID}_lig_\${REC_ID}_rec_pose1.pdbqt"

if [ -f "\$RESULT" ]; then
    last=\$(cat "\$RESULT" 2>/dev/null | awk '{print \$NF}')
    [[ "\$last" =~ ^-?[0-9]+\.\?[0-9]*\$ ]] && exit 0
    rm -f "\$RESULT"
fi

# Extract pose 1 (first MODEL block), stripping MODEL/ENDMDL lines
# so GNINA --score_only accepts the file
awk '
/^MODEL/ { in_model++; if (in_model==1) next; else exit }
/^ENDMDL/ { if (in_model==1) exit; next }
in_model==1 { print }
' "\$VINA_OUT" > "\$POSE1"

if [ ! -s "\$POSE1" ]; then
    echo " Could not extract pose 1 for \${LIG_ID}→\${REC_ID} from \$VINA_OUT"
    exit 1
fi

gnina \\\
--receptor "\$RECEPTOR" \\\
--ligand "\$POSE1" \\\
--score_only \\\
--center_x \${GLOBAL_CENTER_X} \\\
--center_y \${GLOBAL_CENTER_Y} \\\
--center_z \${GLOBAL_CENTER_Z} \\\
--size_x \${GLOBAL_SIZE_X} \\\
--size_y \${GLOBAL_SIZE_Y} \\\
--size_z \${GLOBAL_SIZE_Z} \\\
--cpu \${SLURM_CPUS} \\\
> "\$LOG" 2>&1

cnn_score=\$(grep -m1 "^CNNscore:" "\$LOG" | awk '{print \$2}')
cnn_aff=\$(grep -m1 "^CNNaffinity:" "\$LOG" | awk '{print \$2}')
vina_score=\$(grep -m1 "^Affinity:" "\$LOG" | awk '{print \$2}')

[ -z "\${cnn_score}" ] && cnn_score="N/A"
[ -z "\${cnn_aff}" ] && cnn_aff="N/A"
[ -z "\${vina_score}" ] && { echo " No score for \$LIG_ID→\$REC_ID. Check \$LOG"; exit 1; }

echo "\${cnn_score} \${cnn_aff} \${vina_score}" > "\$RESULT"

```

```

# Self-resubmit: last task in this chunk submits the next chunk
n_pairs=$((wc -l < "${PAIRS_FILE}"))
chunk_last=$((PAIR_OFFSET + ${CHUNK_SIZE} - 1))
[ \${chunk_last} -ge \${n_pairs} ] && chunk_last=$((n_pairs - 1))
last_task_id=$((chunk_last - PAIR_OFFSET))

if [ "\${SLURM_ARRAY_TASK_ID}" -eq "\${last_task_id}" ]; then
    NEXT_OFFSET=$((PAIR_OFFSET + ${CHUNK_SIZE}))
    if [ "\${NEXT_OFFSET}" -lt "\${n_pairs}" ]; then
        bash "${RESUBMIT}" "\${NEXT_OFFSET}"
    else
        echo " Final chunk complete. Run: python3 08_collect_crossdock.py"
    fi
fi
EOF

chmod +x "$WORKER"

# Detect MaxArraySize and validate CHUNK_SIZE
MAX_ARRAY_SIZE=$(scontrol show config 2>/dev/null \
    | grep -i "MaxArraySize" \
    | awk -F= '{print $2}' \
    | tr -d ' ' | head -1)
[[ -z "$MAX_ARRAY_SIZE" || "$MAX_ARRAY_SIZE" -lt 1 ]] && MAX_ARRAY_SIZE=1000

if [ "$CHUNK_SIZE" -gt "$MAX_ARRAY_SIZE" ]; then
    echo " CHUNK_SIZE (${CHUNK_SIZE}) > MaxArraySize ($MAX_ARRAY_SIZE) - capping to $MAX_ARRAY_SIZE"
    CHUNK_SIZE=$MAX_ARRAY_SIZE
fi

n_chunks=$(( (n_pairs + CHUNK_SIZE - 1) / CHUNK_SIZE ))

echo ""
echo " MaxArraySize = $MAX_ARRAY_SIZE"
echo " Chunk size = $CHUNK_SIZE"
echo " Total chunks = $n_chunks (only 1 submitted at a time, self-resubmitting)"
echo " Max running = $MAX_RUNNING"
echo " Rescoring: Vina pose 1 only (GNINA performs no independent search)"
echo ""

# Submit ONLY the first chunk
first_chunk_size=$((CHUNK_SIZE < n_pairs ? CHUNK_SIZE : n_pairs))

JOB_ID=$(sbatch \
    --array="0-$(first_chunk_size - 1))%${MAX_RUNNING}" \
    --export=ALL,PAIR_OFFSET=0 \
    "$WORKER" | awk '{print $NF}')

echo " Chunk 1/$n_chunks: ${first_chunk_size} tasks (pairs 0-$(first_chunk_size-1)) → job ${JOB_ID} [starts immediately]"
echo " Chunks 2-$n_chunks will self-submit as each chunk's last task completes."
echo ""
echo "Monitor: squeue -u \${USER}"
echo "Progress: ls ${RESULTS_DIR}/*.result 2>/dev/null | wc -l (target: $((n_skip + n_pairs)))"
echo "Resubmit log: ${RESULTS_DIR}/resubmit.log"
echo ""
echo "After all tasks finish:"
echo " python3 08_collect_crossdock.py"

```

7.5 Random ChEMBL Ligand Dataset Fetch and Preparation, and ROCS Workflow

7.5.1 ChEMBL Ligand Random Dataset Fetch and Preparation Workflow

```
# -*- coding: utf-8 -*-
```

```

gather all pdb ids for ChEMBL CDK2 ligands
"""

import requests
import json

def get_cdk2_pdb_ids(uniprot_id="P24941"):
    """
    Query RCSB PDB Search API for all PDB IDs containing human CDK2 (UniProt P24941).
    Works in Google Colab without installing anything.
    """

    url = "https://search.rcsb.org/rcsbsearch/v2/query"

    query = {
        "query": {
            "type": "terminal",
            "service": "text",
            "parameters": {
                "attribute":
"rcsb_polymer_entity_container_identifiers.reference_sequence_identifiers.database_accession",
                "operator": "exact_match",
                "value": uniprot_id
            }
        },
        "request_options": {
            "return_all_hits": True
        },
        "return_type": "entry"
    }

    response = requests.post(url, json=query)
    if response.status_code != 200:
        raise RuntimeError("RCSB query failed:", response.text)

    data = response.json()
    pdb_ids = [item["identifier"].upper() for item in data.get("result_set", [])]

    return sorted(set(pdb_ids))

# RUN QUERY
pdb_ids = get_cdk2_pdb_ids()
print("Found", len(pdb_ids), "PDB entries for human CDK2")

# Write output file
with open("pdb_ids.txt", "w") as f:
    for pid in pdb_ids:
        f.write(pid + "\n")

print("Wrote pdb_ids.txt:")
print("\n".join(pdb_ids[:20]), "...") # preview first 20

"""download all cdk2 ligands"""

# Install required packages

import subprocess, sys
for _pkg in ("rdkit", "chembl_webresource_client", "prody"):
    subprocess.run([sys.executable, "-m", "pip", "install", "-q", _pkg], check=True)

"""
Robust CDK2 pipeline with reliable PDB ligand detection + error handling + caching.

Outputs:
    cdk2_standardized_assays.csv
    cdk2_unique_ligands.csv
    assays_grouped/*.csv
"""

import os
import time

```

```

import tempfile
import shutil
import requests
import pandas as pd
import prody
from chembl_webresource_client.new_client import new_client

from rdkit import Chem
from rdkit.Chem import AllChem, Descriptors
from rdkit.Chem.MolStandardize import rdMolStandardize
from rdkit.Chem.SaltRemover import SaltRemover

# Setup writable PDB cache for Colab
PDB_CACHE = "/content/pdb_cache"
os.makedirs(PDB_CACHE, exist_ok=True)
prody.pathPDBFolder(PDB_CACHE)
prody.confProDy(backup=False)

# RDKit Standardization Tools
remover = SaltRemover()
tautomerizer = rdMolStandardize.TautomerEnumerator()
normalizer = rdMolStandardize.Normalizer()
reionizer = rdMolStandardize.Reionizer()

def standardize_smiles(smiles):
    """Full RDKit standardization workflow: returns (std_smiles, inchikey) or None."""
    if not smiles:
        return None
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None

        mol = remover.StripMol(mol, dontRemoveEverything=True)
        Chem.SanitizeMol(mol)
        mol = normalizer.normalize(mol)
        mol = reionizer.reionize(mol)
        mol = tautomerizer.Canonicalize(mol)

        std_smiles = Chem.MolToSmiles(mol, canonical=True)
        inchikey = Chem.MolToInchiKey(mol)
        return std_smiles, inchikey
    except Exception:
        return None

# 1. Fetch CDK2 Activity Data from ChEMBL (with robust error handling)
chembl = new_client
TARGET_CDK2 = "CHEMBL301"
CHEMBL_CACHE_FILE = "chembl_cdk2_activities_cache.csv"

def fetch_chembl_activities_robust(target_id, cache_file=None, max_retries=3):
    """
    Robust ChEMBL fetching with caching, retry logic, exponential backoff
    """
    # Try cache first
    if cache_file and os.path.exists(cache_file):
        print(f"✓ Loading cached ChEMBL data from {cache_file}...")
        df = pd.read_csv(cache_file)
        print(f"✓ Loaded {len(df)} rows from cache.")
        return df

    # Fetch with retries
    for attempt in range(max_retries):
        try:
            print(f"Fetching CDK2 assays from ChEMBL (attempt {attempt + 1}/{max_retries})...")

```

```

    acts = chembl.activity.filter(target_chembl_id=target_id).only(
        "activity_id", "molecule_chembl_id", "assay_id", "standard_type",
        "standard_relation", "standard_value", "standard_units", "pchembl_value",
        "assay_description", "assay_type", "assay_test_type", "assay_category",
        "data_validity_comment", "potential_duplicate"
    )

    # Convert to dataframe (this is where the API call actually happens)
    acts_df = pd.DataFrame(acts)
    print(f" Fetched {len(acts_df)} rows.")

    # Cache for next time
    if cache_file:
        acts_df.to_csv(cache_file, index=False)
        print(f" Cached data to {cache_file}")

    return acts_df

except Exception as e:
    error_msg = str(e)[:300]
    print(f" Attempt {attempt + 1} failed:")
    print(f"      {error_msg}")

    if "500" in error_msg or "HttpApplicationError" in error_msg:
        print("    → ChEMBL server error detected (500 Internal Server Error)")

    if attempt < max_retries - 1:
        delay = 2 ** attempt * 10 # 10s, 20s, 40s
        print(f"    Retrying in {delay} seconds...")
        time.sleep(delay)
    else:
        print("\n All retry attempts exhausted.")
        print("    ChEMBL API may be experiencing issues.")
        print("    Try again later or check https://www.ebi.ac.uk/chembl/")
        raise

acts_df = fetch_chembl_activities_robust(TARGET_CDK2, cache_file=CHEMBL_CACHE_FILE,
max_retries=3)

# 2. Fetch Structures for Ligands (ChEMBL molecules)

mol_ids = sorted(acts_df["molecule_chembl_id"].dropna().unique())
print(f"\n Fetching {len(mol_ids)} molecule structures from ChEMBL...")

MOLECULES_CACHE_FILE = "chembl_molecules_cache.csv"

if os.path.exists(MOLECULES_CACHE_FILE):
    print(f" Loading cached molecule structures from {MOLECULES_CACHE_FILE}...")
    mols_df = pd.read_csv(MOLECULES_CACHE_FILE)
    print(f" Loaded {len(mols_df)} molecule structures from cache.")
else:
    rows = []
    for i, mid in enumerate(mol_ids):
        if i % 100 == 0:
            print(f" Progress: {i}/{len(mol_ids)}...")
        try:
            moldata = chembl.molecule.get(mid)
            if moldata is None:
                continue
            smiles = moldata.get("molecule_structures", {}).get("canonical_smiles")
            rows.append({"molecule_chembl_id": mid, "smiles": smiles})
        except Exception as e:
            print(f" Error fetching {mid}: {str(e)[:100]}")
            continue

    mols_df = pd.DataFrame(rows)
    mols_df.to_csv(MOLECULES_CACHE_FILE, index=False)
    print(f" Retrieved structures for {len(mols_df)} ligands.")
    print(f" Cached to {MOLECULES_CACHE_FILE}")

# 3. RDKit Standardize + Generate InChIKeys for ChEMBL molecules

```

```

STD_MOLECULES_CACHE_FILE = "chembl_molecules_standardized_cache.csv"

if os.path.exists(STD_MOLECULES_CACHE_FILE):
    print(f"\n Loading cached standardized molecules from {STD_MOLECULES_CACHE_FILE}...")
    std_df = pd.read_csv(STD_MOLECULES_CACHE_FILE)
    print(f" Loaded {len(std_df)} standardized molecules from cache.")
else:
    std_rows = []
    print("\nStandardizing ChEMBL ligands...")
    for idx, row in mols_df.iterrows():
        if idx % 100 == 0:
            print(f" Progress: {idx}/{len(mols_df)}...")
            smiles = row["smiles"]
            mid = row["molecule_chembl_id"]
            out = standardize_smiles(smiles)
            if out is None:
                continue
            std_smiles, inchikey = out
            std_rows.append({
                "molecule_chembl_id": mid,
                "original_smiles": smiles,
                "std_smiles": std_smiles,
                "std_inchikey": inchikey
            })

    std_df = pd.DataFrame(std_rows)
    std_df.to_csv(STD_MOLECULES_CACHE_FILE, index=False)
    print(f" Standardized {len(std_df)} molecules.")
    print(f" Cached to {STD_MOLECULES_CACHE_FILE}")

# 4. Robust PDB ligand extraction

# Exclude common non-ligand HETATM residue names (ions, solvents, protein mods)
EXCLUDE_LIGANDS = {
    "HOH", "WAT", "ACE", "NME", "NH2", "MSE", "SEP", "TPO", "PTR",      # protein mods / caps
    "NA", "K", "CL", "CA", "MG", "ZN", "CU", "MN", "FE", "SR", "B", "I",  # ions
    "EDO", "DMS", "MPD", "PEG", "GOL", "TRS", "BME", "SO4", "PO4", "MES" # solvents/additives
}

def is_valid_small_molecule_residue(residue, pdb_path, mw_cutoff=150.0):
    """
    Robust ligand detector:
    - exclude known junk (ions, solvents, mods)
    - require >=4 heavy atoms
    - require molecular weight >= mw_cutoff (default: 150 Da)
    """
    name = residue.getResname().strip().upper()
    if name in EXCLUDE_LIGANDS:
        return False

    atoms = list(residue.iterAtoms())
    heavy = [a for a in atoms if a.getAtomicNum() > 1]
    if len(heavy) < 4:
        return False

    # Extract ligand coords as RDKit molecule for MW calculation
    lig_mol = None
    try:
        with tempfile.NamedTemporaryFile(suffix=".pdb", delete=False) as fh:
            tmp = fh.name
            prody.writePDB(tmp, residue)
            lig_mol = Chem.MolFromPDBFile(tmp, sanitize=False, removeHs=False)
            os.remove(tmp)
    except Exception:
        return False

    if lig_mol is None:
        return False

    # Use largest fragment

```

```

frags = Chem.GetMolFragments(lig_mol, asMols=True)
lig_mol = max(frag, key=lambda m: m.GetNumAtoms())

# Attempt sanitization
try:
    Chem.SanitizeMol(lig_mol)
except Exception:
    pass

mw = Descriptors.MolWt(lig_mol)
return mw >= mw_cutoff

def fetch_smiles_from_rcsb(chem_id):
    """Return SMILES from RCSB ChemComp API or None."""
    url = f"https://data.rcsb.org/rest/v1/core/chemcomp/{chem_id}"
    try:
        r = requests.get(url, timeout=20)
    except Exception:
        return None
    if r.status_code != 200:
        return None
    try:
        js = r.json()
    except Exception:
        return None

    # normal canonical SMILES location
    smi = js.get("chem_comp", {}).get("smiles")
    if smi:
        return smi
    # fallback: check descriptors
    for d in js.get("pdbx_chem_comp_descriptor", []):
        if d.get("type") and d.get("type").upper() == "SMILES":
            return d.get("descriptor")
    return None

def extract_smiles_from_coords(pdb_path, lig_resname):
    """
    Extract ligand coordinates for lig_resname from pdb_path using ProDy and generate SMILES via
    RDKit.
    Returns SMILES string or None.
    """
    try:
        structure = prody.parsePDB(pdb_path)
    except Exception as e:
        print(f"Could not parse PDB for coordinate extraction: {e}")
        return None

    # select atoms with matching residue name (case-insensitive)
    sel = structure.select(f"resname {lig_resname}")
    if sel is None:
        sel = structure.select(f"resname {lig_resname.upper()}")
    if sel is None:
        print(f"Residue {lig_resname} not found in coordinates")
        return None

    # Write to temporary PDB
    tmp = None
    try:
        with tempfile.NamedTemporaryFile(suffix=".pdb", delete=False) as fh:
            tmp = fh.name
            prody.writePDB(tmp, sel)
            mol = Chem.MolFromPDBFile(tmp, sanitize=False, removeHs=False)
            if mol is None:
                print("RDKit could not parse ligand PDB coordinates.")
                return None

    # Keep largest fragment
    frags = Chem.GetMolFragments(mol, asMols=True)
    mol = max(frag, key=lambda m: m.GetNumAtoms())

```

```

# Try to sanitize
try:
    Chem.SanitizeMol(mol)
except Exception:
    try:
        mol = Chem.AddHs(mol)
        Chem.SanitizeMol(mol, catchErrors=True)
        mol = Chem.RemoveHs(mol)
    except Exception:
        pass

smi = Chem.MolToSmiles(mol, canonical=True)
return smi
finally:
    if tmp and os.path.exists(tmp):
        try:
            os.remove(tmp)
        except Exception:
            pass

def fetch_pdb_ligand_std_smiles_from_file(pdb_file, output_prefix="/content"):
    """
    Main function:
    - loops over PDB IDs in pdb_file
    - detects ligands (using ProDy)
    - for each ligand residue name: try RCSB chemcomp SMILES then fallback to coordinate
    extraction
    - standardizes SMILES and returns a set of standardized SMILES
    """
    ligand_smiles_std = set()

    with open(pdb_file, "r") as f:
        pdb_ids = [line.strip().upper() for line in f if line.strip()]

    for pdb_id in pdb_ids:
        print(f"\nProcessing PDB {pdb_id}")

        # 1) fetch PDB into writable cache
        try:
            pdb_path = prody.fetchPDB(pdb_id, folder=PDB_CACHE, compressed=False)
        except Exception as e:
            print(f"    Could not fetch PDB: {e}")
            continue

        # 2) parse structure and locate hetero residues (excluding water)
        try:
            structure = prody.parsePDB(pdb_path)
        except Exception as e:
            print(f"    Could not parse PDB file: {e}")
            continue

        het = structure.select("hetero and not water")
        if het is None:
            print("    No hetero atoms found.")
            continue

        # iterate residues via HierView
        hv = het.getHierView()
        ligand_resnames = []
        for chain in hv.iterChains():
            for res in chain.iterResidues():
                name = res.getResname().strip().upper()
                try:
                    if is_valid_small_molecule_residue(res, pdb_path, mw_cutoff=150.0):
                        ligand_resnames.append(name)
                except Exception as e:
                    print(f"        Validity check failed for residue {name}: "
                          f"{str(e)[:100]} - excluding (unverified).")

        ligand_resnames = sorted(set(ligand_resnames))
        if not ligand_resnames:

```

```

        print("    No valid small-molecule ligands detected after filtering.")
        continue

    print(f"    Found {len(ligand_resnames)} candidate ligand residue name(s):
{ligand_resnames}")

    # For each candidate ligand, attempt to obtain SMILES
    for lig in ligand_resnames:
        smi = None

        # Try RCSB chemcomp first
        smi = fetch_smiles_from_rcsb(lig)
        bond_source = "rcsb-template"
        if smi:
            print(f"    RCSB chemcomp returned SMILES for {lig}.")
        else:
            # Fallback: extract from coordinates
            print(f"    No RCSB SMILES for {lig}; extracting from coordinates "
                  f"(bond orders re-derived from geometry - verify if this "
                  f"ligand shows up as a false negative in the exclusion step)...")
            smi = extract_smiles_from_coords(pdb_path, lig)
            bond_source = "coord-geometry-fallback"

            if smi:
                print(f"    Extracted SMILES from coords for {lig}: {smi}")
            else:
                print(f"    Could not derive SMILES for {lig}; skipping.")
                continue

        # Standardize the SMILES
        std = standardize_smiles(smi)
        if std is None:
            print(f"    Standardization failed for {lig} (SMILES: {smi}); skipping.")
            continue
        std_smiles, _ = std
        print(f"    Standardized ligand SMILES: {std_smiles}")

        ligand_smiles_std.add(std_smiles)

        # Optional!!!! save a copy of the raw SMILES (+ bond source, so
        # coordinate-derived fallback cases are traceable/auditable)
        try:
            with open(os.path.join(output_prefix, f"{pdb_id}_{lig}_raw_smiles.txt"), "w") as
fh:
                fh.write(f"{smi}\n# bond_source={bond_source}\n")
        except Exception:
            pass

    print(f"\nTotal unique standardized PDB ligand SMILES: {len(ligand_smiles_std)}")
    return ligand_smiles_std

# Load your PDB list and run extraction
PDB_LIGANDS_CACHE_FILE = "pdb_ligands_cache.txt"

if os.path.exists(PDB_LIGANDS_CACHE_FILE):
    print(f"\n Loading cached PDB ligands from {PDB_LIGANDS_CACHE_FILE}...")
    with open(PDB_LIGANDS_CACHE_FILE, "r") as f:
        pdb_std_smiles = set(line.strip() for line in f if line.strip())
    print(f" Loaded {len(pdb_std_smiles)} PDB ligands from cache.")
else:
    pdb_std_smiles = fetch_pdb_ligand_std_smiles_from_file("pdb_ids.txt",
output_prefix="/content")
    # Cache the results
    with open(PDB_LIGANDS_CACHE_FILE, "w") as f:
        for smi in sorted(pdb_std_smiles):
            f.write(f"{smi}\n")
    print(f" Cached PDB ligands to {PDB_LIGANDS_CACHE_FILE}")

# 5. Exclude PDB Ligands from ChEMBL (FIXED LOGIC)

print("\nExcluding PDB ligands from ChEMBL dataset...")

```

```

# Filter std_df FIRST (remove PDB ligands)
std_df = std_df[~std_df["std_smiles"].isin(pdb_std_smiles)]

# THEN create unique_ligands from the filtered data
unique_ligands = std_df.drop_duplicates(subset="std_smiles").reset_index(drop=True)

# Merge with activities (no need to filter again since std_df is already filtered)
merged = acts_df.merge(std_df, on="molecule_chembl_id", how="left")

print("Remaining after exclusion:")
print("  standardized:", len(std_df))
print("  unique:", len(unique_ligands))
print("  merged assay rows:", len(merged))

# 6. Group Assays by Method

def classify_assay_method(row):
    atype = row.get("assay_type") or "NA"
    stype = row.get("standard_type") or "NA"
    ttype = row.get("assay_test_type") or "NA"
    return f"{atype}_{stype}_{ttype}"

merged["assay_method_group"] = merged.apply(classify_assay_method, axis=1)

os.makedirs("assays_grouped", exist_ok=True)
for gname, gdf in merged.groupby("assay_method_group"):
    safe = gname.replace("/", "_").replace(" ", "_")
    gdf.to_csv(f"assays_grouped/{safe}.csv", index=False)

# 7. Save Outputs

merged.to_csv("cdk2_standardized_assays.csv", index=False)
unique_ligands.to_csv("cdk2_unique_ligands.csv", index=False)

print("\n" + "="*80)
print(" PIPELINE COMPLETED SUCCESSFULLY")
print("="*80)
print("\nOutputs created:")
print("  cdk2_standardized_assays.csv")
print("  cdk2_unique_ligands.csv")
print("  assays_grouped/*.csv")
print("\nCache files (for faster re-runs):")
print(f"  saved {CHEMBL_CACHE_FILE}")
print(f"  saved {MOLECULES_CACHE_FILE}")
print(f"  saved {STD_MOLECULES_CACHE_FILE}")
print(f"  saved {PDB_LIGANDS_CACHE_FILE}")
print("\n--- DONE ---")

import os
import shutil
import math
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem

# Load your previous output CSV
df = pd.read_csv("cdk2_standardized_assays.csv")
print("Loaded rows:", len(df))

# Keep only Ki / pKi / IC50 / pIC50
keep_types = ["Ki", "pKi", "IC50", "pIC50"]
df = df[df["standard_type"].isin(keep_types)].copy()
print("Rows w/ Ki, pKi, IC50, pIC50:", len(df))

# FIXED data-quality filtering

n_before = len(df)

# units - Ki/IC50 rows only make sense in nM here; pKi/pIC50 rows are
# already unit-less log values so they pass through untouched.

```

```

is_log_type = df["standard_type"].isin(["pKi", "pIC50"])
is_nm_conc = (df["standard_type"].isin(["Ki", "IC50"])) & (df["standard_units"] == "nM")
df = df[is_log_type | is_nm_conc].copy()
print(f"Rows after unit check (nM only for Ki/IC50): {len(df)} (dropped {n_before - len(df)})")

# relation, keep only exact measurements, '=' covers most of them
# pKi/pIC50 rows sometimes have a blank/NaN relation, which this also accepts
n_before = len(df)
valid_relation = df["standard_relation"].isin(["=", "!="]) | df["standard_relation"].isna()
df = df[valid_relation].copy()
print(f"Rows after relation check (exact values only): {len(df)} (dropped {n_before - len(df)})")

# validity / duplicate flags
n_before = len(df)
if "data_validity_comment" in df.columns:
    df = df[df["data_validity_comment"].isna()].copy()
if "potential_duplicate" in df.columns:
    df = df[df["potential_duplicate"] != 1].copy()
print(f"Rows after validity/duplicate check: {len(df)} (dropped {n_before - len(df)})")

# Keep only rows with valid SMILES
df = df[~df["std_smiles"].isna()].copy()
print(f"Rows with valid SMILES:", len(df))

# Compute pKi and pIC50 fields
# Safe now: every remaining Ki/IC50 row has been confirmed to be in nM
# (step 2b-a above), so the *1e-9 conversion is valid for all rows reaching
# this point.
def compute_pKi(row):
    if row["standard_type"] == "Ki" and not pd.isna(row["standard_value"]):
        return -math.log10(row["standard_value"] * 1e-9)
    if row["standard_type"] == "pKi":
        return row["standard_value"]
    return None

def compute_pIC50(row):
    if row["standard_type"] == "IC50" and not pd.isna(row["standard_value"]):
        return -math.log10(row["standard_value"] * 1e-9)
    if row["standard_type"] == "pIC50":
        return row["standard_value"]
    return None

df["pKi_calc"] = df.apply(compute_pKi, axis=1)
df["pIC50_calc"] = df.apply(compute_pIC50, axis=1)

# Categorize ligands BEFORE collapsing

# Ligands with Ki/pKi (any assay)
ligands_with_ki = df[df["standard_type"].isin(["Ki", "pKi"])]["molecule_chembl_id"].unique()

# Ligands with IC50/pIC50 (any assay)
ligands_with_ic50 = df[df["standard_type"].isin(["IC50", "pIC50"])]["molecule_chembl_id"].unique()

set_ki = set(ligands_with_ki)
set_ic50 = set(ligands_with_ic50)

ligands_mixed = list(set_ki & set_ic50)
ligands_ki_only = list(set_ki - set_ic50)
ligands_ic50_only = list(set_ic50 - set_ki)

print("CATEGORY COUNTS (before assay collapse)")
print("Ki-only ligands:", len(ligands_ki_only))
print("IC50-only ligands:", len(ligands_ic50_only))
print("Mixed ligands (have both Ki AND IC50):", len(ligands_mixed))

# Now collapse to BEST assay within each group
def best_assays_for_ligands(df_full, ligand_list):
    """For Ki-only or IC50-only datasets: keep the best (lowest) value per molecule"""

```

```

df_sub = df_full[df_full["molecule_chembl_id"].isin(ligand_list)].copy()
df_sorted = df_sub.sort_values(by="standard_value", ascending=True)
return df_sorted.groupby("molecule_chembl_id", sort=False).first().reset_index()

def get_best_ki_and_ic50(df_full, ligand_list):
    """
    For mixed dataset: keep BOTH the best Ki AND best IC50 for each molecule.
    Returns a dataframe with one row per molecule containing both values.
    """
    results = []

    for chembl_id in ligand_list:
        mol_data = df_full[df_full["molecule_chembl_id"] == chembl_id].copy()

        # Get best Ki measurement (lowest standard_value)
        ki_data = mol_data[mol_data["standard_type"].isin(["Ki", "pKi"])]
        if len(ki_data) > 0:
            best_ki_row = ki_data.sort_values("standard_value", ascending=True).iloc[0]
            best_pKi = best_ki_row["pKi_calc"]
            ki_value = best_ki_row["standard_value"]
            ki_type = best_ki_row["standard_type"]
            ki_assay = best_ki_row.get("assay_description", "")
        else:
            best_pKi = None
            ki_value = None
            ki_type = None
            ki_assay = ""

        # Get best IC50 measurement (lowest standard_value)
        ic50_data = mol_data[mol_data["standard_type"].isin(["IC50", "pIC50"])]
        if len(ic50_data) > 0:
            best_ic50_row = ic50_data.sort_values("standard_value", ascending=True).iloc[0]
            best_pIC50 = best_ic50_row["pIC50_calc"]
            ic50_value = best_ic50_row["standard_value"]
            ic50_type = best_ic50_row["standard_type"]
            ic50_assay = best_ic50_row.get("assay_description", "")
        else:
            best_pIC50 = None
            ic50_value = None
            ic50_type = None
            ic50_assay = ""

        # Use the first row as base (for SMILES and other properties)
        base_row = mol_data.iloc[0].to_dict()

        # Update with both Ki and IC50 information
        base_row.update({
            # Ki data
            "best_pKi": best_pKi,
            "Ki_value": ki_value,
            "Ki_type": ki_type,
            "Ki_assay": ki_assay,
            # IC50 data
            "best_pIC50": best_pIC50,
            "IC50_value": ic50_value,
            "IC50_type": ic50_type,
            "IC50_assay": ic50_assay
        })

        results.append(base_row)

    return pd.DataFrame(results)

# Process each dataset type
df_ki_only = best_assays_for_ligands(df, ligands_ki_only)
df_ic50_only = best_assays_for_ligands(df, ligands_ic50_only)
df_mixed = get_best_ki_and_ic50(df, ligands_mixed) # <-- FIXED: Now keeps BOTH values

print("\n FINAL DATASET COUNTS")
print("Ki-only:", len(df_ki_only))
print("IC50-only:", len(df_ic50_only))

```

```

print("Mixed (with BOTH Ki and IC50):", len(df_mixed))

if len(df_mixed) > 0:
    print("\n MIXED DATASET STATISTICS")
    print(f"Molecules with both measurements: {len(df_mixed)}")
    print(f"Average pKi: {df_mixed['best_pKi'].mean():.2f} ± {df_mixed['best_pKi'].std():.2f}")
    print(f"Average pIC50: {df_mixed['best_pIC50'].mean():.2f} ± {df_mixed['best_pIC50'].std():.2f}")

    # Calculate correlation
    import numpy as np
    valid_pairs = df_mixed[['best_pKi', 'best_pIC50']].dropna()
    if len(valid_pairs) > 1:
        correlation = np.corrcoef(valid_pairs['best_pKi'], valid_pairs['best_pIC50'])[0, 1]
        print(f"Pearson correlation (Ki vs IC50): {correlation:.3f}")

# 7. SMILES → PDB generation

EMBED_SEED = 42

def smiles_to_pdb(smiles, outfile):
    if smiles is None or pd.isna(smiles):
        return False
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return False
    mol = Chem.AddHs(mol)

    params = AllChem.ETKDGv3()
    params.randomSeed = EMBED_SEED
    ok = AllChem.EmbedMolecule(mol, params)
    if ok != 0:
        params.useRandomCoords = True
        ok = AllChem.EmbedMolecule(mol, params)
    if ok != 0:
        return False

    try:
        AllChem.UFFOptimizeMolecule(mol)
    except Exception:
        # optimization failing is non-fatal – keep the embedded (unoptimized)
        # geometry rather than discarding the ligand entirely
        pass

    Chem.MolToPDBFile(mol, outfile)
    return True

# 8. Create dataset folder and dump files
def create_dataset_folder(df_subset, folder_name):
    # Remove existing folder entirely to avoid old files accumulating
    if os.path.exists(folder_name):
        print(f"Removing existing folder: {folder_name}")
        shutil.rmtree(folder_name)

    os.makedirs(folder_name, exist_ok=True)
    pdb_dir = os.path.join(folder_name, "pdb")
    os.makedirs(pdb_dir, exist_ok=True)

    # Deduplicate defensively (one row per molecule_chembl_id)
    df_unique = df_subset.drop_duplicates(subset=["molecule_chembl_id"],
    keep="first").reset_index(drop=True)

    # Write spreadsheet
    csv_path = os.path.join(folder_name, "ligands.csv")
    df_unique.to_csv(csv_path, index=False)

    # Write PDBs
    written = 0
    for idx, row in df_unique.iterrows():
        smi = row.get("std_smiles", None)
        chembl_id = row.get("molecule_chembl_id", f"mol_{idx}")

```

```

        outfile = os.path.join(pdb_dir, f"{chembl_id}.pdb")
        ok = smiles_to_pdb(smi, outfile)
        if ok:
            written += 1
        else:
            # if conversion failed, remove partial file if present
            if os.path.exists(outfile):
                try:
                    os.remove(outfile)
                except Exception:
                    pass

    print(f" Created dataset: {folder_name}")
    print(f"  Ligands (rows in CSV): {len(df_unique)}")
    print(f"  PDBs written: {written}")
    print(f"  Spreadsheet: {csv_path}")
    print(f"  PDB folder: {pdb_dir}")

# 9. Sampling logic (final rules)
def sample_sets(df_source, base_name):
    n = len(df_source)

    # Determine set_size and replace policy
    if n >= 300:
        set_size = 100
        replace = False
    elif n >= 225:
        set_size = 75
        replace = False
    else:
        set_size = 75
        replace = True

    # If replace is False but set_size > n, cap it to n to avoid errors
    if not replace and set_size > n:
        set_size = n

    print(f"\nCreating 3 sets of size {set_size} for {base_name} (replace={replace};
source_size={n})")

    if not replace and n >= set_size * 3:
        remaining = df_source.copy()
        for i in range(1, 4):
            sample_df = remaining.sample(set_size, replace=False, random_state=42 + i)
            remaining = remaining.drop(sample_df.index)
            folder = os.path.join(base_name, f"set_{i}")
            create_dataset_folder(sample_df, folder)
    else:
        if not replace:
            print(f"  Pool size ({n}) is too small to draw 3 disjoint sets "
                  f"of {set_size} - sets will overlap.")
        for i in range(1, 4):
            sample_df = df_source.sample(set_size, replace=replace, random_state=42 + i)
            folder = os.path.join(base_name, f"set_{i}")
            create_dataset_folder(sample_df, folder)

# 10. Generate datasets

# Ki-only dataset, up to 3 sets
if len(df_ki_only) > 0:
    sample_sets(df_ki_only, "sample_sets_ki_only")

# IC50-only dataset, up to 3 sets
if len(df_ic50_only) > 0:
    sample_sets(df_ic50_only, "sample_sets_ic50_only")

# Mixed dataset, ALL ligands (single folder)
# This now contains BOTH Ki and IC50 values in separate columns
if len(df_mixed) > 0:
    create_dataset_folder(df_mixed, "sample_sets_mixed")
    print("\n Mixed dataset contains both Ki and IC50 for each molecule")

```

```

    print(" Use 'best_pKi' and 'best_pIC50' columns for separate analyses")
else:
    print("\n No mixed Ki/IC50 ligands available.")

print("\n--- DONE ---")
print("\nOutput summary:")
print("- Ki-only sets: 3 sampled sets in 'sample_sets_ki_only/'")
print("- IC50-only sets: 3 sampled sets in 'sample_sets_ic50_only/'")
print("- Mixed set: All molecules with BOTH measurements in 'sample_sets_mixed/'")
print("\nFor mixed dataset analysis, see columns:")
print(" - best_pKi: Best Ki measurement (for graphing)")
print(" - best_pIC50: Best IC50 measurement (for graphing)")
print(" - Ki_assay / IC50_assay: Assay descriptions")

import os

for root, dirs, files in os.walk("/content", topdown=True):
    print(root, dirs, files)

import shutil

try:
    from google.colab import files as _colab_files
    IN_COLAB = True
except ImportError:
    _colab_files = None
    IN_COLAB = False

folders = {
    "mixed": "/content/sample_sets_mixed",
    "ic50_only": "/content/sample_sets_ic50_only",
    "ki_only": "/content/sample_sets_ki_only",
}

for zipname, folder in folders.items():
    if not os.path.exists(folder):
        print(f" Skipping {zipname}: {folder} does not exist.")
        continue
    print(f"Zipping {folder} to {zipname}.zip ...")
    shutil.make_archive(zipname, 'zip', folder)
    if IN_COLAB:
        _colab_files.download(f"{zipname}.zip")
    else:
        print(f" (not running in Colab - {zipname}.zip left in the working directory)")

```

7.5.2 ChEMBL Ligand Random Dataset OMEGA Conformer Generation and ROCS Prep

a) Bash submission script

```

#!/bin/bash
# submit_conformers.sh
#
# Submits a SLURM array job for gen_conformers.py.
# One task per dataset folder - each task generates all OEBs for that folder.
#
# After all tasks complete, verify with:
#   python gen_conformers.py --check --root <ROOT>
#
# Then submit the ROCS step:
#   bash submit_rocs_array.sh
#
# Options and settings for running
# --conformers  omega OR single  Conformer engine (default: omega)
# --max_confs   N                 Max conformers per ligand (default: 200)
# --root        DIR               Dataset root (default: .)
# --time        HH:MM:SS          Wall time per task (default: 04:00:00)
# --mem         MEM               Memory per task (default: 8G)
# --account     ACCOUNT           SLURM account
# --dry_run     Print job script without submitting

```

```

set -euo pipefail

# Defaults
CONFORMERS="omega"
MAX_CONFS=200
ROOT="."
WALL_TIME="04:00:00"
MEM="8G"
ACCOUNT="${SLURM_ACCOUNT:-}"
DRY_RUN=0

# Use the current working directory - submit scripts must be run from the
# project directory containing gen_conformers.py. SCRIPT_DIR via $0 fails
# inside SLURM because $0 resolves to the spool copy, not the original path.
SCRIPT_DIR="$(pwd)"
PYTHON_SCRIPT="${SCRIPT_DIR}/gen_conformers.py"

# Parse args
while [[ $# -gt 0 ]]; do
    case "$1" in
        --conformers) CONFORMERS="$2"; shift 2 ;;
        --max_confs) MAX_CONFS="$2"; shift 2 ;;
        --root) ROOT="$2"; shift 2 ;;
        --time) WALL_TIME="$2"; shift 2 ;;
        --mem) MEM="$2"; shift 2 ;;
        --account) ACCOUNT="$2"; shift 2 ;;
        --dry_run) DRY_RUN=1; shift ;;
        *) echo "Unknown option: $1"; exit 1 ;;
    esac
done

echo " Conformer Generation Job Submission"
echo " Conformers : ${CONFORMERS}"
echo " Max confs : ${MAX_CONFS} (OMEGA only)"
echo " Root : ${ROOT}"
echo " Wall time : ${WALL_TIME} per task"
echo " Memory : ${MEM} per task"
echo ""

# Count dataset folders
echo "Counting dataset folders"
N_FOLDERS=$(python3 - <<PYEOF
import sys
from pathlib import Path
import pandas as pd, logging
logging.disable(logging.CRITICAL)

ACT_FILENAME = "ligands.csv"
ID_COL = "molecule_chembl_id"
SKIP_PATTERNS = ("rocs_summary", "oeomega", "_VS_", "shape_matrix",
                 "color_matrix", "combo_matrix", "activity_matrix")

#follows a specific folder structure
def has_activity_file(folder):
    f = folder / ACT_FILENAME
    if not f.exists():
        for ext in ("csv", "xls", "xlsx"):
            for cand in sorted(folder.glob(f"*.{ext}")):
                if any(p in cand.name for p in SKIP_PATTERNS):
                    continue
                try:
                    df_peek = (pd.read_excel(cand, nrows=0)
                               if cand.suffix.lower() in (".xls", ".xlsx")
                               else pd.read_csv(cand, nrows=0))
                    if ID_COL in df_peek.columns:
                        return True
                except Exception:
                    continue
    return False
try:
    df_peek = pd.read_csv(f, nrows=0)
    return ID_COL in df_peek.columns

```

```

        except Exception:
            return False

root = Path("${ROOT}")
folders = sorted(d for d in root.iterdir()
                  if d.is_dir() and has_activity_file(d))
for i, d in enumerate(folders):
    print(f" [{i}] {d.name}", file=sys.stderr)
print(len(folders))
PYEOF
)

if [[ -z "$N_FOLDERS" || "$N_FOLDERS" -lt 1 ]]; then
    echo "ERROR: No dataset folders found under ${ROOT}."
    exit 1
fi

echo " Found ${N_FOLDERS} folder(s)"

MAX_IDX=$(( N_FOLDERS - 1 ))
ACCOUNT_LINE=""
if [[ -n "$ACCOUNT" ]]; then
    ACCOUNT_LINE="#SBATCH --account=${ACCOUNT}"
fi

# Build SLURM script - necessary to make sure confs dont overload the submission queue
SLURM_SCRIPT="#!/bin/bash
#SBATCH --job-name=gen_confs
#SBATCH --output=logs/gen_confs_%A_%a.out
#SBATCH --error=logs/gen_confs_%A_%a.err
#SBATCH --time=${WALL_TIME}
#SBATCH --mem=${MEM}
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
${ACCOUNT_LINE}

module purge
module load python/3.11.8
module load openeye/2022.2.1

export OE_LICENSE=/global/software/alma8/commercial/openeye/2022.2.1/oe_license.txt
export OE_ARCH=redhat-RHEL7-x64 # AlmaLinux 8 nodes: force RHEL7 binary (only arch available)

# OEChem Python bindings (needed for macrocycle single-conformer fallback)
export PYTHONPATH=/global/software/alma8/commercial/openeye/2022.2.1/arch/redhat-RHEL7-
x64/vida/5.0.3.1/lib:${PYTHONPATH:-}
export LD_LIBRARY_PATH=/global/software/alma8/commercial/openeye/2022.2.1/arch/redhat-RHEL7-
x64/vida/5.0.3.1/bin:/global/software/alma8/commercial/openeye/2022.2.1/arch/redhat-RHEL7-
x64/vida/5.0.3.1/lib/openeye/libs/python3-redhat-7-x64-
g++7.4:/global/software/alma8/commercial/openeye/2022.2.1/arch/redhat-RHEL7-
x64/vida/5.0.3.1/lib/openeye/libs:${LD_LIBRARY_PATH:-}
export OMP_NUM_THREADS=1

mkdir -p logs

cd ${SCRIPT_DIR}

echo \"Task \${SLURM_ARRAY_TASK_ID} / \${MAX_IDX} - folder index \${SLURM_ARRAY_TASK_ID} -
\${hostname}\"

python3 ${PYTHON_SCRIPT} \\
    --conformers ${CONFORMERS} \\
    --max_confs ${MAX_CONFS} \\
    --root ${ROOT} \\
    --folder_index \${SLURM_ARRAY_TASK_ID}

echo \"Task \${SLURM_ARRAY_TASK_ID} done\"

SUBMIT_SCRIPT="gen_confs_job.slurm"
echo "$SLURM_SCRIPT" > "$SUBMIT_SCRIPT"

```

```

echo "Step 2: Job script written to ${SUBMIT_SCRIPT}"

if [[ "$DRY_RUN" -eq 1 ]]; then
    echo ""
    echo "DRY RUN - would submit:"
    echo "  sbatch --array=0-${MAX_IDX} ${SUBMIT_SCRIPT}"
    echo ""
    echo "After job completes, verify:"
    echo "  python3 ${PYTHON_SCRIPT} --check --root ${ROOT}"
    echo ""
    echo "Then submit ROCS:"
    echo "  bash submit_rocs_array.sh --root ${ROOT}"
else
    mkdir -p logs
    JOB_ID=$(sbatch --array="0-${MAX_IDX}" "${SUBMIT_SCRIPT}" | awk '{print $NF}')
    echo ""
    echo "Submitted array job ${JOB_ID}  (${N_FOLDERS} tasks)"
    echo ""
    echo "Monitor:  squeue -u \${USER}"
    echo "Logs:     tail -f logs/gen_confs_${JOB_ID}_0.out"
    echo ""
    echo "After all tasks finish:"
    echo "  python3 ${PYTHON_SCRIPT} --check --root ${ROOT}"
    echo "  bash submit_rocs_array.sh --root ${ROOT}"
fi

```

b) Python conformer generation script

```

#!/usr/bin/env python3
"""
gen_conformers.py

Generate per-molecule multi-conformer OEB files from a ChEMBL ligand CSV,
using OpenEye OMEGA (default) or OpenBabel --gen3D (fallback).

Each dataset folder gets a conformers/subdirectory containing one <CHEMBL_ID>.oeb.gz per ligand.

each array task handles one dataset folder.  The task ID is used as a
folder index, so the array size equals the number of dataset folders.

    sbatch --array=0-<N_FOLDERS-1> gen_conformers_job.slurm

After all tasks complete, verify with:
    python gen_conformers.py --check --root .

Sequential (local / interactive) mode options - max_confs gives 100 conformers if the structure
can have that many

    python gen_conformers.py                # all folders, OMEGA 200 confs
    python gen_conformers.py --conformers single # obabel fallback
    python gen_conformers.py --max_confs 100
    python gen_conformers.py --check          # report conformer counts

Options for interactive mode (necessary to check that run is progressing well)

    --conformers  omega or single  Conformer engine (default: omega)
    --max_confs   N                Max conformers per ligand for OMEGA (default: 200)
    --root        DIR              Root containing dataset folders (default: .)
    --folder_index N              Process only folder N (set by SLURM array)
    --check                           Report how many OEBs exist per folder and exit
"""

import os, sys, math, logging, argparse, subprocess, shutil
from pathlib import Path

import pandas as pd
import numpy as np

```

```

try:
    from tqdm import tqdm
except ImportError:
    def tqdm(it, **kw):
        it = list(it); n = len(it)
        for k, v in enumerate(it, 1):
            if n and k % max(1, n // 10) == 0:
                print(f" progress {k}/{n}")
            yield v

# CONFIG
OMEGA_CMD = "omega2" # falls back to "omega" if not found
ROCS_CMD = "rocs" # needed only for --conformers single
ACT_FILENAME = "ligands.csv"
LOG_NAME = "gen_conformers.log"
MIN_OEB_BYTES = 100 # below this a file is an empty gzip shell

# LOGGING
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s %(levelname)s: %(message)s",
    handlers=[logging.FileHandler(LOG_NAME), logging.StreamHandler(sys.stdout)],
)
log = logging.getLogger("gen_conformers")

# ARGS
def parse_args():
    p = argparse.ArgumentParser(
        description="Generate conformer OEB files for ChEMBL ligand sets",
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog=__doc__,
    )
    p.add_argument("--conformers", choices=["omega", "single"], default="omega")
    p.add_argument("--max_confs", type=int, default=200)
    p.add_argument("--root", default=".", help="Root dir containing dataset folders")
    p.add_argument("--folder_index", type=int, default=None,
        help="Process only this folder index (0-based); set by SLURM array")
    p.add_argument("--check", action="store_true",
        help="Report conformer counts per folder and exit")
    return p.parse_args()

# ACTIVITY FILE READER
def read_activity(folder):
    f = folder / ACT_FILENAME
    if not f.exists():
        for ext in ("csv", "xls", "xlsx"):
            cands = list(folder.glob(f"*.{ext}"))
            if cands:
                f = cands[0]; break
    if not f.exists():
        return None
    try:
        return (pd.read_excel(f) if f.suffix.lower() in (".xls", ".xlsx")
            else pd.read_csv(f))
    except Exception as e:
        log.error(f"Cannot read {f}: {e}")
        return None

# CONFORMER ENGINES
def _cmd_exists(cmd):
    return shutil.which(cmd) is not None

def _generate_single_ochem(smiles, lig_id, out_oeb):
    """
    Generate a single 3D conformer using the OEChem Python API.
    Used as a fallback for molecules that omega2 cannot handle (e.g. macrocycles
    since omega_macrocycle is not available for the current architecture).

    Uses OEDepict/OEChem to parse SMILES, then OEOMega in fast single-conf mode
    if available, otherwise OEChem's basic 3D builder (OEFF).
    """

```

```

Writes a single-molecule OEB.gz that ROCS can read.
"""
# Ensure OEChem shared libs and Python bindings are on the path.
# The module system may not set these for the RHEL7 vida install.
# May change for other systems
import sys as _sys
_vida = ("/global/software/alma8/commercial/openeye/2022.2.1"
         "/arch/redhat-RHEL7-x64/vida/5.0.3.1")
_lib = f"{_vida}/lib"
_bin = f"{_vida}/bin"
_so = f"{_lib}/openeye/libs/python3-redhat-7-x64-g++7.4"
if _lib not in _sys.path:
    _sys.path.insert(0, _lib)
for _p in [_bin, _so, f"{_lib}/openeye/libs"]:
    if _p not in os.environ.get("LD_LIBRARY_PATH", ""):
        os.environ["LD_LIBRARY_PATH"] = _p + ":" + os.environ.get("LD_LIBRARY_PATH", "")

try:
    from openeye import oechem, oeomega
except ImportError:
    log.error("OEChem Python API not available - cannot generate single conformer")
    return False

mol = oechem.OEMol()
if not oechem.OESmilesToMol(mol, smiles):
    log.error(f"OEChem could not parse SMILES for {lig_id}: {smiles[:80]}")
    return False

mol.SetTitle(lig_id)
oechem.OEAddExplicitHydrogens(mol)

# Try OEOmega in classic mode with strictstereo off - just want 1 conformer
omega = oeomega.OEOmega()
omega.SetMaxConfs(1)
omega.SetStrictStereo(False)
omega.SetStrictAtomTypes(False)
omega.SetIncludeInput(False)

if not omega(mol):
    # Omega failed even for 1 conformer - try the basic OEChem 3D builder
    builder = oeomega.OEOmega()
    builder.SetMaxConfs(1)
    builder.SetStrictStereo(False)
    builder.SetStrictAtomTypes(False)
    builder.SetBuildForceField(oeomega.OEOmegaForceFieldType_MMFF94S)
    if not builder(mol):
        log.error(f"OEChem OEOmega (1 conf) also failed for {lig_id}")
        return False

ofs = oechem.oemolostream(str(out_oeb))
ofs.SetFormat(oechem.OEFormat_OEB)
oechem.OEWriteMolecule(ofs, mol) # may return False for stereo warnings
ofs.close() # even when the molecule was written - check
# file size instead of return value

if not out_oeb.exists() or out_oeb.stat().st_size < MIN_OEB_BYTES:
    log.error(f"OEChem wrote empty/missing OEB for {lig_id} "
             f"(size={out_oeb.stat().st_size} if out_oeb.exists() else 'missing')")
    return False
return True

def generate_omega(smiles, lig_id, out_oeb, max_confs):
    """
    Multi-conformer OEB via OMEGA.

    omega2 is a shell wrapper that uses pushd/popd internally.
    Never pass cwd= to subprocess - if Python cleans up work_dir before
    the wrapper's popd fires, it crashes with rc=1 on every molecule.
    Use fully-resolved absolute paths for all arguments instead.
    """
    omega = (OMEGA_CMD if _cmd_exists(OMEGA_CMD)

```

```

        else ("omega" if _cmd_exists("omega") else None))
if omega is None:
    log.error("OMEGA not found. Load the openeye module or use --conformers single.")
    return False, False

work_dir = out_oeb.parent / f"_omega_tmp_{lig_id}"
work_dir.mkdir(exist_ok=True)

smi_file = work_dir / f"{lig_id}.smi"
tmp_oeb = work_dir / f"{lig_id}.oeb.gz"
smi_file.write_text(f"{smiles} {lig_id}\n")

cmd = [
    omega,
    "-in",          str(smi_file.resolve()),
    "-out",         str(tmp_oeb.resolve()),
    "-maxconfs",    str(max_confs),
    "-strictstereo", "false",
    "-rms",         "0.5",
    "-warts",       "false",
]

# Pass the full environment so OE_LICENSE and OE_ARCH are visible to the
# omega2/omega_macrocycle shell wrappers. OE_ARCH forces the RHEL7 binary
# on AlmaLinux 8 nodes – the wrapper reads /etc/redhat-release, detects
# RHEL8, then fails to find arch/redhat-RHEL8-x64/ (only RHEL7 installed).
# Always set OE_ARCH explicitly – don't rely on it being inherited, since
# the wrapper script resets PATH=/usr/bin:/bin internally.
import os as _os
env = _os.environ.copy()
env["OE_ARCH"] = "redhat-RHEL7-x64"

try:
    r = subprocess.run(cmd, stdout=subprocess.PIPE,
                      stderr=subprocess.PIPE, timeout=600,
                      env=env)
except subprocess.TimeoutExpired:
    log.error(f"OMEGA timeout for {lig_id}")
    shutil.rmtree(work_dir, ignore_errors=True)
    return False, False
except Exception as e:
    log.error(f"OMEGA exception {lig_id}: {e}")
    shutil.rmtree(work_dir, ignore_errors=True)
    return False, False

if r.returncode != 0:
    stderr = r.stderr.decode(errors="ignore")
    stdout = r.stdout.decode(errors="ignore")
    # Log the full output – the real error is often after the ASCII art header
    log.error(f"OMEGA failed {lig_id} (rc={r.returncode})")
    for line in (stderr + stdout).splitlines():
        line = line.strip()
        if line:
            log.error(f"  omega2: {line}")
    shutil.rmtree(work_dir, ignore_errors=True)
    return False, False

# A valid OEB is always several KB. A 20-byte file is an empty gzip
# shell left behind when OMEGA fails silently. Troubleshooting step.
if not tmp_oeb.exists() or tmp_oeb.stat().st_size < MIN_OEB_BYTES:
    candidates = [f for f in work_dir.glob("*.oeb.gz")
                  if f.stat().st_size >= MIN_OEB_BYTES]
    if candidates:
        tmp_oeb = candidates[0]
        log.warning(f"OMEGA wrote to {tmp_oeb.name} (not expected name) for {lig_id}")
    else:
        # Standard OMEGA failed – retry with omega_macrocycle.
        # Macrocycles with ring-closure trans double bonds (/C=C/ inside
        # a large ring) routinely fail omega2 but succeed with the
        # dedicated macrocycle engine.
        mac = shutil.which("omega_macrocycle")

```

```

        # omega_macrocycle binary not available for this arch - fall back to
        # a single conformer using the OEChem Python API (always available
        # when the openeye module is loaded).
        log.info(f"omega2 failed for {lig_id} - generating single conformer "
                f"via OEChem Python API")
        for f in work_dir.glob("*.oeb.gz"): # remove empty shell
            f.unlink()
        if _generate_single_oechem(smiles, lig_id, tmp_oeb):
            log.info(f"OEChem single conformer succeeded for {lig_id}")
            shutil.move(str(tmp_oeb), str(out_oeb))
            shutil.rmtree(work_dir, ignore_errors=True)
            return True, True # success, used fallback=True
        else:
            size = tmp_oeb.stat().st_size if tmp_oeb.exists() else "missing"
            log.error(f"OEChem single conformer also failed for {lig_id} (size={size})")
            shutil.rmtree(work_dir, ignore_errors=True)
            return False, False

    shutil.move(str(tmp_oeb), str(out_oeb))
    shutil.rmtree(work_dir, ignore_errors=True)
    log.info(f"OMEGA: {lig_id} -> {out_oeb.name}")
    return True, False

def generate_single(smiles, lig_id, out_oeb, workdir):
    """Single conformer via obabel --gen3D, then wrapped into OEB via ROCS."""
    mol2 = workdir / f"{lig_id}.mol2"
    r = subprocess.run(
        ["obabel", f"-:{smiles}", "-O", str(mol2), "--gen3D"],
        stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL, timeout=120,
    )
    if r.returncode != 0 or not mol2.exists():
        log.error(f"obabel failed for {lig_id}")
        return False

    r2 = subprocess.run(
        [ROCS_CMD, "-query", str(mol2), "-dbase", str(mol2),
         "-prefix", str(workdir / lig_id),
         "-oformat", "oeb.gz", "-opt", "-besthits", "1"],
        stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL, timeout=300,
    )
    produced = workdir / f"{lig_id}_hits_1.oeb.gz"
    if not produced.exists():
        produced = workdir / f"{lig_id}.oeb.gz"
    if not produced.exists():
        log.error(f"Single-conformer ROCS produced no OEB for {lig_id}")
        return False
    shutil.copy2(produced, out_oeb)
    log.info(f"Single conformer: {out_oeb.name}")
    return True

def build_oeb(df_act, folder, conformer_mode, max_confs):
    """
    Generate OEB files for all ligands in df_act.
    Skips ligands whose OEB already exists and is valid (>= MIN_OEB_BYTES).
    Deletes and regenerates corrupt/empty files.
    Returns {chembl_id -> Path}.
    """
    conf_dir = folder / "conformers"
    conf_dir.mkdir(exist_ok=True)
    smiles_cols = ["std_smiles", "canonical_smiles", "original_smiles", "smiles"]
    oeb = {}
    fallback_records = []

    rows = list(df_act.iterrows())
    for _, row in tqdm(rows, desc=f"{folder.name} conformers"):
        lig_id = str(row.get("molecule_chembl_id", "")).strip()
        if not lig_id:
            continue

```

```

out_oeb = conf_dir / f"{lig_id}.oeb.gz"

# Resume: valid OEB already present
if out_oeb.exists() and out_oeb.stat().st_size >= MIN_OEB_BYTES:
    oebs[lig_id] = out_oeb
    continue

# Remove corrupt/empty shells from failed previous runs
if out_oeb.exists():
    log.warning(f"Removing corrupt OEB for {lig_id} "
               f"({out_oeb.stat().st_size} bytes), regenerating")
    out_oeb.unlink()

smiles = next(
    (row.get(c, "") for c in smiles_cols
     if isinstance(row.get(c, ""), str) and row.get(c, "").strip()),
    None,
)
if not smiles:
    log.warning(f"No SMILES for {lig_id}")
    continue

if conformer_mode == "omega":
    ok, used_fallback = generate_omega(smiles, lig_id, out_oeb, max_confs)
else:
    ok = generate_single(smiles, lig_id, out_oeb, conf_dir)
    used_fallback = False

if ok:
    oebs[lig_id] = out_oeb
    if used_fallback:
        fallback_records.append({
            "molecule_chembl_id": lig_id,
            "smiles": smiles,
            "folder": folder.name,
            "reason": "omega2 failed; single conformer via OEChem Python API",
        })

log.info(f"{len(oebs)}/{len(rows)} conformer OEBs ready in {conf_dir}")

# Write fallback report if any molecules needed the single-conformer fallback
if fallback_records:
    import csv as _csv
    report_path = conf_dir / "fallback_report.csv"
    write_header = not report_path.exists()
    with open(report_path, "a", newline="") as fh:
        writer = _csv.DictWriter(
            fh,
            fieldnames=["molecule_chembl_id", "smiles", "folder", "reason"],
        )
        if write_header:
            writer.writeheader()
        writer.writerows(fallback_records)
    log.info(f"Fallback report: {report_path} ({len(fallback_records)} entries)")

return oebs

# CHECK MODE - added bc run kept failing
def check_conformers(folders):
    print(f"\n{'Folder':<20} {'Ligands':>8} {'OEBs OK':>8} {'Missing':>8}")
    print("-" * 48)
    total_lig = total_oeb = 0
    for folder in folders:
        df = read_activity(folder)
        if df is None:
            continue
        n_lig = df["molecule_chembl_id"].dropna().nunique()
        conf_dir = folder / "conformers"
        n_oeb = sum(
            1 for f in conf_dir.glob("*.oeb.gz")
            if f.stat().st_size >= MIN_OEB_BYTES

```

```

        ) if conf_dir.exists() else 0
        missing = n_lig - n_oeb
        flag = " <-- incomplete" if missing > 0 else ""
        print(f"{folder.name:<20} {n_lig:>8} {n_oeb:>8} {missing:>8}{flag}")
        total_lig += n_lig
        total_oeb += n_oeb
    print("-" * 48)
    print(f"{'TOTAL':<20} {total_lig:>8} {total_oeb:>8} {total_lig-total_oeb:>8}\n")

# MAIN
def main():
    args = parse_args()
    root = Path(args.root)

    # SLURM array: folder_index from environment
    slurm_id = os.environ.get("SLURM_ARRAY_TASK_ID")
    if slurm_id is not None:
        args.folder_index = int(slurm_id)
        log.info(f"SLURM_ARRAY_TASK_ID={args.folder_index}")

    folders = sorted(
        d for d in root.iterdir()
        if d.is_dir() and read_activity(d) is not None
    )
    if not folders:
        log.error(f"No dataset folders with {ACT_FILENAME} found under {root}")
        sys.exit(1)

    log.info(f"{len(folders)} folder(s): {[f.name for f in folders]}")

    # --check: just report counts and exit
    if args.check:
        check_conformers(folders)
        return

    # SLURM array: one folder per task
    if args.folder_index is not None:
        if args.folder_index >= len(folders):
            log.error(f"folder_index {args.folder_index} out of range "
                      f"(only {len(folders)} folders)")
            sys.exit(1)
        folder = folders[args.folder_index]
        log.info(f"Processing folder [{args.folder_index}]: {folder.name}")
        df_act = read_activity(folder)
        build_oebps(df_act, folder, args.conformers, args.max_confs)
        return

    # Sequential: all folders
    for folder in folders:
        df_act = read_activity(folder)
        if df_act is None:
            continue
        log.info(f"--- {folder.name} ---")
        build_oebps(df_act, folder, args.conformers, args.max_confs)

    log.info("All conformers done. Run check with: python gen_conformers.py --check")
    log.info("Then submit ROCS with: bash submit_rocs_array.sh")

if __name__ == "__main__":
    main()

```

7.5.3 ChEMBL Ligand Random Dataset ROCS Run

b) Bash submission script

```

#!/bin/bash
# submit_rocs_array.sh
#
# Counts total pairs across all dataset folders, then submits a SLURM array

```

```

# job for run_rocs.py.
#
# Run gen_conformers.py (via submit_conformers.sh) FIRST and verify with
# python gen_conformers.py --check
# before submitting this.
#
# After all tasks complete, assemble matrices and plots:
# python run_rocs.py --collect --root <ROOT>
#
# Options for running
# --root          DIR          Dataset root directory (default: .)
# --time          HH:MM:SS     Wall time per task (default: 02:00:00)
# --mem           MEM          Memory per task (default: 8G)
# --account       ACCOUNT      SLURM account
# --max_array     N            Override max array size (default: auto-detect)
# --max_running   N            Max simultaneously running tasks (default: 200)
# --dry_run       Print commands without submitting - test that it works!

set -euo pipefail

# Defaults
ROOT="."
WALL_TIME="02:00:00"
MEM="8G"
ACCOUNT="${SLURM_ACCOUNT:-}"
MAX_ARRAY_OVERRIDE=""
MAX_RUNNING=200      # max simultaneously running tasks (throttle to avoid overloading)
DRY_RUN=0
SCRIPT_DIR="$(pwd)"
PYTHON_SCRIPT="${SCRIPT_DIR}/run_rocs.py"

# Parse args
while [[ $# -gt 0 ]]; do
    case "$1" in
        --root)          ROOT="$2";                shift 2 ;;
        --time)          WALL_TIME="$2";            shift 2 ;;
        --mem)           MEM="$2";                  shift 2 ;;
        --account)       ACCOUNT="$2";              shift 2 ;;
        --max_array)     MAX_ARRAY_OVERRIDE="$2";    shift 2 ;;
        --max_running)   MAX_RUNNING="$2";          shift 2 ;;
        --dry_run)       DRY_RUN=1;                 shift ;;
        *) echo "Unknown option: $1"; exit 1 ;;
    esac
done

echo " ROCS Array Job Submission"
echo " Root       : ${ROOT}"
echo " Wall time   : ${WALL_TIME} per task"
echo " Memory       : ${MEM} per task"
echo ""

# Detect cluster MaxArraySize
if [[ -n "$MAX_ARRAY_OVERRIDE" ]]; then
    MAX_ARRAY_SIZE="$MAX_ARRAY_OVERRIDE"
    echo " Max array size : ${MAX_ARRAY_SIZE} (manual override)"
else
    MAX_ARRAY_SIZE=$(scontrol show config 2>/dev/null \
        | grep -i "MaxArraySize" \
        | awk -F= '{print $2}' \
        | tr -d ' ' \
        | head -1)
    if [[ -z "$MAX_ARRAY_SIZE" || "$MAX_ARRAY_SIZE" -lt 1 ]]; then
        MAX_ARRAY_SIZE=1000
        echo " Max array size : ${MAX_ARRAY_SIZE} (could not detect, using default)"
    else
        echo " Max array size : ${MAX_ARRAY_SIZE} (detected from cluster)"
    fi
fi

# Count total pairs
echo ""

```

```

echo "Counting pairs from existing conformers"
N_PAIRS=$(python3 - <<PYEOF
import sys
from pathlib import Path
from itertools import combinations
import pandas as pd, logging
logging.disable(logging.CRITICAL)

ACT_FILENAME = "ligands.csv"
ID_COL = "molecule_chembl_id"
MIN_OEB_BYTES = 100

# Non-activity CSVs that must never be picked up as the ligand file
SKIP_PATTERNS = ("rocs_summary", "oeomega", "_VS_", "shape_matrix",
                  "color_matrix", "combo_matrix", "activity_matrix")

def read_act(folder):
    f = folder / ACT_FILENAME
    if not f.exists():
        for ext in ("csv", "xls", "xlsx"):
            for cand in sorted(folder.glob(f"*.{ext}")):
                if any(p in cand.name for p in SKIP_PATTERNS):
                    continue
                try:
                    df_peek = (pd.read_excel(cand, nrows=0)
                               if cand.suffix.lower() in (".xls", ".xlsx")
                               else pd.read_csv(cand, nrows=0))
                    if ID_COL in df_peek.columns:
                        f = cand
                        break
                except Exception:
                    continue
    if not f.exists():
        return None
    try:
        df = (pd.read_excel(f) if f.suffix.lower() in (".xls", ".xlsx")
              else pd.read_csv(f))
        if ID_COL not in df.columns:
            print(f" SKIP {folder.name}: no '{ID_COL}' column in {f.name}",
                  file=sys.stderr)
            return None
        return df
    except Exception as e:
        print(f" SKIP {folder.name}: cannot read {f.name}: {e}", file=sys.stderr)
        return None

root = Path("${ROOT}")
total = 0
for d in sorted(root.iterdir()):
    if not d.is_dir():
        continue
    df = read_act(d)
    if df is None:
        continue
    # Count only ligands that actually have a valid OEB
    conf_dir = d / "conformers"
    if conf_dir.exists():
        oeb_ids = {
            f.stem.replace(".oeb", "")
            for f in conf_dir.glob("*.oeb.gz")
            if f.stat().st_size >= MIN_OEB_BYTES
        }
    else:
        oeb_ids = set()
    all_ids = set(df[ID_COL].dropna().unique())
    ids = sorted(all_ids & oeb_ids) # only ligands with valid OEB
    missing = len(all_ids) - len(ids)
    n = len(list(combinations(ids, 2)))
    print(f" {d.name}: {len(ids)} ligands with OEBs "
          f"({missing} missing), {n} pairs", file=sys.stderr)
    total += n

```

```

print(total)
PYEOF
)

if [[ -z "$N_PAIRS" || "$N_PAIRS" -lt 1 ]]; then
    echo "ERROR: No pairs found. Have you run submit_conformers.sh and verified with"
    echo "      python gen_conformers.py --check ?"
    exit 1
fi

echo "  Total pairs : ${N_PAIRS}"

# Work out chunking
if [[ "$N_PAIRS" -le "$MAX_ARRAY_SIZE" ]]; then
    N_TASKS="$N_PAIRS"
    CHUNK_SIZE=1
else
    N_TASKS="$MAX_ARRAY_SIZE"
    CHUNK_SIZE=$(( (N_PAIRS + N_TASKS - 1) / N_TASKS ))
fi

MAX_IDX=$(( N_TASKS - 1 ))
echo "  Tasks      : ${N_TASKS} (chunk size: ${CHUNK_SIZE} pairs/task)"
echo "  Array range : 0-${MAX_IDX}"
echo ""

# Build SLURM script
ACCOUNT_LINE=""
if [[ -n "$ACCOUNT" ]]; then
    ACCOUNT_LINE="#SBATCH --account=${ACCOUNT}"
fi

SLURM_SCRIPT="#!/bin/bash
#SBATCH --job-name=rocs_pairs
#SBATCH --output=logs/rocs_%A.out
#SBATCH --error=logs/rocs_%A.err
#SBATCH --time=${WALL_TIME}
#SBATCH --mem=${MEM}
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
${ACCOUNT_LINE}

module purge
module load python/3.11.8
module load openeye/2022.2.1

export OE_LICENSE=/global/software/alma8/commercial/openeye/2022.2.1/oe_license.txt
export OE_ARCH=redhat-RHEL7-x64
export PATH=/global/software/alma8/commercial/openeye/2022.2.1/bin:${PATH}
export OMP_NUM_THREADS=1

mkdir -p logs

cd ${SCRIPT_DIR}

FIRST_PAIR=$(( SLURM_ARRAY_TASK_ID * ${CHUNK_SIZE} ))
echo "\"Task \${SLURM_ARRAY_TASK_ID} / \${MAX_IDX} - pairs \${FIRST_PAIR} to \${( FIRST_PAIR +
\${CHUNK_SIZE} - 1 )) - \${(hostname)}\""

python3 ${PYTHON_SCRIPT} \\\
    --root      \${ROOT}      \\\
    --pair_index \${FIRST_PAIR} \\\
    --chunk_size \${CHUNK_SIZE}

echo "\"Task \${SLURM_ARRAY_TASK_ID} done\""

SUBMIT_SCRIPT="rocs_array_job.slurm"
echo "$SLURM_SCRIPT" > "$SUBMIT_SCRIPT"
echo "Step 2: Job script written to ${SUBMIT_SCRIPT}"

```

```

if [[ "$DRY_RUN" -eq 1 ]]; then
    echo ""
    echo "DRY RUN - would submit:"
    echo "  sbatch --array=0-${MAX_IDX}%${MAX_RUNNING} ${SUBMIT_SCRIPT}"
    echo ""
    echo "After job completes, run:"
    echo "  python3 ${PYTHON_SCRIPT} --collect --root ${ROOT}"
else
    mkdir -p logs
    JOB_ID=$(sbatch --array="0-${MAX_IDX}%${MAX_RUNNING}" "${SUBMIT_SCRIPT}" | awk '{print $NF}')
    echo ""
    echo "Submitted array job ${JOB_ID}"
    echo "  ${N_TASKS} tasks × ${CHUNK_SIZE} pairs/task = ${N_PAIRS} total pairs"
    echo ""
    echo "Monitor:    squeue -u \${USER}"
    echo "Logs:       tail -f logs/rocs_${JOB_ID}_0.out"
    echo ""
    echo "After all tasks finish:"
    echo "  python3 ${PYTHON_SCRIPT} --collect --root ${ROOT}"
fi

```

b) Python conformer generation script

```

#!/usr/bin/env python3
"""
run_rocs.py

Step 2 of the ChEMBL ligand similarity pipeline.

Assumes conformer OEB files already exist in <dataset>/conformers/,
generated by gen_conformers.py. Run that script first and verify
with --check before submitting this one.

What this script does
For each ligand pair (id1, id2), it generates a single-conformer self-ROCS
OEB for id1 (cached per ligand). This is the EON -query input, needed to run
EON on these ligand pairs.
It also runs ROCS with -report each, using the query OEB as -query and the
full conformer ensemble of id2 as -dbase. This finds the true best overlay
across all conformers of id2 rather than already having 3D coordinates.
It then saves the best-scoring aligned hit OEB: pair_results/<id1>_VS_<id2>_best.oeb.gz
This is the EON -dbase input and also the ROCS file that's read. It writes the ROCS
scores to pair_results/<id1>_VS_<id2>.csv (shape / color / combo Tanimoto)
All intermediate ROCS output - up to 200 RPTs + 200 OEBs per pair -
is written to a per-pair tmpdir and deleted immediately to save space.

Permanent files per pair
pair_results/<id1>_VS_<id2>.csv          ~200 bytes (scores)
pair_results/<id1>_VS_<id2>_best.oeb.gz  ~10-50 KB (aligned hit; EON -dbase)

At N=100 ligands (4950 pairs) the permanent file count is:
4950 CSVs + 4950 best OEBs + 100 query OEBs = 10,000 files.

SLURM array mode
Use submit_rocs_array.sh to count pairs, compute chunking, and submit.
After all array tasks finish:
python run_rocs.py --collect --root .
python run_eon_morgan.py --root .

Options for running:
--pair_index N      First pair index (set by SLURM array)
--chunk_size N      Pairs per SLURM task (default: 1)
--collect           Assemble pair_results/ into matrices + scatter plots
--root DIR          Root directory containing dataset folders (default: .)
"""

import os, sys, math, logging, argparse, subprocess, shutil
from pathlib import Path
from itertools import combinations

import pandas as pd

```

```

import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

try:
    from tqdm import tqdm
except ImportError:
    def tqdm(it, **kw):
        it = list(it); n = len(it)
        for k, v in enumerate(it, 1):
            if n and k % max(1, n // 10) == 0:
                print(f" progress {k}/{n}")
            yield v

# CONFIG
ROCS_CMD = "rocs"
ACT_FILENAME = "ligands.csv"
LOG_NAME = "rocs_error_log.txt"
MIN_OEB_BYTES = 100
PREFER_ACTIVITY = ["pIC50_calc", "pKi_calc", "pchembl_value"]
UNIT_FACTORS = {"m": 1.0, "mm": 1e-3, "um": 1e-6, "\u00b5m": 1e-6,
                 "nm": 1e-9, "pm": 1e-12}
SKIP_PATTERNS = ("rocs_summary", "oeomega", "VS_", "shape_matrix",
                  "color_matrix", "combo_matrix", "activity_matrix",
                  "eon_matrix", "morgan_matrix")

# LOGGING
logging.basicConfig(
    level=logging.INFO,
    format="%asctime)s %(levelname)s: %(message)s",
    handlers=[logging.FileHandler(LOG_NAME), logging.StreamHandler(sys.stdout)],
)
log = logging.getLogger("run_rocs")

# ARGS
def parse_args():
    p = argparse.ArgumentParser(
        description="Pairwise ROCS for ChEMBL ligand sets",
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog=__doc__,
    )
    p.add_argument("--pair_index", type=int, default=None,
                    help="First pair index (set by SLURM array)")
    p.add_argument("--chunk_size", type=int, default=1,
                    help="Pairs per SLURM task (default: 1)")
    p.add_argument("--collect", action="store_true",
                    help="Assemble pair results/ into matrices + scatter plots")
    p.add_argument("--root", default=".",
                    help="Root directory containing dataset folders (default: .)")
    return p.parse_args()

# ACTIVITY
def to_p(value, units):
    try:
        v = float(value)
    except Exception:
        return np.nan
    if v <= 0:
        return np.nan
    if not isinstance(units, str) or not units.strip():
        return np.nan
    key = units.strip().lower().replace("\u03bc", "u")
    for k, f in UNIT_FACTORS.items():
        if key == k or key.endswith(k):
            m = v * f
            return -math.log10(m) if m > 0 else np.nan
    return np.nan

def compute_pactivity(df):

```

```

for col in PREFER_ACTIVITY:
    if col in df.columns:
        df = df.copy()
        df[col] = pd.to_numeric(df[col], errors="coerce")
        best = (df.sort_values(col, ascending=False, na_position="last")
                 .groupby("molecule_chembl_id", as_index=False).first())
        vals = best.set_index("molecule_chembl_id")[col].to_dict()
        if any(np.isfinite(v) for v in vals.values()):
            return vals, best
df = df.copy()
df["_p"] = df.apply(
    lambda r: to_p(r.get("standard_value"), r.get("standard_units")), axis=1
)
best = (df.sort_values("_p", ascending=False, na_position="last")
        .groupby("molecule_chembl_id", as_index=False).first())
return best.set_index("molecule_chembl_id")["_p"].to_dict(), best

def read_activity(folder):
    f = folder / ACT_FILENAME
    if not f.exists():
        for ext in ("csv", "xls", "xlsx"):
            for cand in sorted(folder.glob(f"*.{ext}")):
                if any(p in cand.name for p in SKIP_PATTERNS):
                    continue
                try:
                    peek = (pd.read_excel(cand, nrows=0)
                            if cand.suffix.lower() in (".xls", ".xlsx")
                            else pd.read_csv(cand, nrows=0))
                    if "molecule_chembl_id" in peek.columns:
                        f = cand
                        break
                except Exception:
                    continue
    if not f.exists():
        return None
    try:
        return (pd.read_excel(f) if f.suffix.lower() in (".xls", ".xlsx")
                else pd.read_csv(f))
    except Exception as e:
        log.error(f"Cannot read {f}: {e}")
        return None

# OEB MAP
def load_oeb_map(folder):
    conf_dir = folder / "conformers"
    if not conf_dir.exists():
        log.error(f"{folder.name}: conformers/ not found - run gen_conformers.py first")
        return {}
    oebs = {
        f.stem.replace(".oeb", ""): f
        for f in conf_dir.glob("*.oeb.gz")
        if f.stat().st_size >= MIN_OEB_BYTES
    }
    if not oebs:
        log.error(f"{folder.name}: no valid OEB files in {conf_dir}")
    else:
        log.info(f"{folder.name}: {len(oebs)} conformer OEBs loaded")
    return oebs

# EXCLUSIONS
def load_exclusion_lists(folder):
    """
    Returns (excluded_3d, excluded_all).
    excluded_3d - excluded from ROCS/EON (single-conformer macrocycles);
                  still receive Morgan fingerprints in run_eon_morgan.py.
    excluded_all - excluded from everything including Morgan.
    """
    excluded_3d = set()
    excluded_all = set()
    fallback = folder / "conformers" / "fallback_report.csv"

```

```

if fallback.exists():
    try:
        ids = pd.read_csv(fallback)["molecule_chembl_id"].dropna().tolist()
        excluded_3d.update(ids)
        if ids:
            log.info(f"{folder.name}: {len(ids)} macrocycle fallbacks "
                    f"excluded from 3D analysis")
    except Exception as e:
        log.warning(f"Could not read fallback_report.csv: {e}")
excl_3d = folder / "conformers" / "excluded_ligands.txt"
if excl_3d.exists():
    ids = [l.strip() for l in excl_3d.read_text().splitlines()
           if l.strip() and not l.startswith("#")]
    excluded_3d.update(ids)
    if ids:
        log.info(f"{folder.name}: {len(ids)} manual 3D exclusions")
excl_all = folder / "conformers" / "excluded_all.txt"
if excl_all.exists():
    ids = [l.strip() for l in excl_all.read_text().splitlines()
           if l.strip() and not l.startswith("#")]
    excluded_all.update(ids)
    if ids:
        log.info(f"{folder.name}: {len(ids)} full exclusions")
return excluded_3d, excluded_all

# ROCS PAIR RUN
def _parse_rpt(rpt_path):
    """
    Parse a ROCS RPT file.

    Returns (shape, color, combo, db_conf_idx) or None.
    db_conf_idx - id2 conformer index, extracted from the Name field.
                  ROCS writes the hit as "<MolTitle> <ConfIdx>" in the
                  Name column, e.g. "CHEMBL1081104_45" db_conf_idx="45"

    RPT column order (from empirical header):
    Name ShapeQuery Rank TanimotoCombo ShapeTanimoto ColorTanimoto ...
    0      1      2      3          4          5
    """
    try:
        lines = rpt_path.read_text().splitlines()
        header = next(
            (l.split() for l in lines if "shapetanimoto" in l.lower()), None
        )
        if header:
            col = {h.lower(): i for i, h in enumerate(header)}
            for line in lines[1:]:
                parts = line.split()
                if not parts:
                    continue
                try:
                    shape = float(parts[col.get("shapetanimoto", 4)])
                    color = float(parts[col.get("colortanimoto", 5)])
                    combo = float(parts[col.get("tanimotocombo", 3)])
                    # Name field is "<MolTitle> <ConfIdx>", e.g. "CHEMBL1081104_45"
                    # ConfIdx is 0-based in ROCS output (single-conformer molecules
                    # get index 0, multi-conformer also start from 0)
                    db_conf_idx = None
                    name_field = parts[col.get("name", 0)]
                    if "_" in name_field:
                        suffix = name_field.rsplit("_", 1)[-1]
                        if suffix.isdigit():
                            db_conf_idx = suffix
                    return shape, color, combo, db_conf_idx
                except (IndexError, ValueError):
                    continue
            # Fallback: no header - parse positionally using observed column order
            for line in lines[1:]:
                parts = line.split()
                if len(parts) >= 6:

```

```

        try:
            # col 0 = Name, col 3 = TanimotoCombo, col 4 = Shape, col 5 = Color
            combo = float(parts[3])
            shape = float(parts[4])
            color = float(parts[5])
            db_conf_idx = None
            if "_" in parts[0]:
                suffix = parts[0].rsplit("_", 1)[-1]
                if suffix.isdigit():
                    db_conf_idx = suffix
            return shape, color, combo, db_conf_idx
        except ValueError:
            continue
    except Exception as e:
        log.error(f"RPT parse error {rpt_path}: {e}")
    return None

def run_rocs_pair(id1, id2, oeb1, oeb2, pair_dir):
    """
    Run ROCS for one ligand pair.

    Query = full OMEGA conformer ensemble of id1 (conformers/<id1>.oeb.gz)
    Dbase = full OMEGA conformer ensemble of id2 (conformers/<id2>.oeb.gz)
    ROCS tries all 200x200 conformer overlays and retains the global best.

    Permanently keeps pair_dir/<id1>_VS_<id2>.csv          shape/color/combo scores
    and pair_dir/<id1>_VS_<id2>_best.oeb.gz      best-scoring aligned id2 pose

    Deletes all intermediate ROCS output (RPTs, remaining OEBs) via a tmpdir.

    Returns (shape, color, combo) or None.
    """
    tag = f"{id1}_VS_{id2}"
    result_csv = pair_dir / f"{tag}.csv"
    best_oeb = pair_dir / f"{tag}_best.oeb.gz"

    # Resume: already done
    if result_csv.exists():
        try:
            row = pd.read_csv(result_csv).iloc[0]
            return float(row["shape"]), float(row["color"]), float(row["combo"])
        except Exception:
            pass

    # Per-pair tmpdir: all ROCS output lands here
    import tempfile as _tf
    tmp_dir = Path(_tf.mkdtemp(prefix=f"rocs_{tag}_", dir=pair_dir))

    cmd = [
        ROCS_CMD,
        "-query",      str(oeb1),      # full OMEGA conformer ensemble of id1 (200 confs)
        "-dbase",      str(oeb2),      # full OMEGA conformer ensemble of id2 (200 confs)
        "-prefix",     str(tmp_dir / tag),
        "-besthits",   "1",
        "-rankby",     "TanimotoCombo",
        "-shapeonly",  "false",
        "-oformat",    "oeb.gz",
        "-report",     "each",
    ]

    env = os.environ.copy()
    env["OE_ARCH"] = "redhat-RHEL7-x64"

    try:
        r = subprocess.run(cmd, stdout=subprocess.DEVNULL,
                           stderr=subprocess.PIPE, timeout=1800, env=env)
    except Exception as e:
        log.error(f"ROCS exception {tag}: {e}")
        shutil.rmtree(tmp_dir, ignore_errors=True)
        return None

```

```

if r.returncode != 0:
    log.error(f"ROCS failed {tag}: {r.stderr.decode(errors='ignore')[:300]}")
    shutil.rmtree(tmp_dir, ignore_errors=True)
    return None

# Collect RPT files
all_rpts = sorted(tmp_dir.glob(f"{tag}*.rpt"))
if not all_rpts:
    single = tmp_dir / f"{tag}.rpt"
    if single.exists():
        all_rpts = [single]
if not all_rpts:
    log.error(f"No RPT files for {tag}")
    shutil.rmtree(tmp_dir, ignore_errors=True)
    return None

# Log the full RPT inventory so we can audit which conformers were evaluated
log.info(f"RPT INVENTORY | {tag} | {len(all_rpts)} RPT(s): "
        f"[{r.name for r in all_rpts}]")

# Score every RPT - track both query (id1) and db (id2) conformer indices
best_scores = None
best_conf_idx = None # id1 winning conformer (from RPT filename)
best_db_conf_idx = None # id2 winning conformer (from DbConformerIndex column)
rpt_scores = {} # query_conf_idx -> combo score for summary log
for rpt in all_rpts:
    result = _parse_rpt(rpt)
    stem = rpt.stem.replace(tag + "_", "")
    conf_idx = stem if stem.isdigit() else rpt.stem
    if result is None:
        log.warning(f"RPT PARSE FAIL | {tag} | {rpt.name}")
        rpt_scores[conf_idx] = None
        continue
    shape, color, combo, db_conf_idx = result
    rpt_scores[conf_idx] = combo
    log.debug(f"RPT SCORE | {tag} | id1_conf={conf_idx} id2_conf={db_conf_idx} | "
            f"shape={shape:.4f} color={color:.4f} combo={combo:.4f}")
    if best_scores is None or combo > best_scores[2]:
        best_scores = (shape, color, combo)
        best_conf_idx = stem if stem.isdigit() else None
        best_db_conf_idx = db_conf_idx

# Summarise at INFO level
if rpt_scores:
    scored = {k: v for k, v in rpt_scores.items() if v is not None}
    if scored:
        ranking = sorted(scored.items(), key=lambda x: x[1], reverse=True)
        top3 = ", ".join(f"conf{k}={v:.4f}" for k, v in ranking[:3])
        log.info(f"RPT SELECTION | {tag} | "
                f"winner: id1_conf={best_conf_idx} id2_conf={best_db_conf_idx} "
                f"combo={best_scores[2]:.4f} | top3=[{top3}] "
                f"| {len(scored)}/{len(all_rpts)} RPTs parsed")

if best_scores is None:
    log.warning(f"ROCS NO SCORE | {tag}")
    shutil.rmtree(tmp_dir, ignore_errors=True)
    return None

# Write permanent score CSV with both winning conformer indices
# id1_conf_idx: which of id1's 200 conformers was the query that won
# id2_conf_idx: which of id2's 200 conformers was the dbase hit that won
# Together these identify the exact pair of structures in _best.oeb.gz.
pd.DataFrame([
    {
        "id1": id1,
        "id2": id2,
        "shape": best_scores[0],
        "color": best_scores[1],
        "combo": best_scores[2],
        "id1_conf_idx": best_conf_idx if best_conf_idx is not None else "",
        "id2_conf_idx": best_db_conf_idx if best_db_conf_idx is not None else "",
    }
]).to_csv(result_csv, index=False)

```

```

log.info(f"ROCS OK | {tag} | shape={best_scores[0]:.3f} "
        f"color={best_scores[1]:.3f} combo={best_scores[2]:.3f} "
        f"| id1_conf={best_conf_idx} id2_conf={best_db_conf_idx}")

# Locate the best-hit OEB – log every candidate found so we can verify
# the right file is selected and not just whichever is alphabetically first
all_hit_oeps = sorted(tmp_dir.glob(f"{tag}_hits_*.oeb.gz"))
log.info(f"OEB INVENTORY | {tag} | {len(all_hit_oeps)} hit OEB(s): "
        f"{[o.name for o in all_hit_oeps]}")

src_oeb = None
if best_conf_idx:
    expected = tmp_dir / f"{tag}_hits_{best_conf_idx}.oeb.gz"
    if expected.exists():
        src_oeb = expected
        log.info(f"OEB MATCH | {tag} | best conf={best_conf_idx} → "
                f"{expected.name} ({expected.stat().st_size} bytes) ✓")
    else:
        log.warning(f"OEB MISMATCH | {tag} | expected {expected.name} "
                    f"(conf={best_conf_idx}) but file not found – "
                    f"available: {[o.name for o in all_hit_oeps]}")

if src_oeb is None and all_hit_oeps:
    src_oeb = all_hit_oeps[0]
    log.warning(f"OEB FALLBACK | {tag} | best-conf OEB missing, "
                f"using first available: {src_oeb.name} "
                f"({src_oeb.stat().st_size} bytes) – "
                f"this OEB may NOT correspond to the best-scoring conformer")

if src_oeb is not None:
    shutil.move(str(src_oeb), str(best_oeb))
    log.info(f"OEB SAVED | {tag} | → {best_oeb.name} "
            f"({best_oeb.stat().st_size} bytes)")
else:
    log.warning(f"OEB MISSING | {tag} | no aligned OEB found – EON will be skipped")

# Delete tmpdir – unlink files individually first, then remove dirs.
import time as _time
n_tmp = 0
n_failed = 0
if tmp_dir.exists():
    # First pass: delete all files individually
    for f in sorted(tmp_dir.rglob("*"), reverse=True):
        if f.is_file():
            n_tmp += 1
            try:
                f.unlink()
            except Exception:
                n_failed += 1
    # Second pass: remove empty subdirs bottom-up, then the tmpdir itself
    for _attempt in range(3):
        try:
            shutil.rmtree(tmp_dir, ignore_errors=True)
        except Exception:
            pass
        if not tmp_dir.exists():
            break
        _time.sleep(0.3)

log.info(f"CLEANUP START | {tag} | {n_tmp} files in {tmp_dir.name}")
if tmp_dir.exists():
    survivors = [f for f in tmp_dir.rglob("*") if f.is_file()]
    log.warning(f"CLEANUP INCOMPLETE | {tag} | {tmp_dir.name} still exists "
                f"with {len(survivors)} file(s) – failed_unlinks={n_failed}: "
                f"{[f.name for f in survivors[:10]]}"
                + (" ..." if len(survivors) > 10 else ""))
else:
    log.info(f"CLEANUP OK | {tag} | {tmp_dir.name} deleted ({n_tmp} files removed)")

return best_scores

```

```

# FOLDER SETUP
def setup_folder(folder):
    df_act = read_activity(folder)
    if df_act is None:
        return None
    oeb_map = load_oeb_map(folder)
    if len(oeb_map) < 2:
        log.error(f"Not enough OEBs for {folder.name} - run gen_conformers.py first")
        return None
    excluded_3d, excluded_all = load_exclusion_lists(folder)
    excluded = excluded_3d | excluded_all
    if excluded:
        before = len(oeb_map)
        oeb_map = {k: v for k, v in oeb_map.items() if k not in excluded}
        log.info(f"{folder.name}: excluded {before - len(oeb_map)} ligands, "
                f"{len(oeb_map)} remaining for ROCS")
    if len(oeb_map) < 2:
        log.error(f"Not enough OEBs after exclusions for {folder.name}")
        return None
    pact_map, _ = compute_pactivity(df_act)
    names = sorted(oeb_map.keys())
    pairs = list(combinations(names, 2))
    return names, pairs, oeb_map, pact_map

# COLLECT / FINALISE
def collect_matrices(names, pair_dir):
    shape = pd.DataFrame(np.nan, index=names, columns=names)
    color = pd.DataFrame(np.nan, index=names, columns=names)
    combo = pd.DataFrame(np.nan, index=names, columns=names)
    for csv_path in pair_dir.glob("*_VS_*.csv"):
        if "_eon" in csv_path.name:
            continue
        try:
            row = pd.read_csv(csv_path).iloc[0]
            a, b = str(row["id1"]), str(row["id2"])
            if a not in names or b not in names:
                continue
            for df, k in [(shape, "shape"), (color, "color"), (combo, "combo")]:
                v = float(row[k])
                df.at[a, b] = v
                df.at[b, a] = v
        except Exception:
            continue
    for n in names:
        shape.at[n, n] = color.at[n, n] = combo.at[n, n] = 1.0
    return shape, color, combo

def save_matrices(folder, shape, color, combo, act):
    """Write each matrix as a CSV and all four as sheets in one workbook."""
    sheets = [
        (shape, "Shape", "shape_matrix.csv"),
        (color, "Color", "color_matrix.csv"),
        (combo, "Combo", "combo_matrix.csv"),
        (act, "Activity", "activity_matrix.csv"),
    ]
    for df, _, csv_name in sheets:
        df.to_csv(folder / csv_name)

    wb_path = folder / "rocs_matrices.xlsx"
    with pd.ExcelWriter(wb_path, engine="openpyxl") as writer:
        for df, sheet_name, _ in sheets:
            df.to_excel(writer, sheet_name=sheet_name)
    log.info(f"Workbook written: {wb_path.name}")

def make_scatter(metric_df, act_mat, label, names, folder):
    xs, ys = [], []
    for i, a in enumerate(names):
        for b in names[i + 1:]:

```

```

        mv = metric_df.at[a, b] if a in metric_df.index and b in metric_df.columns else
np.nan
        av = act_mat.at[a, b] if a in act_mat.index and b in act_mat.columns else
np.nan
        if pd.notna(mv) and pd.notna(av):
            xs.append(1.0 - float(mv))
            ys.append(float(av))
    if not xs:
        return
    plt.figure(figsize=(6, 5))
    plt.scatter(xs, ys, s=10, alpha=0.6)
    plt.xlabel(f"ROCS distance (1 \u2013 {label})")
    plt.ylabel(f"|activity| (pX)")
    plt.title(f"{folder.name}: {label}")
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig(folder / f"scatter_{label}_vs_activity.png", dpi=200)
    plt.close()

def finalise(folder, names, pair_dir, pact_map):
    shape, color, combo = collect_matrices(names, pair_dir)
    act = pd.DataFrame(np.nan, index=names, columns=names)
    for a in names:
        for b in names:
            pa, pb = pact_map.get(a, np.nan), pact_map.get(b, np.nan)
            if np.isfinite(pa) and np.isfinite(pb):
                act.at[a, b] = abs(pa - pb)
    save_matrices(folder, shape, color, combo, act)
    for df, label in [(shape, "Shape"), (color, "Color"), (combo, "Combo")]:
        make_scatter(df, act, label, names, folder)
    n_pairs = len(list(combinations(names, 2)))
    best_oeps = len(list(pair_dir.glob("*_VS_*_best.oeb.gz")))
    pd.DataFrame([
        {"n_ligands": len(names),
         "n_pairs": n_pairs,
         "n_valid_activity": sum(1 for n in names if np.isfinite(pact_map.get(n, np.nan))),
         "n_best_oeb": best_oeps,
        }]).to_csv(folder / "rocs_summary.csv", index=False)
    log.info(f"Finalised {folder.name}: {n_pairs} pairs, {best_oeps} _best.oeb.gz")

# MAIN
def main():
    args = parse_args()
    root = Path(args.root)

    slurm_id = os.environ.get("SLURM_ARRAY_TASK_ID")
    if slurm_id is not None:
        log.info(f"SLURM_ARRAY_TASK_ID={slurm_id} pair_index={args.pair_index}")

    folders = [d for d in sorted(root.iterdir())
                if d.is_dir() and read_activity(d) is not None]
    if not folders:
        log.error(f"No dataset folders with {ACT_FILENAME} under {root}")
        sys.exit(1)
    log.info(f"{len(folders)} folder(s): {[f.name for f in folders]}")

    # collect mode
    if args.collect:
        log.info("Collect mode: assembling ROCS results from pair_results/...")
        for folder in folders:
            df_act = read_activity(folder)
            if df_act is None:
                continue
            pact_map, _ = compute_pactivity(df_act)
            excluded_3d, excluded_all = load_exclusion_lists(folder)
            excluded = excluded_3d | excluded_all
            names = sorted(
                str(r["molecule_chembl_id"])
                for _, r in df_act.iterrows()
                if (pd.notna(r.get("molecule_chembl_id", ""))

```

```

        and str(r["molecule_chembl_id"]) not in excluded)
    )
    pair_dir = folder / "pair_results"
    if not pair_dir.exists():
        log.warning(f"No pair_results/ in {folder.name}")
        continue
    finalise(folder, names, pair_dir, pact_map)
log.info("ROCS collection done.")
log.info("Next: python run_eon_morgan.py --root " + str(root))
return

# SLURM array task
if args.pair_index is not None:
    all_pairs_flat = []
    for folder in folders:
        res = setup_folder(folder)
        if res is None:
            continue
        names, pairs, oeb_map, _ = res
        pair_dir = folder / "pair_results"
        pair_dir.mkdir(exist_ok=True)
        for id1, id2 in pairs:
            all_pairs_flat.append(
                (pair_dir, id1, id2, oeb_map[id1], oeb_map[id2])
            )
    total = len(all_pairs_flat)
    start = args.pair_index
    end = min(start + args.chunk_size, total)
    if start >= total:
        log.error(f"pair_index {start} out of range (total={total})")
        return
    log.info(f"Running pairs {start}-{end-1} of {total} "
            f"({end - start} pairs in this task)")
    for idx in range(start, end):
        pair_dir, id1, id2, oeb1, oeb2 = all_pairs_flat[idx]
        log.info(f"[{idx}/{total-1}] {id1} VS {id2}")
        run_rocs_pair(id1, id2, oeb1, oeb2, pair_dir)
    return

# Sequential mode
for folder in folders:
    try:
        res = setup_folder(folder)
        if res is None:
            continue
        names, pairs, oeb_map, pact_map = res
        pair_dir = folder / "pair_results"
        pair_dir.mkdir(exist_ok=True)
        log.info(f"Running {len(pairs)} pairs for {folder.name}...")
        for id1, id2 in tqdm(pairs, desc=folder.name):
            run_rocs_pair(id1, id2, oeb_map[id1], oeb_map[id2], pair_dir)
        finalise(folder, names, pair_dir, pact_map)
    except Exception as e:
        log.exception(f"Error in {folder.name}: {e}")

log.info("All ROCS done.")
log.info("Run: python run_rocs.py --collect --root " + str(root))
log.info("Then: python run_eon_morgan.py --root " + str(root))

if __name__ == "__main__":
    main()

```

7.5.4 ChEMBL Ligand Random Dataset ROCS EON and Morgan Fingerprint Run

```

#!/usr/bin/env python3
"""
run_eon_morgan.py

Step 3 of the ChEMBL ligand similarity pipeline.

```

run after the first two scripts, gen_conformers and run_rocs

For each dataset folder this script runs EON on the ROCS-aligned OEB pairs produced by run_rocs.py, with the query and database file format

```
-query conformers/<idl>.oeb.gz          (full OMEGA ensemble of id1)
-dbase pair_results/<idl>_VS_<id2>_best.oeb.gz (aligned id2 pose)
```

Then computes Morgan fingerprint Tanimoto using RDKit (radius=2, 2048 bits). Finally, merges all similarity metrics (shape, color, combo, EON, Morgan) with pairwise activity differences into a flat results table.

Writes per-folder matrices, scatter plots, and a similarity CSV and assembles a multi-sheet Excel workbook (results/similarity_report.xlsx). Not necessary to keep this.

EON is run as a subprocess with LD_LIBRARY_PATH set in the child environment. This avoids an liboebio-3.3.1.1.so import failure that occurs when trying to set LD_LIBRARY_PATH inside an already-running Python process (the dynamic linker resolves library search paths at process startup and ignores subsequent changes to os.environ).

Usage options

```
python run_eon_morgan.py          # all folders, sequential
python run_eon_morgan.py --root .
python run_eon_morgan.py --skip_eon      # Morgan only (no EON licence)
python run_eon_morgan.py --skip_morgan   # EON only
python run_eon_morgan.py --pair_index N --chunk_size C # SLURM array
python run_eon_morgan.py --collect      # assemble results after array
"""
```

```
import os, sys, math, logging, argparse, subprocess, shutil
from pathlib import Path
from itertools import combinations
```

```
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
```

```
try:
    from tqdm import tqdm
except ImportError:
    def tqdm(it, **kw):
        it = list(it); n = len(it)
        for k, v in enumerate(it, 1):
            if n and k % max(1, n // 10) == 0:
                print(f" {k}/{n}")
            yield v
```

```
# Config
EON_CMD = "/global/software/alma8/commercial/openeye/2022.2.1/bin/eon"
ACT_FILENAME = "ligands.csv"
RESULTS_DIR = Path("results")
LOG_NAME = "eon_morgan_log.txt"
MIN_OEB_BYTES = 100
MORGAN_RADIUS = 2
MORGAN_BITS = 2048
SKIP_PATTERNS = ("rocs_summary", "oeomega", "_VS_", "shape_matrix",
                  "color_matrix", "combo_matrix", "activity_matrix",
                  "eon_matrix", "morgan_matrix")
```

```
# OpenEye shared library paths injected into the EON subprocess environment.
# These must be in LD_LIBRARY_PATH BEFORE the EON binary starts - setting
# os.environ inside a running Python process is too late for dlopen().
```

```
_VIDA_ROOT = ("/global/software/alma8/commercial/openeye/2022.2.1"
              "/arch/redhat-RHEL7-x64/vida/5.0.3.1")
```

```
_EON_LD_PATHS = [
    f"{_VIDA_ROOT}/bin",
    f"{_VIDA_ROOT}/lib/openeye/libs/python3-redhat-7-x64-g++7.4",
    f"{_VIDA_ROOT}/lib/openeye/libs",
    f"{_VIDA_ROOT}/lib",
```

```

]

# Logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s %(levelname)s: %(message)s",
    handlers=[logging.FileHandler(LOG_NAME), logging.StreamHandler(sys.stdout)],
)
log = logging.getLogger("run_eon_morgan")

# Args
def parse_args():
    p = argparse.ArgumentParser(description="EON + Morgan fingerprint analysis")
    p.add_argument("--root", default=".")
    p.add_argument("--pair_index", type=int, default=None)
    p.add_argument("--chunk_size", type=int, default=1)
    p.add_argument("--collect", action="store_true")
    p.add_argument("--skip_eon", action="store_true")
    p.add_argument("--skip_morgan", action="store_true")
    return p.parse_args()

# Activity helpers
UNIT_FACTORS = {"m": 1.0, "mm": 1e-3, "um": 1e-6, "\u00b5m": 1e-6,
                "nm": 1e-9, "pm": 1e-12}

def to_p(value, units):
    try:
        v = float(value)
    except Exception:
        return np.nan
    if v <= 0:
        return np.nan
    if not isinstance(units, str) or not units.strip():
        return np.nan
    key = units.strip().lower().replace("\u03bc", "u")
    for k, f in UNIT_FACTORS.items():
        if key == k or key.endswith(k):
            m = v * f
            return -math.log10(m) if m > 0 else np.nan
    return np.nan

def compute_pactivity(df):
    prefer = ["pIC50_calc", "pKi_calc", "pchembl_value"]
    for col in prefer:
        if col in df.columns:
            df = df.copy()
            df[col] = pd.to_numeric(df[col], errors="coerce")
            best = (df.sort_values(col, ascending=False, na_position="last")
                    .groupby("molecule_chembl_id", as_index=False).first())
            vals = best.set_index("molecule_chembl_id")[col].to_dict()
            if any(np.isfinite(v) for v in vals.values()):
                return vals
    df = df.copy()
    df["_p"] = df.apply(
        lambda r: to_p(r.get("standard_value"), r.get("standard_units")), axis=1
    )
    best = (df.sort_values("_p", ascending=False, na_position="last")
            .groupby("molecule_chembl_id", as_index=False).first())
    return best.set_index("molecule_chembl_id")["_p"].to_dict()

def read_activity(folder):
    f = folder / ACT_FILENAME
    if not f.exists():
        for ext in ("csv", "xls", "xlsx"):
            for cand in sorted(folder.glob(f"*. {ext}")):
                if any(p in cand.name for p in SKIP_PATTERNS):
                    continue
                try:

```

```

        peek = (pd.read_excel(cand, nrows=0)
                 if cand.suffix.lower() in (".xls", ".xlsx")
                 else pd.read_csv(cand, nrows=0))
        if "molecule_chembl_id" in peek.columns:
            f = cand
            break
    except Exception:
        continue
if not f.exists():
    return None
try:
    return (pd.read_excel(f) if f.suffix.lower() in (".xls", ".xlsx")
            else pd.read_csv(f))
except Exception as e:
    log.error(f"Cannot read {f}: {e}")
    return None

def load_exclusion_lists(folder):
    excluded_3d = set()
    excluded_all = set()
    fallback = folder / "conformers" / "fallback_report.csv"
    if fallback.exists():
        try:
            excluded_3d.update(
                pd.read_csv(fallback)["molecule_chembl_id"].dropna().tolist()
            )
        except Exception:
            pass
    excl_3d = folder / "conformers" / "excluded_ligands.txt"
    if excl_3d.exists():
        try:
            excluded_3d.update(
                l.strip() for l in excl_3d.read_text().splitlines()
                if l.strip() and not l.startswith("#")
            )
        except Exception:
            pass
    excl_all = folder / "conformers" / "excluded_all.txt"
    if excl_all.exists():
        try:
            excluded_all.update(
                l.strip() for l in excl_all.read_text().splitlines()
                if l.strip() and not l.startswith("#")
            )
        except Exception:
            pass
    return excluded_3d, excluded_all

def load_exclusion_list(folder):
    exc3d, excall = load_exclusion_lists(folder)
    return exc3d | excall

def load_smiles_map(df_act):
    smiles_cols = ["std_smiles", "canonical_smiles", "original_smiles", "smiles"]
    smap = {}
    for _, row in df_act.iterrows():
        lid = str(row.get("molecule_chembl_id", "")).strip()
        if not lid:
            continue
        smi = next(
            (row.get(c, "") for c in smiles_cols
             if isinstance(row.get(c, ""), str) and row.get(c, "").strip()),
            None,
        )
        if smi and lid not in smap:
            smap[lid] = smi
    return smap

```

```

# Morgan fingerprints
def compute_morgan_matrix(names, smiles_map):
    try:
        from rdkit import Chem
        from rdkit.Chem import AllChem, DataStructs
    except ImportError:
        log.error("RDKit not available - skipping Morgan fingerprints. "
                  "Load with: module load rdkit")
        return None

    fps = {}
    for name in names:
        smi = smiles_map.get(name)
        if not smi:
            continue
        mol = Chem.MolFromSmiles(smi)
        if mol is None:
            log.warning(f"RDKit could not parse SMILES for {name}")
            continue
        fps[name] = AllChem.GetMorganFingerprintAsBitVect(
            mol, MORGAN_RADIUS, nBits=MORGAN_BITS
        )

    mat = pd.DataFrame(np.nan, index=names, columns=names)
    valid = [n for n in names if n in fps]
    for i, a in enumerate(valid):
        mat.at[a, a] = 1.0
        for b in valid[i + 1:]:
            tc = DataStructs.TanimotoSimilarity(fps[a], fps[b])
            mat.at[a, b] = tc
            mat.at[b, a] = tc

    log.info(f"Morgan: {len(valid)}/{len(names)} molecules with valid fingerprints")
    return mat

# EON (subprocess)
def _eon_env():
    """
    Return the current environment unchanged - EON works with whatever
    environment the parent process inherited (module loads, OE_LICENSE etc).
    Modifying LD_LIBRARY_PATH here was causing EON to fail; passing
    os.environ directly works correctly.
    """
    return os.environ.copy()

def _eon_exists():
    return Path(EON_CMD).exists() or shutil.which(EON_CMD) is not None

def _parse_eon_rpt(rpt_path):
    """
    Parse EON RPT file.

    Actual column layout (from empirical header):
    Name EON_Query EON_Rank EON_ET_pb EON_ET_coul EON_ET_combo EON_ShapeTanimoto

    EON writes the query molecule itself as rank 1 (self-score = 2.000).
    The actual hit (id2) is rank 2. We skip any row where Name == EON_Query
    (self-score) and return the first non-self row.

    Returns (eon_et_combo, eon_et_pb, eon_et_coul) or None.
    """
    try:
        lines = rpt_path.read_text().splitlines()
        header = next(
            (l.split("\t") for l in lines if "EON_ET_combo" in l),
            None,
        )
        if header is None:
            header = next(

```

```

        (l.split() for l in lines if "eon_et_combo" in l.lower()),
        None,
    )
    if header is None:
        log.error(f"EON RPT has no recognised header: {rpt_path}")
        return None

    col = {h.strip().lower(): i for i, h in enumerate(header)}
    idx_combo = col.get("eon_et_combo", 5)
    idx_pb = col.get("eon_et_pb", 3)
    idx_coul = col.get("eon_et_coul", 4)
    idx_name = col.get("name", 0)
    idx_query = col.get("eon_query", 1)

    for line in lines[1:]:
        parts = line.split("\t") if "\t" in line else line.split()
        if not parts or len(parts) < 4:
            continue
        # Skip self-score row (Name == EON_Query)
        name = parts[idx_name].strip()
        query = parts[idx_query].strip() if idx_query < len(parts) else ""
        if name == query:
            continue
        try:
            return (float(parts[idx_combo]),
                    float(parts[idx_pb]),
                    float(parts[idx_coul]))
        except (IndexError, ValueError):
            continue
    except Exception as e:
        log.error(f"EON rpt parse error {rpt_path}: {e}")
    return None

def run_eon_pair(id1, id2, pair_dir):
    """
    Run EON on the ROCS-aligned pair saved by run_rocs.py.

    _best.oeb.gz already contains exactly the two winning conformers:
    molecule 1 - id1's winning conformer (id1_conf_idx from ROCS CSV)
    molecule 2 - id2's winning conformer (id2_conf_idx), aligned to molecule 1

    EON receives this file as both -query and -dbase. It reads the first
    molecule as the reference and the second as the aligned hit, so it
    evaluates ESP similarity between the exact same pair of conformers that
    ROCS identified as the best shape/colour match. No re-searching.

    Returns (eon_et_combo, eon_et_pb, eon_et_coul) or None.
    """
    tag = f"{id1}_VS_{id2}"
    result_csv = pair_dir / f"{tag}_eon.csv"

    # Resume: already done
    if result_csv.exists():
        try:
            row = pd.read_csv(result_csv).iloc[0]
            return (float(row["eon_et_combo"]),
                    float(row["eon_et_pb"]),
                    float(row["eon_et_coul"]))
        except Exception:
            pass

    best_oeb = pair_dir / f"{tag}_best.oeb.gz"

    # Handle reversed tag (shouldn't happen with sorted pair ordering but be safe)
    if not best_oeb.exists():
        rev_tag = f"{id2}_VS_{id1}"
        rev_best = pair_dir / f"{rev_tag}_best.oeb.gz"
        if rev_best.exists():
            log.info(f"EON | {tag} | using reversed pair ({rev_tag})")
            best_oeb = rev_best

```

```

if not best_oeb.exists():
    log.warning(f"EON SKIP | {tag} | missing best OEB")
    return None

# Log which conformers EON will see
rocs_csv = pair_dir / f"{tag}.csv"
if not rocs_csv.exists():
    rocs_csv = pair_dir / f"{id2}_VS_{id1}.csv"
if rocs_csv.exists():
    try:
        rocs_row = pd.read_csv(rocs_csv).iloc[0]
        id1_conf = rocs_row.get("id1_conf_idx", "?")
        id2_conf = rocs_row.get("id2_conf_idx", "?")
        log.info(f"EON | {tag} | query={id1} conf{id1_conf}, "
                f"dbase={id2} conf{id2_conf} - both from {best_oeb.name}")
    except Exception:
        log.info(f"EON | {tag} | using {best_oeb.name}")

import tempfile as _tf
eon_tmp = Path(_tf.mkdtemp(prefix=f"eon_{tag}_", dir=pair_dir))
prefix = str(eon_tmp / tag)

cmd = [
    EON_CMD,
    "-query", str(best_oeb), # molecule 1 = id1 winning conformer
    "-dbase", str(best_oeb), # molecule 2 = id2 winning conformer, aligned
    "-prefix", prefix,
]

try:
    r = subprocess.run(cmd, stdout=subprocess.PIPE,
                       stderr=subprocess.PIPE, timeout=600,
                       env=_eon_env())
except Exception as e:
    log.error(f"EON exception {tag}: {e}")
    shutil.rmtree(eon_tmp, ignore_errors=True)
    return None

# EON prints its banner to stdout and returns 0 on success.
# Log both streams at DEBUG so we can diagnose failures without noise.
stdout_txt = r.stdout.decode(errors="ignore")
stderr_txt = r.stderr.decode(errors="ignore")
log.debug(f"EON stdout | {tag} | {stdout_txt[:500]}")
if stderr_txt.strip():
    log.debug(f"EON stderr | {tag} | {stderr_txt[:500]}")

if r.returncode != 0:
    log.error(f"EON failed {tag} (rc={r.returncode})")
    log.error(f" stdout: {stdout_txt[:400]}")
    log.error(f" stderr: {stderr_txt[:400]}")
    shutil.rmtree(eon_tmp, ignore_errors=True)
    return None

# Find EON report file
rpt = next(
    (Path(f"{prefix}{s}") for s in ("_1.rpt", ".rpt")
     if Path(f"{prefix}{s}").exists()),
    None,
)
if rpt is None:
    all_rpts = sorted(eon_tmp.glob("*.rpt"))
    rpt = all_rpts[0] if all_rpts else None
if rpt is None:
    log.error(f"EON produced no RPT for {tag}")
    shutil.rmtree(eon_tmp, ignore_errors=True)
    return None

scores = _parse_eon_rpt(rpt)

# Delete only the EON tmpdir - query and best OEBs are kept

```

```

shutil.rmtree(eon_tmp, ignore_errors=True)

if scores is not None:
    eon_et_combo, eon_et_pb, eon_et_coul = scores
    esp_tanimoto = eon_et_pb + eon_et_coul # matches rocs_esp_morgan.py
    pd.DataFrame([
        {"id1": id1,
         "id2": id2,
         "eon_et_combo": eon_et_combo,
         "eon_et_pb": eon_et_pb,
         "eon_et_coul": eon_et_coul,
         "esp_tanimoto": esp_tanimoto,
        }]).to_csv(result_csv, index=False)
    log.info(f"EON OK | {tag} | et_combo={eon_et_combo:.3f} "
            f"et_pb={eon_et_pb:.3f} et_coul={eon_et_coul:.3f} "
            f"esp={esp_tanimoto:.3f}")
else:
    log.warning(f"EON NO SCORE | {tag}")

return scores

# Matrix helpers
def load_rocs_matrices(folder, names):
    mats = {}
    for label in ("shape", "color", "combo"):
        path = folder / f"{label}_matrix.csv"
        if path.exists():
            try:
                df = pd.read_csv(path, index_col=0)
                df = df.reindex(index=names, columns=names)
                mats[label] = df
            except Exception as e:
                log.warning(f"Could not load {path}: {e}")
    return mats

def collect_eon_matrix(names, pair_dir):
    """Assemble EON score matrices from per-pair_eon.csv files."""
    combo = pd.DataFrame(np.nan, index=names, columns=names)
    et_pb = pd.DataFrame(np.nan, index=names, columns=names)
    et_co = pd.DataFrame(np.nan, index=names, columns=names)
    esp = pd.DataFrame(np.nan, index=names, columns=names)
    for csv_path in pair_dir.glob("*_eon.csv"):
        try:
            row = pd.read_csv(csv_path).iloc[0]
            a, b = str(row["id1"]), str(row["id2"])
            if a not in names or b not in names:
                continue
            for df, k in [(combo, "eon_et_combo"),
                          (et_pb, "eon_et_pb"),
                          (et_co, "eon_et_coul"),
                          (esp, "esp_tanimoto")]:
                if k in row:
                    v = float(row[k])
                    df.at[a, b] = v
                    df.at[b, a] = v
        except Exception:
            continue
    for n in names:
        combo.at[n, n] = et_pb.at[n, n] = et_co.at[n, n] = esp.at[n, n] = 1.0
    return combo, et_pb, et_co, esp

def make_scatter(metric_df, act_mat, label, names, out_path):
    xs, ys = [], []
    for i, a in enumerate(names):
        for b in names[i + 1:]:
            mv = metric_df.at[a, b] if a in metric_df.index and b in metric_df.columns else
np.nan
            av = act_mat.at[a, b] if a in act_mat.index and b in act_mat.columns else
np.nan

```

```

        if pd.notna(mv) and pd.notna(av):
            xs.append(1.0 - float(mv))
            ys.append(float(av))

    if not xs:
        return
    plt.figure(figsize=(6, 5))
    plt.scatter(xs, ys, s=10, alpha=0.6)
    plt.xlabel(f"Distance (1 \u2013 {label})")
    plt.ylabel(f"\u0394 activity (pX)")
    plt.title(label)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig(out_path, dpi=200)
    plt.close()

# Results table
def build_results_table(folder, all_names, names_3d, pact_map,
                        rocs_mats, morgan_mat, eon_combo_mat, esp_mat):
    rows = []
    for i, a in enumerate(all_names):
        for b in all_names[i + 1:]:
            row = {"folder": folder.name, "id1": a, "id2": b}
            for label, mat in rocs_mats.items():
                try:
                    row[f"rocs_{label}"] = (
                        float(mat.at[a, b])
                        if a in mat.index and b in mat.columns else np.nan
                    )
                except Exception:
                    row[f"rocs_{label}"] = np.nan
            if morgan_mat is not None:
                try:
                    row["morgan_tanimoto"] = float(morgan_mat.at[a, b])
                except Exception:
                    row["morgan_tanimoto"] = np.nan
            else:
                row["morgan_tanimoto"] = np.nan
            if eon_combo_mat is not None:
                try:
                    row["eon_et_combo"] = (
                        float(eon_combo_mat.at[a, b])
                        if a in eon_combo_mat.index and b in eon_combo_mat.columns
                        else np.nan
                    )
                except Exception:
                    row["eon_et_combo"] = np.nan
            else:
                row["eon_et_combo"] = np.nan
            if esp_mat is not None:
                try:
                    row["esp_tanimoto"] = (
                        float(esp_mat.at[a, b])
                        if a in esp_mat.index and b in esp_mat.columns else np.nan
                    )
                except Exception:
                    row["esp_tanimoto"] = np.nan
            else:
                row["esp_tanimoto"] = np.nan
            pa = pact_map.get(a, np.nan)
            pb = pact_map.get(b, np.nan)
            row["pactivity_id1"] = pa if np.isfinite(pa) else np.nan
            row["pactivity_id2"] = pb if np.isfinite(pb) else np.nan
            row["delta_pactivity"] = (abs(pa - pb)
                                     if np.isfinite(pa) and np.isfinite(pb)
                                     else np.nan)
            combo = row.get("rocs_combo", np.nan)
            delta = row["delta_pactivity"]
            row["activity_cliff"] = (
                True if (pd.notna(combo) and combo >= 0.8
                        and pd.notna(delta) and delta >= 2.0)
                else False if (pd.notna(combo) and pd.notna(delta))
            )

```

```

        else np.nan
    )
    rows.append(row)
return pd.DataFrame(rows)

# Excel workbook
def write_excel_report(folder_tables, combined, output_path):
    try:
        from openpyxl import Workbook
        from openpyxl.styles import Font, PatternFill, Alignment, Border, Side
        from openpyxl.utils import get_column_letter
    except ImportError:
        log.warning("openpyxl not available - skipping Excel report")
        return

    HEADER_FILL = PatternFill("solid", start_color="2C5F8A", end_color="2C5F8A")
    HEADER_FONT = Font(bold=True, color="FFFFFF", name="Arial", size=10)
    CLIFF_FILL = PatternFill("solid", start_color="FFE0E0", end_color="FFE0E0")
    ALT_FILL = PatternFill("solid", start_color="F2F7FC", end_color="F2F7FC")
    BODY_FONT = Font(name="Arial", size=9)
    HDR_ALIGN = Alignment(horizontal="center", vertical="center", wrap_text=True)
    CELL_ALIGN = Alignment(horizontal="center", vertical="center")
    THIN = Side(style="thin", color="CCCCCC")
    BORDER = Border(left=THIN, right=THIN, top=THIN, bottom=THIN)

    SCORE_KEYS = ("rocs", "morgan", "eon", "esp", "pactivity", "delta")

    def _write_sheet(ws, df, title):
        ws.title = title[:31]
        ws.freeze_panes = "A2"
        cols = list(df.columns)
        for ci, c in enumerate(cols, 1):
            cell = ws.cell(row=1, column=ci, value=c)
            cell.font = HEADER_FONT; cell.fill = HEADER_FILL
            cell.alignment = HDR_ALIGN; cell.border = BORDER
        ws.row_dimensions[1].height = 36
        score_cols = {ci + 1 for ci, c in enumerate(cols)
                       if any(k in c for k in SCORE_KEYS)}
        for ri, (_, row) in enumerate(df.iterrows(), 2):
            is_cliff = row.get("activity_cliff") is True
            for ci, c in enumerate(cols, 1):
                val = row[c]
                if not isinstance(val, bool) and pd.isna(val):
                    val = ""
                cell = ws.cell(row=ri, column=ci, value=val)
                cell.font = BODY_FONT; cell.alignment = CELL_ALIGN
                cell.border = BORDER
                cell.fill = CLIFF_FILL if is_cliff else (
                    ALT_FILL if ri % 2 == 0 else cell.fill
                )
                if ci in score_cols and isinstance(val, float):
                    cell.number_format = "0.000"
            for ci, c in enumerate(cols, 1):
                w = 18 if c in ("id1", "id2") else 8 if c == "folder" else 13
                ws.column_dimensions[get_column_letter(ci)].width = w
            ws.auto_filter.ref = f"A1:{get_column_letter(len(cols))}{len(df) + 1}"

    def _write_summary(ws, folder_tables, combined):
        ws.title = "Summary"
        ws.freeze_panes = "A2"
        headers = ["Set", "Ligands", "Pairs", "Activity Pairs",
                  "Activity Cliffs", "Mean ROCS Combo",
                  "Mean Morgan", "Mean EON ET Combo", "Mean ESP Tanimoto"]
        for ci, h in enumerate(headers, 1):
            cell = ws.cell(row=1, column=ci, value=h)
            cell.font = HEADER_FONT; cell.fill = HEADER_FILL
            cell.alignment = HDR_ALIGN; cell.border = BORDER
        ws.row_dimensions[1].height = 36
        THIN2 = Side(style="thin", color="CCCCCC")
        B2 = Border(left=THIN2, right=THIN2, top=THIN2, bottom=THIN2)
        for ri, (name, df) in enumerate(

```

```

        list(folder_tables.items()) + [("ALL", combined)], 2):
    is_all = name == "ALL"
    vals = {
        "Set": name,
        "Ligands": len(set(df["id1"]) | set(df["id2"])),
        "Pairs": len(df),
        "Activity Pairs": int(df["delta_pactivity"].notna().sum()),
        "Activity Cliffs": int((df["activity_cliff"] == True).sum()),
        "Mean ROCS Combo": df.get("rocs_combo", pd.Series()).mean(),
        "Mean Morgan": df.get("morgan_tanimoto", pd.Series()).mean(),
        "Mean EON ET Combo": df.get("eon_et_combo", pd.Series()).mean(),
        "Mean ESP Tanimoto": df.get("esp_tanimoto", pd.Series()).mean(),
    }
    for ci, h in enumerate(headers, 1):
        val = vals[h]
        if isinstance(val, float) and np.isnan(val):
            val = ""
        cell = ws.cell(row=ri, column=ci, value=val)
        cell.font = Font(name="Arial", size=9, bold=is_all)
        cell.alignment = Alignment(horizontal="center")
        cell.border = B2
        if ci >= 6 and isinstance(val, float):
            cell.number_format = "0.000"
    for ci in range(1, len(headers) + 1):
        ws.column_dimensions[get_column_letter(ci)].width = 16

wb = Workbook()
wb.remove(wb.active)
_write_summary(wb.create_sheet(), folder_tables, combined)
for name, df in folder_tables.items():
    _write_sheet(wb.create_sheet(), df, name)
_write_sheet(wb.create_sheet(), combined, "All Sets")
output_path.parent.mkdir(parents=True, exist_ok=True)
wb.save(str(output_path))
log.info(f"Excel report: {output_path}")

# Main
def main():
    args = parse_args()
    root = Path(args.root)
    RESULTS_DIR.mkdir(exist_ok=True)

    slurm_id = os.environ.get("SLURM_ARRAY_TASK_ID")
    if slurm_id is not None:
        log.info(f"SLURM_ARRAY_TASK_ID={slurm_id} pair_index={args.pair_index}")

    folders = [d for d in sorted(root.iterdir())
                if d.is_dir() and read_activity(d) is not None]
    if not folders:
        log.error(f"No dataset folders found under {root}")
        sys.exit(1)
    log.info(f"{len(folders)} folder(s): {[f.name for f in folders]}")

    # SLURM array: EON only
    if args.pair_index is not None and not args.collect:
        if args.skip_eon:
            log.info("--skip_eon: nothing to do in array mode")
            return
        if not _eon_exists():
            log.error("EON binary not found - load openeye module")
            sys.exit(1)

    all_pairs = []
    for folder in folders:
        df_act = read_activity(folder)
        if df_act is None:
            continue
        excluded = load_exclusion_list(folder)
        names_3d = sorted(
            str(r["molecule_chembl_id"])
            for _, r in df_act.iterrows()

```

```

        if (pd.notna(r.get("molecule_chembl_id", ""))
            and str(r["molecule_chembl_id"]) not in excluded)
    )
    pair_dir = folder / "pair_results"
    pair_dir.mkdir(exist_ok=True)
    for id1, id2 in combinations(names_3d, 2):
        all_pairs.append((id1, id2, pair_dir))

total = len(all_pairs)
start = args.pair_index
end = min(start + args.chunk_size, total)
if start >= total:
    log.error(f"pair_index {start} out of range (total={total})")
    return
log.info(f"Running EON pairs {start}-{end-1} of {total}")
for idx in range(start, end):
    id1, id2, pair_dir = all_pairs[idx]
    log.info(f"[{idx}/{total-1}] EON {id1} VS {id2}")
    run_eon_pair(id1, id2, pair_dir)
return

# Sequential / collect mode
for folder in folders:
    log.info(f"--- {folder.name} ---")

    pair_dir = folder / "pair_results"
    pair_dir.mkdir(exist_ok=True)

    # Detect ligand names from existing ROCS score CSVs.
    # This avoids any dependency on activity column names or exclusion lists
    # - if a pair_results CSV exists, both ligands are eligible for EON/Morgan.
    rocs_csvs = sorted(f for f in pair_dir.glob("*_VS_*.csv")
                        if "_eon" not in f.name)
    if not rocs_csvs:
        log.warning(f"{folder.name}: no ROCS pair CSVs found in {pair_dir} "
                    f"-- run run_rocs.py and run_rocs.py --collect first")
        continue

    names_3d = sorted([
        lid
        for csv in rocs_csvs
        for lid in pd.read_csv(csv)[["id1", "id2"]].values.flatten().tolist()
    ])
    log.info(f"{folder.name}: {len(names_3d)} ligands, {len(rocs_csvs)} ROCS pairs")

    # EON
    if not args.skip_eon:
        if not _eon_exists():
            log.warning(f"{folder.name}: EON binary not found - "
                        f"load openeye module or use --skip_eon")
        else:
            pairs_3d = list(combinations(names_3d, 2))
            log.info(f"{folder.name}: running EON for {len(pairs_3d)} pairs")
            for id1, id2 in tqdm(pairs_3d, desc=f"{folder.name} EON"):
                run_eon_pair(id1, id2, pair_dir)
            eon_combo_mat, eon_et_mat, eon_eq_mat, esp_mat = (
                collect_eon_matrix(names_3d, pair_dir)
            )
            eon_combo_mat.to_csv(folder / "eon_et_combo_matrix.csv")
            eon_et_mat.to_csv(folder / "eon_et_pb_matrix.csv")
            eon_eq_mat.to_csv(folder / "eon_et_coul_matrix.csv")
            esp_mat.to_csv(folder / "esp_tanimoto_matrix.csv")

            wb_path = folder / "eon_matrices.xlsx"
            with pd.ExcelWriter(wb_path, engine="openpyxl") as writer:
                eon_combo_mat.to_excel(writer, sheet_name="EON_ET_combo")
                eon_et_mat.to_excel(writer, sheet_name="EON_ET_pb")
                eon_eq_mat.to_excel(writer, sheet_name="EON_ET_coul")
                esp_mat.to_excel(writer, sheet_name="ESP_Tanimoto")
            log.info(f"{folder.name}: EON matrices written -> {wb_path.name}")

```

Sources

- (1) Wu, S. Y.; McNae, I.; Kontopidis, G.; McClue, S. J.; McInnes, C.; Stewart, K. J.; Wang, S.; Zheleva, D. I.; Marriage, H.; Lane, D. P.; Taylor, P.; Fischer, P. M.; Walkinshaw, M. D. Discovery of a Novel Family of CDK Inhibitors with the Program LIDAEUS: Structural Basis for Ligand-Induced Disordering of the Activation Loop. *Structure* **2003**, *11* (4), 399–410. [https://doi.org/10.1016/S0969-2126\(03\)00060-1](https://doi.org/10.1016/S0969-2126(03)00060-1).
- (2) Williamson, D. S.; Parratt, M. J.; Bower, J. F.; Moore, J. D.; Richardson, C. M.; Dokurno, P.; Cansfield, A. D.; Francis, G. L.; Hebdon, R. J.; Howes, R.; Jackson, P. S.; Lockie, A. M.; Murray, J. B.; Nunns, C. L.; Powles, J.; Robertson, A.; Surgenor, A. E.; Torrance, C. J. Structure-Guided Design of Pyrazolo[1,5-a]Pyrimidines as Inhibitors of Human Cyclin-Dependent Kinase 2. *Bioorg. Med. Chem. Lett.* **2005**, *15* (4), 863–867. <https://doi.org/10.1016/j.bmcl.2004.12.073>.
- (3) Vulpetti, A.; Casale, E.; Roletto, F.; Amici, R.; Villa, M.; Pevarello, P. Structure-Based Drug Design to the Discovery of New 2-Aminothiazole CDK2 Inhibitors. *J. Mol. Graph. Model.* **2006**, *24* (5), 341–348. <https://doi.org/10.1016/j.jmgm.2005.09.012>.
- (4) Hole, A. J.; Baumli, S.; Shao, H.; Shi, S.; Huang, S.; Pepper, C.; Fischer, P. M.; Wang, S.; Endicott, J. A.; Noble, M. E. Comparative Structural and Functional Studies of 4-(Thiazol-5-Yl)-2- (Phenylamino)Pyrimidine-5-Carbonitrile CDK9 Inhibitors Suggest the Basis for Isotype Selectivity. *J. Med. Chem.* **2013**, *56* (3), 660–670. <https://doi.org/10.1021/jm301495v>.
- (5) Martin, M. P.; Endicott, J. A.; Noble, M. E. M. Structure-Based Discovery of Cyclin-Dependent Protein Kinase Inhibitors. *Essays in Biochemistry*. Portland Press Ltd November 8, 2017, pp 439–452. <https://doi.org/10.1042/EBC20170040>.
- (6) Liu, J. J.; Daniewski, I.; Ding, Q.; Higgins, B.; Ju, G.; Kolinsky, K.; Konzelmann, F.; Lukacs, C.; Pizzolato, G.; Rossman, P.; Swain, A.; Thakkar, K.; Wei, C. C.; Miklowski, D.; Yang, H.; Yin, X.; Wovkulich, P. M. Pyrazolobenzodiazepines: Part I. Synthesis and SAR of a Potent Class of Kinase Inhibitors. *Bioorg. Med. Chem. Lett.* **2010**, *20* (20), 5984–5987. <https://doi.org/10.1016/j.bmcl.2010.08.079>.
- (7) Schonbrunn, E.; Betzi, S.; Alam, R.; Martin, M. P.; Becker, A.; Han, H.; Francis, R.; Chakrasali, R.; Jakkaraj, S.; Kazi, A.; Sebt, S. M.; Cubitt, C. L.; Gebhard, A. W.; Hazlehurst, L. A.; Tash, J. S.; Georg, G. I. Development of Highly Potent and Selective Diaminothiazole Inhibitors of Cyclin-Dependent Kinases. *J. Med. Chem.* **2013**, *56* (10), 3768–3782. <https://doi.org/10.1021/jm301234k>.

- (8) Hardcastle, I. R.; Arris, C. E.; Bentley, J.; Boyle, F. T.; Chen, Y.; Curtin, N. J.; Endicott, J. A.; Gibson, A. E.; Golding, B. T.; Griffin, R. J.; Jewsbury, P.; Menyerol, J.; Mesguiche, V.; Newell, D. R.; Noble, M. E. M.; Pratt, D. J.; Wang, L. Z.; Whitfield, H. J. N²-Substituted O⁶-Cyclohexylmethylguanine Derivatives: Potent Inhibitors of Cyclin-Dependent Kinases 1 and 2. *J. Med. Chem.* **2004**, *47* (15), 3710–3722. <https://doi.org/10.1021/jm0311442>.
- (9) Kryštof, V.; Cankař, P.; Fryšová, I.; Slouka, J.; Kontopidis, G.; Džubák, P.; Hajdúch, M.; Srovnal, J.; De Azevedo, W. F.; Orság, M.; Paprskářová, M.; Rolčík, J.; Látr, A.; Fischer, P. M.; Strnad, M. 4-Arylazo-3,5-Diamino-1H-Pyrazole CDK Inhibitors: SAR Study, Crystal Structure in Complex with CDK2, Selectivity, and Cellular Effects. *J. Med. Chem.* **2006**, *49* (22), 6500–6509. <https://doi.org/10.1021/jm0605740>.
- (10) Zatloukal, M.; Jorda, R.; Gucký, T.; Řezníčková, E.; Voller, J.; Pospíšil, T.; Malínková, V.; Adamcová, H.; Kryštof, V.; Strnad, M. Synthesis and in Vitro Biological Evaluation of 2,6,9-Trisubstituted Purines Targeting Multiple Cyclin-Dependent Kinases. *Eur. J. Med. Chem.* **2013**, *61*, 61–72. <https://doi.org/10.1016/j.ejmech.2012.06.036>.
- (11) Sielecki, T. M.; Johnson, T. L.; Liu, J.; Muckelbauer, J. K.; Grafstrom, R. H.; Cox, S.; Boylan, J.; Burton, C. R.; Chen, H.; Smallwood, A.; Chang, C.-H.; Boisclair, M.; Benfield, P. A.; Trainor, G. L.; Seitz, S. P. Quinazolines as Cyclin Dependent Kinase Inhibitors. *Bioorg. Med. Chem. Lett.* **2001**, *11* (9), 1157–1160.
- (12) Yue, E. W.; DiMeo, S. V.; Higley, C. A.; Markwalder, J. A.; Burton, C. R.; Benfield, P. A.; Grafstrom, R. H.; Cox, S.; Muckelbauer, J. K.; Smallwood, A. M.; Chen, H.; Chang, C. H.; Trainor, G. L.; Seitz, S. P. Synthesis and Evaluation of Indenopyrazoles as Cyclin-Dependent Kinase Inhibitors. Part 4: Heterocycles at C3. *Bioorg. Med. Chem. Lett.* **2004**, *14* (2), 343–346. <https://doi.org/10.1016/j.bmcl.2003.11.008>.
- (13) Markwalder, J. A.; Arnone, M. R.; Benfield, P. A.; Boisclair, M.; Burton, C. R.; Chang, C. H.; Cox, S. S.; Czerniak, P. M.; Dean, C. L.; Doleniak, D.; Grafstrom, R.; Harrison, B. A.; Kaltenbach, R. F.; Nugiel, D. A.; Rossi, K. A.; Sherk, S. R.; Sisk, L. M.; Stouten, P.; Trainor, G. L.; Worland, P.; Seitz, S. P. Synthesis and Biological Evaluation of 1-Aryl-4,5-Dihydro-1H-Pyrazolo[3,4-*d*]Pyrimidin-4-One Inhibitors of Cyclin-Dependent Kinases. *J. Med. Chem.* **2004**, *47* (24), 5894–5911. <https://doi.org/10.1021/jm020455u>.
- (14) Echalié, A.; Bettayeb, K.; Ferandin, Y.; Lozach, O.; Clément, M.; Valette, A.; Liger, F.; Marquet, B.; Morris, J. C.; Endicott, J. A.; Joseph, B.; Meijer, L. Meriolins (3-(Pyrimidin-4-Yl)-7-Azaindoles): Synthesis, Kinase Inhibitory Activity, Cellular Effects, and Structure of a CDK2/Cyclin A/Meriolin Complex. *J. Med. Chem.* **2008**, *51* (4), 737–751. <https://doi.org/10.1021/jm700940h>.
- (15) Betzi, S.; Alam, R.; Martin, M.; Lubbers, D. J.; Han, H.; Jakkaraj, S. R.; Georg, G. I.; Schönbrunn, E. Discovery of a Potential Allosteric Ligand Binding Site in CDK2. *ACS Chem. Biol.* **2011**, *6* (5), 492–501. <https://doi.org/10.1021/cb100410m>.

- (16) Martin, M. P.; Alam, R.; Betzi, S.; Ingles, D. J.; Zhu, J. Y.; Schönbrunn, E. A Novel Approach to the Discovery of Small-Molecule Ligands of CDK2. *ChemBioChem* **2012**, *13* (14), 2128–2136. <https://doi.org/10.1002/cbic.201200316>.
- (17) Carbain, B.; Paterson, D. J.; Anscombe, E.; Campbell, A. J.; Cano, C.; Echalié, A.; Endicott, J. A.; Golding, B. T.; Haggerty, K.; Hardcastle, I. R.; Jewsbury, P. J.; Newell, D. R.; Noble, M. E. M.; Roche, C.; Wang, L. Z.; Griffin, R. J. 8-Substituted O 6-Cyclohexylmethylguanine CDK2 Inhibitors: Using Structure-Based Inhibitor Design to Optimize an Alternative Binding Mode. *J. Med. Chem.* **2014**, *57* (1), 56–70. <https://doi.org/10.1021/jm401555v>.
- (18) Pevarello, P.; Brasca, M. G.; Amici, R.; Orsini, P.; Traquandi, G.; Corti, L.; Piutti, C.; Sansonna, P.; Villa, M.; Pierce, B. S.; Pulici, M.; Giordano, P.; Martina, K.; Fritzen, E. L.; Nugent, R. A.; Casale, E.; Cameron, A.; Ciomei, M.; Roletto, F.; Isacchi, A.; Fogliatto, G. P.; Pesenti, E.; Pastori, W.; Marsiglio, A.; Leach, K. L.; Clare, P. M.; Fiorentini, F.; Varasi, M.; Vulpetti, A.; Warpehoski, M. A. 3-Aminopyrazole Inhibitors of CDK2/Cyclin A as Antitumor Agents. 1. Lead Finding. *J. Med. Chem.* **2004**, *47* (13), 3367–3380. <https://doi.org/10.1021/jm031145u>.
- (19) Kim, K. S.; Kimball, S. D.; Misra, R. N.; Rawlins, D. B.; Hunt, J. T.; Xiao, H. Y.; Lu, S.; Qian, L.; Han, W. C.; Shan, W.; Mitt, T.; Cai, Z. W.; Poss, M. A.; Zhu, H.; Sack, J. S.; Tokarski, J. S.; Chang, C. Y.; Pavletich, N.; Kamath, A.; Humphreys, W. G.; Marathe, P.; Bursuker, I.; Kellar, K. A.; Roongta, U.; Batorsky, R.; Mulheron, J. G.; Bol, D.; Fairchild, C. R.; Lee, F. Y.; Webster, K. R. Discovery of Aminothiazole Inhibitors of Cyclin-Dependent Kinase 2: Synthesis, x-Ray Crystallographic Analysis, and Biological Activities. *J. Med. Chem.* **2002**, *45* (18), 3905–3927. <https://doi.org/10.1021/jm0201520>.
- (20) Kontopidis, G.; McInnes, C.; Pandalaneni, S. R.; McNae, I.; Gibson, D.; Mezna, M.; Thomas, M.; Wood, G.; Wang, S.; Walkinshaw, M. D.; Fischer, P. M. Differential Binding of Inhibitors to Active and Inactive CDK2 Provides Insights for Drug Design. *Chem. Biol.* **2006**, *13* (2), 201–211. <https://doi.org/10.1016/j.chembiol.2005.11.011>.
- (21) Bramson, H. N.; Holmes, W. D.; Hunter, R. N.; Lackey, K. E.; Lovejoy, B.; Luzzio, M. J.; Montana, V.; Rocque, W. J.; Rusnak, D.; Shewchuk, L.; Veal, J. M.; Corona, J.; Walker, D. H.; Kuyper, L. F.; Davis, S. T.; Dickerson, S. H.; Edelstein, M.; Frye, S. V.; Gampe, R. T.; Harris, P. A.; Hassell, A. Oxindole-Based Inhibitors of Cyclin-Dependent Kinase 2 (CDK2): Design, Synthesis, Enzymatic Activities, and X-Ray Crystallographic Analysis. *J. Med. Chem.* **2001**, *44* (25), 4339–4358. <https://doi.org/10.1021/jm010117d>.
- (22) Malínková, V.; Řezníčková, E.; Jorda, R.; Gucký, T.; Kryštof, V. Trisubstituted Purine Inhibitors of PDGFR α and Their Antileukemic Activity in the Human Eosinophilic Cell Line EOL-1. *Bioorg. Med. Chem.* **2017**, *25* (24), 6523–6535. <https://doi.org/10.1016/j.bmc.2017.10.032>.

- (23) Wang, T.; Block, M. A.; Cowen, S.; Davies, A. M.; Devereaux, E.; Gingipalli, L.; Johannes, J.; Larsen, N. A.; Su, Q.; Tucker, J. A.; Whitston, D.; Wu, J.; Zhang, H. J.; Zinda, M.; Chuaqui, C. Discovery of Azabenzimidazole Derivatives as Potent, Selective Inhibitors of TBK1/IKK Kinases. *Bioorg. Med. Chem. Lett.* **2012**, *22* (5), 2063–2069. <https://doi.org/10.1016/j.bmcl.2012.01.018>.
- (24) Richardson, C. M.; Williamson, D. S.; Parratt, M. J.; Borgognoni, J.; Cansfield, A. D.; Dokurno, P.; Francis, G. L.; Howes, R.; Moore, J. D.; Murray, J. B.; Robertson, A.; Surgenor, A. E.; Torrance, C. J. Triazolo[1,5-a]Pyrimidines as Novel CDK2 Inhibitors: Protein Structure-Guided Design and SAR. *Bioorg. Med. Chem. Lett.* **2006**, *16* (5), 1353–1357. <https://doi.org/10.1016/j.bmcl.2005.11.048>.
- (25) Kosugi, T.; Mitchell, D. R.; Fujino, A.; Imai, M.; Kambe, M.; Kobayashi, S.; Makino, H.; Matsueda, Y.; Oue, Y.; Komatsu, K.; Imaizumi, K.; Sakai, Y.; Sugiura, S.; Takenouchi, O.; Unoki, G.; Yamakoshi, Y.; Cunliffe, V.; Frearson, J.; Gordon, R.; John Harris, C.; Kalloo-Hosein, H.; Le, J.; Patel, G.; Simpson, D. J.; Sherborne, B.; Thomas, P. S.; Suzuki, N.; Takimoto-Kamimura, M.; Kataoka, K. I. Mitogen-Activated Protein Kinase-Activated Protein Kinase 2 (MAPKAP-K2) as an Antiinflammatory Target: Discovery and in Vivo Activity of Selective Pyrazolo[1,5- a]Pyrimidine Inhibitors Using a Focused Library and Structure-Based Optimization Approach. *J. Med. Chem.* **2012**, *55* (15), 6700–6715. <https://doi.org/10.1021/jm300411k>.
- (26) Wyatt, P. G.; Woodhead, A. J.; Berdini, V.; Boulstridge, J. A.; Carr, M. G.; Cross, D. M.; Davis, D. J.; Devine, L. A.; Early, T. R.; Feltell, R. E.; Lewis, E. J.; McMenamin, R. L.; Navarro, E. F.; O'Brien, M. A.; O'Reilly, M.; Reule, M.; Saxty, G.; Seavers, L. C. A.; Smith, D. M.; Squires, M. S.; Trewartha, G.; Walker, M. T.; Woolford, A. J. A. Identification of N-(4-Piperidiny)-4-(2,6-Dichlorobenzoylamino)-1H- Pyrazole-3-Carboxamide (AT7519), a Novel Cyclin Dependent Kinase Inhibitor Using Fragment-Based X-Ray Crystallography and Structure Based Drug Design. *J. Med. Chem.* **2008**, *51* (16), 4986–4999. <https://doi.org/10.1021/jm800382h>.
- (27) Lücking, U.; Jautelat, R.; Krüger, M.; Brumby, T.; Lienau, P.; Schäfer, M.; Briem, H.; Schulze, J.; Hillisch, A.; Reichel, A.; Wengner, A. M.; Siemeister, G. The Lab Oddity Prevails: Discovery of Pan-CDK Inhibitor (R)-S-Cyclopropyl-S-(4-{[4-{[(1R,2R)-2-Hydroxy-1-Methylpropyl]Oxy}-5-(Trifluoromethyl)Pyrimidin-2-Yl]Amino}phenyl)Sulfoximide (BAY1000394) for the Treatment of Cancer. *ChemMedChem* **2013**, *8* (7), 1067–1085. <https://doi.org/10.1002/cmdc.201300096>.
- (28) Sayle, K. L.; Bentley, J.; Boyle, F. T.; Calvert, A. H.; Cheng, Y.; Curtin, N. J.; Endicott, J. A.; Golding, B. T.; Hardcastle, I. R.; Jewsbury, P.; Mesguiche, V.; Newell, D. R.; Noble, M. E. M.; Parsons, R. J.; Pratt, D. J.; Wang, L. Z.; Griffin, R. J. Structure-Based Design of 2-Arylamino-4-Cyclohexylmethyl-5-Nitroso-6-Aminopyrimidine Inhibitors of Cyclin-

- Dependent Kinases 1 and 2. *Bioorg. Med. Chem. Lett.* **2003**, *13* (18), 3079–3082.
[https://doi.org/10.1016/S0960-894X\(03\)00651-6](https://doi.org/10.1016/S0960-894X(03)00651-6).
- (29) Lee, J. W.; Hirota, T.; Ono, D.; Honma, S.; Honma, K. I.; Park, K.; Kay, S. A. Chemical Control of Mammalian Circadian Behavior through Dual Inhibition of Casein Kinase Ia and δ . *J. Med. Chem.* **2019**, *62* (4), 1989–1998.
<https://doi.org/10.1021/acs.jmedchem.8b01541>.
- (30) Li, X.; Huang, P.; Cui, J. J.; Zhang, J.; Tang, C. Novel Pyrrolyllactone and Pyrrolyllactam Indolinones as Potent Cyclin-Dependent Kinase 2 Inhibitors. *Bioorg. Med. Chem. Lett.* **2003**, *13* (11), 1939–1942. [https://doi.org/10.1016/S0960-894X\(03\)00312-3](https://doi.org/10.1016/S0960-894X(03)00312-3).
- (31) Hazel, P.; Kroll, S. H. B.; Bondke, A.; Barbazanges, M.; Patel, H.; Fuchter, M. J.; Coombes, R.; Ali, S.; Barrett, A. G. M.; Freemont, P. S. Inhibitor Selectivity for Cyclin-Dependent Kinase 7: A Structural, Thermodynamic, and Modelling Study. *ChemMedChem* **2017**, *12*, 372–380. <https://doi.org/10.1002/cmdc.201600535>:analytical.
- (32) Gaisina, I. N.; Gallier, F.; Ougolkov, A. V.; Kim, K. H.; Kurome, T.; Guo, S.; Holzle, D.; Luchini, D. N.; Blond, S. Y.; Billadeau, D. D.; Kozikowski, A. P. From a Natural Product Lead to the Identification of Potent and Selective Benzofuran-3-Yl-(Indol-3-Yl)Maleimides as Glycogen Synthase Kinase 3 β inhibitors That Suppress Proliferation and Survival of Pancreatic Cancer Cells. *J. Med. Chem.* **2009**, *52* (7), 1853–1863.
<https://doi.org/10.1021/jm801317h>.
- (33) Guzi, T. J. , P. K. D. M. P. , L. M. A. , K. K. M. Substituted Pyrazolo[1,5-a]Pyrimidines as Cyclin Dependent Kinase Inhibitors . US8580782B2, November 12, 2013.
- (34) Jones, C. D.; Andrews, D. M.; Barker, A. J.; Blades, K.; Daunt, P.; East, S.; Geh, C.; Graham, M. A.; Johnson, K. M.; Loddick, S. A.; McFarland, H. M.; McGregor, A.; Moss, L.; Rudge, D. A.; Simpson, P. B.; Swain, M. L.; Tam, K. Y.; Tucker, J. A.; Walker, M. The Discovery of AZD5597, a Potent Imidazole Pyrimidine Amide CDK Inhibitor Suitable for Intravenous Dosing. *Bioorg. Med. Chem. Lett.* **2008**, *18* (24), 6369–6373.
<https://doi.org/10.1016/j.bmcl.2008.10.102>.