Toward Foundational Advancements in Temporal Graph Learning

by

Kiarash Shamsi

A thesis submitted to Faculty of Graduate and Postgraduate Studies
of
The University of Manitoba
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science
University of Manitoba
Winnipeg

Artificial Intelligence Training Statement

The author does not grant permission for this thesis to be used in the training or development of artificial intelligence systems, machine-learning models, or related technologies, except where such use is explicitly allowed under an applied open license. The author requests that all AI developers and data aggregators exclude this work from training datasets.

Abstract

Temporal graphs capture the evolving interactions that define financial systems, communication platforms, and social networks. Their continuously changing structure makes them difficult to model, yet understanding their evolution is essential for applications such as fraud detection, behavioral prediction, and structural forecasting. The central goal of this thesis is to advance temporal graph learning toward generality and foundation-model capabilities, while opening this research direction to the community through new frameworks, datasets, tasks, and transferable modeling principles.

To initiate this progression, we first investigate whether novel computational frameworks from machine learning can reveal the global structural signals that evolving graphs exhibit. Through this exploration, Topological Forest shows that Topological Data Analysis (TDA) provides stable multiscale descriptors that traditional models fail to capture, demonstrating that topological reasoning can serve as a powerful foundation for understanding complex temporal behavior. A second major gap in the field is the absence of standardized, diverse temporal graph datasets. This limitation has historically prevented fair comparison between methods and slowed the development of new ideas. Chartalist addresses this barrier by providing the first machine-learning-ready blockchain dataset with rich temporal structure and clearly defined tasks, enabling reproducible research and supplying the breadth necessary for scalable model development. With these foundations in place, the thesis introduces GraphPulse, which formalizes the task of temporal graph property prediction and shows how topology-inspired temporal signatures can capture global structural dynamics. These insights reveal that temporal networks across domains share common patterns, motivating the question of whether models can generalize across networks. MiNT answers this by establishing the first multi-network training paradigm for temporal graphs, demonstrating that a single model can learn transferable temporal representations and perform well on unseen networks without retraining. Finally, Hydra brings these contributions together by unifying classification, regression, and forecasting within a single temporal backbone, showing that multi-task learning across

many networks is both feasible and effective. Hydra marks a concrete step toward temporal graph foundation models by enabling broad generalization across tasks and domains through a shared representation.

This thesis advances temporal graph learning by introducing new structural perspectives, establishing standardized resources, and developing models that capture and generalize the behavior of evolving networks. Through contributions that deepen our understanding of global temporal structure, enable meaningful comparison across diverse datasets, and demonstrate transferable and multi-task temporal modeling, this work pushes the field toward broader, more scalable capabilities. It sets the stage for the emergence of temporal graph foundation models and provides the conceptual tools, practical methodologies, and empirical evidence needed to drive the next generation of research in dynamic network analysis.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr. Cuneyt Gurcan Akcora, for his continuous guidance, encouragement, and mentorship throughout my doctoral studies. His insight, patience, and vision have shaped every stage of this research and inspired my development as a researcher and scholar.

I am deeply thankful to my wife, Yasmin Namiri, for her constant encouragement, patience, and support throughout this journey. I am also thankful to my parents, Farokh Shamsi and Manijeh Khodabakhshi, whose love, guidance, and unwavering belief in me made this path possible. I am grateful as well to my brother, Siavash Shamsi, for his continued encouragement and support along the way.

I would also like to thank Dr. David Gerhard, Head of the Department of Computer Science, for his leadership and support, as well as Dr. Celine Latulipe and Dr. Andrea Bunt, Associate Heads for Graduate Studies, for their guidance and for fostering a strong research culture within the department.

I am very grateful to Dr. Baris Coskunuzer and Dr. Guillaume Rabusseau for their collaboration, valuable feedback, and continued support throughout my research journey. I would also like to extend my sincere appreciation to Dr. Sara Rouhani and Dr. Saman Muthukumarana, for their technical expertise and thoughtful discussions, which contributed meaningfully to the development of this work.

I am sincerely thankful to my lab members in both of my supervisors' research groups for their discussions, ideas, and help along the way. I am also fortunate to have collaborated with many talented researchers, including Tran Gia Bao Ngo, Farimah Poursafaei, Shenyang Huang, Poupak Azad, and Razieh Shirzadkhani whose contributions and teamwork greatly strengthened this research and helped bring key projects to completion.

This research was supported by the Canadian NSERC Discovery Grant titled *Data Science on Blockchains*, the National Science Foundation, the Simons Foundation, and the Canadian Institute

for Advanced Research through the CIFAR AI Chair Program. Additional financial support was provided by the University of Manitoba Graduate Fellowship. Their generous funding made this research possible and enabled the exploration of new directions in temporal graph learning and data science.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent decades, the rapid digitization of society has produced unprecedented volumes of interconnected data. Social networks connect billions of users worldwide [1, 2], enabling continuous streams of interactions through messages, posts, and shared content. Financial systems process millions of transactions every day [3], reflecting the movement of capital across institutions, markets, and individuals. At the same time, blockchain infrastructures have introduced decentralized platforms where people exchange value globally without relying on traditional intermediaries [4]. These and many other modern systems generate data that is inherently relational and dynamic. A natural way to model such systems is through *temporal graphs*, where entities are represented as nodes, interactions as edges, and the entire structure evolves over time.

Temporal graph learning has emerged as a new field of research for analyzing such data, enabling the discovery and prediction of patterns in dynamic systems. Applications of this paradigm are wide-ranging: improving recommendation systems by modeling user preferences as they evolve over time [5, 6], detecting fraud by spotting unusual transaction patterns in financial or blockchain networks [7], understanding how diseases spread across populations [8], or predicting how traffic flows and congests in urban areas [9]. For these applications, models must be not only accurate but also efficient and scalable, as real-world temporal graphs often contain millions of nodes and billions of timestamped interactions.

The central challenge in temporal graph learning is that graphs are not static objects. Their structure is inherently time dependent, meaning models must capture how networks grow, reorganize, or decay across different points in time [10]. Unlike static graph learning, which assumes a fixed snapshot, temporal graph learning must manage complex dependencies across a timeline of evolving snapshots that differ in structure and patterns [11]. These challenges set it apart from more established fields such as computer vision and natural language processing. In those areas, the availability of shared large-scale benchmarks, such as ImageNet [12] and GLUE [13], has driven rapid methodological progress and reproducible evaluation. Temporal graph learning, however, is still in its early development stages. Large and standardized datasets remain scarce, experimental setups are often fragmented, and the use of novel approaches in this domain is still at an early stage of research [14].

To move the field forward, there is a clear need for both better datasets and innovative methods that can capture the unique dynamics of evolving graphs. On one hand, datasets must be sufficiently large and diverse to reflect the complexity of real-world systems and provide fair ground for evaluation. On the other hand, methods must go beyond static or local views of networks and be capable of capturing global temporal structures, allowing predictions that show how entire systems change over time. Systematically addressing these needs will enable temporal graph learning to transition from an emerging field into a mature research area. By doing so, it has the potential to reach the same impact seen in fields like computer vision and natural language processing, where the synergy of robust benchmarks and innovative methods has led to significant breakthroughs.

1.1 Research Motivation

Temporal graph learning provides a new way to study systems that change over time, but several main obstacles have slowed progress. The first obstacle is the need for methods that go beyond traditional machine learning techniques and can capture the overall structures of evolving graphs. Traditional models often focus only on local features, which makes them less effective at track-

ing the structural changes in dynamic systems. This encourages exploring novel methods that can capture the changing nature of these graphs. Topological data analysis has strong potential in this area, as it emphasizes the shape and structure of data and can identify global patterns concisely and reliably. Although it has been used in other areas of machine learning, its ability to build efficient, interpretable models that summarize complex temporal patterns has not yet been extensively studied in the field of temporal graph learning.

The second obstacle is the shortage of large-scale datasets that support systematic research. Many available graph datasets are either static or too small to represent the dynamics of real-world systems. This gap is clear in fields like finance and blockchain, where data is plentiful but rarely prepared in formats suitable for machine learning. Without such resources, it becomes difficult to compare methods fairly or test them under realistic conditions. Providing datasets and libraries that capture the temporal and multilayer aspects of these systems is therefore essential to enable consistent and reproducible progress.

The third obstacle involves the scope of tasks used to evaluate temporal graph models. Most existing works on temporal graph learning have focused on local predictions at the node or edge level, ignoring graph-level properties that describe how entire systems grow, shrink, or reorganize. Expanding the field to include graph-level property prediction tasks is crucial because these tasks uncover patterns that local predictions cannot reveal.

The fourth obstacle concerns transferability. Most existing models are trained on a single network and do not generalize well to new, unseen networks or domains. This limitation is impractical in areas like financial systems, where networks are numerous, large, and constantly evolving. Overcoming this requires frameworks capable of learning from collections of different temporal graphs and still generalizing to new ones. Achieving transferability is a crucial step toward making temporal graph learning scalable and toward developing foundational models for dynamic networks.

A fifth obstacle arises in the efficiency and adaptability of temporal graph transferable models. Even with transferability, most existing approaches are not designed to handle multiple predictive tasks simultaneously or adapt efficiently across diverse domains. Real-world systems often

involve many interconnected processes that require both classification and regression predictions under a shared framework. Addressing this limitation would enable models that are not only transferable but also multi-task, efficient, and more powerful in capturing diverse temporal behaviors across networks. Such advancement would mark an important step toward practical, scalable, and general-purpose models for temporal graphs.

In response to these challenges, this thesis aims to create a unified research framework that introduces innovative methods, develops large datasets, defines new tasks, and proposes transferable and adaptable models, thus laying the groundwork for future advancements in temporal graph learning.

1.2 Research Objectives

This thesis presents five primary objectives that together create a structured framework for advancing temporal graph learning.

The first goal is to explore novel approaches in machine learning that can enhance both efficiency and scalability of the model. Traditional methods often focus on local patterns or require large ensembles, which are computationally costly. To address this, we evaluate topological data analysis as a framework that captures the overall structure of data and offers concise yet informative representations. Demonstrating the value of such approaches in traditional settings creates a methodological foundation that can later be expanded to temporal graphs, where changing structures require models that summarize complex patterns over time.

The second objective is to address the lack of large-scale temporal graph datasets and benchmarks. Without these resources, it is challenging to test models in realistic conditions or compare approaches reliably. We achieve this by developing a library and datasets that convert raw blockchain data into machine learning-ready temporal graphs. These datasets are designed to reflect both the temporal and multilayer aspects of real-world networks, making them suitable for large-scale experiments. Providing these resources to the community promotes reproducibility,

lowers the barrier for new researchers, and lays a strong foundation for systematic progress in temporal graph learning.

The third goal is to develop a state-of-the-art temporal graph model that applies innovative methods to real-world data and introduces new temporal tasks for the field. While most previous work has focused on node- or edge-level predictions, we shift the focus to graph-level property prediction, which reflects global structural changes, such as the growth or fragmentation of networks. By incorporating topological descriptors into the model design, we aim to show that topology-informed temporal models can achieve high predictive performance on large dynamic graphs and open up a new direction of graph-level temporal tasks.

The fourth objective is to push the boundaries of temporal graph learning by addressing the challenge of transferability. Current methods usually need training separate models for each network, which is impractical in fields like finance, where networks are numerous and constantly changing. We present a framework that enables models to learn from collections of different temporal graphs and apply them to unseen ones. By showing transferability at scale and providing the first benchmark dedicated to this area, this work paves a new path in temporal graph learning, moving the field closer to creating foundation models for dynamic networks.

The fifth objective is to advance efficiency, adaptability, and predictive capacity in temporal graph transferable models by unifying multiple learning objectives under a shared framework. Real-world systems require models capable of performing a range of predictive tasks across diverse temporal networks. This objective focuses on developing an approach that supports multi-task learning, improves computational efficiency, and enhances generalization across domains. Addressing this goal will lead to models that are not only transferable but also more powerful and practical, thereby forming an essential step toward scalable, general-purpose temporal graph foundation models.

Together, these five objectives define a unified research path that connects methods, datasets, tasks, and frameworks, with the goal of advancing temporal graph learning toward efficiency, transferability, and broader applicability across domains.

1.3 Thesis Overview

This thesis takes a structured path that reflects the limitations of temporal graph learning and the steps taken to overcome them.

- **Chapter 2** provides the literature review and preliminary studies. This chapter introduces existing work in temporal graph learning and related domains, and lays the groundwork for the contributions that follow.

- **Chapter 3** begins the original contributions by studying the role of topology in machine learning. We investigate whether topological tools can improve efficiency and highlight their potential as a guiding principle for future model design. This serves as a conceptual starting point, showing that topology can enhance learning systems and motivating its use in subsequent chapters.

- **Chapter 4** turns to the challenge of data. Temporal graph research requires high-quality benchmarks, yet most existing datasets are either static, small in scale, or limited in scope. To overcome this barrier, we introduce a dataset resource that makes large, dynamic blockchain graphs accessible for machine learning and defines tasks that enable systematic evaluation. This step ensures that temporal graph learning can be studied on real-world, large-scale networks.

- **Chapter 5**, after providing both a novel methodological approach and a new large-scale dataset, introduces a state-of-the-art temporal model for graph-level tasks. This model, GraphPulse, integrates topological data analysis into temporal graph learning, establishing a new research direction by predicting graph-level properties rather than focusing solely on nodes or edges. GraphPulse is novel in both formulation and design, and it demonstrates how TDA can capture evolving global structures in large temporal networks.

- **Chapter 6**, after introducing a state-of-the-art model, pushes the boundaries of temporal graph learning further by moving to the challenge of transferability across networks. We present a framework that enables models to learn across multiple temporal graphs and generalize to unseen ones, showing clear improvements in zero-shot performance. This step represents the first

systematic attempt to study transferability in temporal graphs and paves the way for temporal graph foundation models.

- **Chapter 7** extends the path toward foundation models by addressing the next challenge which is efficiency, adaptability, and multi-task learning in temporal graphs. This chapter introduces a unified framework that can handle multiple prediction objectives, including both classification and regression tasks, across diverse temporal networks. By combining transferability with multi-task learning, this contribution advances the development of more powerful and scalable temporal graph models, marking a significant step toward practical temporal graph foundation models.

- **Chapter 8** concludes the thesis with a discussion of limitations. It outlines future opportunities, including directions for multi-task learning, improving efficiency, and continuing the path toward developing graph foundation models.

1.4 Contributions Within the Overall Research Path

This section provides an overview of the works included in the thesis and explains how each contributes to the overall research path.

- **Topological Forest** In Chapter 3, we introduce Topological Forest, a novel machine learning model that leverages topological data analysis (TDA) to improve the efficiency of random forest ensembles. Each decision tree is first encoded as a graph, where parent-child feature splits become edges, and a collection of topological and structural features is extracted from these graphs. Using TDA Mapper, similar decision trees are clustered in a topological network, and representative trees are selected to form a reduced ensemble. This process yields a significantly smaller forest while preserving the diversity of predictive patterns. Experiments on large-scale classification benchmarks show that Topological Forest achieves inference speeds more than 100 times faster than standard random forests, with at most 2% loss in AUC. This chapter shows that

TDA can be systematically integrated into classical ML to achieve both efficiency and scalability. Topological Forest provides an important conceptual step toward using topology as a tool for machine learning model design.

- **Chartalist** Chapter 4 presents Chartalist, the first ML-ready dataset collection for blockchain transaction networks. An important obstacle in temporal graph learning has been the absence of real-world datasets that are both large and well-structured for research. Chartalist addresses this need by turning raw blockchain data into temporal graphs that reflect the evolving and multi-layer nature of both systems for both UTXO-based blockchains (e.g., Bitcoin) and account-based blockchains (e.g., Ethereum). With this resource, researchers can now explore blockchain as a temporal graph domain in a consistent and reproducible way. Chartalist is important because it lowers the barrier to working with complex financial networks, provides a foundation for fair comparison across methods, and creates new opportunities to study temporal graph learning on real-world systems at scale. *Dataset Website:* `https://chartalist.org` The datasets and benchmark tasks are openly available through this website, ensuring easy access and reproducibility for the community.

- **GraphPulse** After introducing a novel methodological approach and a large-scale dataset, Chapter 5 presents GraphPulse, a state-of-the-art temporal graph model that combines TDA for graph-level property prediction. In this formulation, temporal graphs are represented as sequences of snapshots, and topological descriptors are extracted to summarize the global structure of each snapshot. GraphPulse then learns to predict whether high-level graph properties, such as edge growth or largest component expansion, will increase or decrease in future intervals. Unlike existing approaches that focus on node or edge-level tasks, GraphPulse emphasizes graph-level prediction, extending the scope of temporal graph learning by modeling global structural evolution as a predictive signal. Experiments demonstrate that GraphPulse achieves competitive or superior performance compared to standard temporal graph neural networks, establishing a new research direction and highlighting the practical utility of TDA in temporal graph learning.

- **MiNT** Chapter 6 pushes the boundaries of temporal graph learning by addressing the challenge of transferability in temporal graph learning. Until now, most approaches required training a separate model for each network. This is not practical in many domains, such as financial systems, where networks are numerous, large, and constantly changing. MiNT (Multi-Network Transfer) introduces the first large-scale benchmark of 84 Ethereum token networks, complemented by eight social interaction networks, and proposes a multi-network training algorithm that enables models to learn across distinct temporal graphs. The MiNT framework uses all networks together within the training process and relies on the MiNT-train algorithm, which is designed to avoid information leakage or context bias during learning. This setup shows, for the first time, how temporal models can be trained on collections of independent dynamic graphs while remaining capable of generalizing to unseen ones. MiNT models trained on up to 64 networks generalize effectively to 20 held-out networks, showing a clear neural scaling trend: performance improves consistently with the number of pre-training networks. Alongside the framework, we publicly release the MiNT benchmark dataset and provide open access to the codebase, introducing this direction to the research community. MiNT provides the first systematic evidence that temporal graph models can capture transferable dynamics across different networks, paving the way toward the development of temporal graph foundation models.

- **Hydra** Chapter 7 presents an advanced temporal graph model that builds upon the ideas of transferability introduced earlier and takes a major step toward efficiency and adaptability. While MiNT showed that knowledge can be transferred across multiple temporal networks, it still required retraining for each individual task and was computationally demanding. The model introduced in this chapter addresses these challenges by learning shared temporal representations that can support several prediction objectives at once, including both classification and regression tasks. It achieves up to **22 times faster** training speed than MiNT while maintaining stronger performance across diverse networks. By combining efficiency, multi-task learning, and broad generalization, this contribution overcomes the limitations of previous frameworks and represents a significant move toward practical and scalable temporal graph foundation models.

1.5 Contribution of Authors

Chapter 1 was written for this thesis.

Chapter 2 was written for this thesis.

Chapter 3 is based on the *IEEE Access* journal article [15] authored by Murat Ali Bayir, Kiarash Shamsi, Huseyincan Kaynak, and Cuneyt Gurcan Akcora. As the second author, I contributed in designing the Topological Forest framework, implemented major components of the model, conducted experiments, analyzed results, and contributed to writing and revising the manuscript. Murat led the project and initial development. Huseyincan supported framework design and implementation. Cuneyt supervised the work and provided guidance and feedback on the paper.

Chapter 4 is based on the NeurIPS 2022 [16] paper authored by Kiarash Shamsi, Friedhelm Victor, Murat Kantarcioglu, Yulia Gel, and Cuneyt Gurcan Akcora. As the first author, I contributed in designing Chartalist, performed data cleaning and preprocessing, developed the Chartalist website and Python library, prepared baseline tables, conducted experiments, and contributed to writing and revising the manuscript. Friedhelm Victor supported dataset preparation and consistency checks across networks, assisted in preparing baseline models, and provided domain expertise on blockchain transaction structures. Murat and Yulia provided methodological support and manuscript feedback. Cuneyt supervised the overall project and contributed to refining the narrative.

Chapter 5 is based on the ICLR 2024 paper [17] authored by Kiarash Shamsi, Farimah Poursafaei, Shenyang Huang, Tran Gia Bao Ngo, Baris Coskunuzer, and Cuneyt Gurcan Akcora. As the first author, I proposed the GraphPulse framework, introduced the temporal graph property prediction task, developed the TDA-inspired representations, implemented the method, preparing baselines, conducted primary experiments, and manuscript writing. Farimah and Shenyang contributed to the overall framework design and assisted in preparing models. Bao contributed to implementing

and coding the empirical evaluation for Mapper trajectories and supported additional experimental components. Baris provided theoretical guidance on Mapper-based topology, TDA concepts, and structural interpretation of temporal trajectories. Cuneyt supervised the project and provided detailed feedback on modeling decisions, experiments, and the presentation of results.

Chapter 6 is based on the NeurIPS 2025 paper [18] authored by Kiarash Shamsi, Tran Gia Bao Ngo, Razieh Shirzadkhani, Shenyang Huang, Farimah Poursafaei, Poupak Azad, Reihaneh Rabbany, Baris Coskunuzer, Guillaume Rabusseau, and Cuneyt Gurcan Akcora. As a co-first author, I co-developed the MiNT framework, implemented the multi-network training pipeline, designed the transfer evaluation protocol, conducted large-scale experiments across all networks, analyzed results, and contributed in paper writing. Bao and Razieh contributed significantly to MiNT's development, experimental evaluation, and baseline implementations. Shenyang and Farimah assisted with framework design, benchmark preparation, and evaluation discussions. Poupak supported data collection efforts. Reihaneh, Baris and Guillaume contributed conceptual guidance on transferability and manuscript refinement. Cuneyt supervised the project, shaped the research direction, and provided high-level feedbacks.

Chapter 7 is based on the recent work that was submitted to the ICLR 2026 submission authored by Kiarash Shamsi, Farimah Poursafaei, Tran Gia Bao Ngo, Reihaneh Rabbany, Baris Coskunuzer, Guillaume Rabusseau, Shenyang Huang, and Cuneyt Gurcan Akcora. As the first author, I led the development of Hydra architecture, designed the multi-task temporal learning framework, implemented the unified model, conducted all major experiments and ablations, and wrote the major parts of the manuscript. Farimah contributed to model design discussions, model implementation, experimental analysis and manuscript writing. Bao supported model design choices, baseline comparisons, and evaluation setup. Reihaneh, Guillaume, and Baris contributed insights on multi-task learning, refinement of the methodological framing, theoretical and structural guidance related to topology and temporal modeling, and feedback on the manuscript. Shenyang contributed to the supervision of the project and provided feedback and guidance on the DOS formulation, spectral

trajectory design, and structural aspects of the model. Cuneyt supervised the project and provided direction and feedback on the architecture, experiments, and writing.

Chapter 8 was written for the thesis.

Chapter 2

Related Work and Background

In this chapter, we present the essential background and notations for this thesis. We start with topological data analysis, a framework that captures the structural and geometric properties of data and has recently gained popularity in machine learning. Next, we move to graph learning, introducing graph notation and then reviewing the literature on temporal graph learning, which is central to this work. Finally, we explore the emerging research area of temporal graph foundation models (TGFM), highlighting how recent advances aim to create generalizable models in graph learning. This thesis advances temporal graph learning in this area, and the overview here provides the context for our contributions within the broader research landscape.

2.1 Topological Data Analysis

Topological Data Analysis provides a framework for extracting structural descriptors from data using algebraic topology. Its main objective is to capture global properties such as connectivity, cycles, and voids that persist across multiple scales. These properties are invariant under coordinate transformations and stable with respect to noise, which makes TDA suitable for analyzing complex and high-dimensional data [19, 20, 21].

2.1.1 Foundations and Core Methods

At the conceptual level, TDA represents data as simplicial complexes and studies topological invariants that describe their shape. A key theoretical milestone is the stability of persistence diagrams, which guarantees that small perturbations in the data lead to only minor changes in the resulting topological summaries [22]. Two methods are most commonly used in TDA.

- **Persistent Homology (PH)** studies the evolution of topological features (connected components, loops, voids, etc.) of a dataset at various spatial resolutions [23]. It can produce persistence diagrams or barcodes that describe when features appear and disappear. This method was formally introduced by Edelsbrunner, Letscher, and Zomorodian [24]. To make PH applicable in machine learning, several strategies have been proposed, including persistence images [25], persistence landscapes [26], and differentiable layers such as PersLay [27]. These approaches allow persistence diagrams to be integrated directly or indirectly into standard learning pipelines. PH has recently been utilized as a powerful feature extractor and combined with deep learning methods [28, 27, 29, 30] in node and graph classification tasks. However, PH-based methods face limitations in their application to large networks due to their high computational complexity.

- **Mapper** introduced by Singh, Mémoli, and Carlsson [31] and further formalized by Carlsson [32], is a combinatorial TDA method that converts complex, high-dimensional data into interpretable graph-based summaries. It operates by combining a filter function, a covering of its range, and local clustering, producing a low-dimensional, coordinate-independent graph representation that captures the macro-level organization of data [33, 34]. This representation exposes structural features such as branching trajectories, transitions, and rare subgroups that traditional clustering or dimensionality-reduction techniques often overlook. Mapper has been widely adopted across scientific and industrial domains where data are high-dimensional, heterogeneous, or noisy. In biomedicine, it revealed previously unknown breast cancer subtypes with distinct mutational profiles [35] and stratified patient populations from clinical records, uncovering rare subgroups not detected by conventional clustering [36]. In materials science, it

summarizes heterogeneous atomic configurations and reveals structural transitions in crystalline and amorphous solids [37]. In finance, it identified critical market transitions and anomalies during periods of economic instability [38]. Further applications include neuroscience, where it modeled dynamic brain states [39], and image and shape analysis, where it exposed global manifold structures in visual data [31]. Recent developments, such as HYppo [40] and Pheno-Mapper [41], have extended Mapper to more scalable and automated frameworks for modern large-scale datasets. Beyond these domains, Mapper principles have inspired advances in graph representation learning, where its graph-construction paradigm has been integrated into neural frameworks to enhance structural representation quality [42, 43]. Collectively, these studies demonstrate that Mapper is particularly effective for analyzing sparse, noisy, or heterogeneous datasets, providing interpretable and scalable graph representations. Compared to persistent homology, it delivers explicit structural summaries of the data manifold, making it especially powerful for exploratory analysis and data compression. To the best of our knowledge, this thesis presents the first application of Mapper in the context of *temporal graph machine learning*, extending its utility from static data analysis to dynamic network modeling.

TDA can be used to extract topological features from data to use them as inputs for the machine learning models, or it can be used to improve the model's design and study some aspects of the model [44]. In terms of TDA feature extraction, Harer et al. [45] used total persistence of a persistence diagram, Chen et al. [46] applied p-norm, and Atienza et al. [47] used persistent entropy as a topological descriptor input feature. Rieck et al. [48] used Betti curves [49] to summarize the features. Chevyrev et al. [50] utilized this representation and persistent diagrams to develop a classifier using a random forest and support vector machine. Bubenik et al. presented persistent landscape, a new topological descriptor for mapping the persistent diagram to function space [26]. Zhao et al.used TDA features for graph classification [51]. These studies showed that TDA features could be well-suited as inputs for machine learning models to improve their accuracy.

Some studies applied TDA to design a machine learning model or improve some aspects of a model. Moor et al. presented a topological auto-encoder for low-dimensional representation of

input data features [52]. Yuvaraj et al. used TDA to study complex multilayer networks [53] and to cluster them based on topological approaches, and Bulauan et al. [54] clustered complex multilayer networks with topological approaches. Chen et al. [55] introduced an approach for measuring the classification boundary of a classifier by using a topological complexity, and Hofer et al. [56] developed topological constraint to improve the generalization performance of their model. Our method is categorized in the second area of research on topological data analysis. TDA helps improve our Random Forest quality and considerably increases the model's performance.

2.1.2 Mapper Formal Definition

In this thesis, we employ the Mapper method as introduced by [31, 32]. For a compact topological space $\mathcal{X}$ and a real valued function $f : \mathcal{X} \to \mathbb{R}$, the Mapper algorithm provides a general framework to study the topological changes in $\mathcal{X}$ with respect to the function f, which is commonly referred to as a *filter function* or *lens*. The choice of lens is crucial in Mapper construction as various lenses provide distinct insights on the data [31, 57]. In practice, $\mathcal{X}$ is mostly a point cloud in $\mathbb{R}^N$, and f is a function from $\mathbb{R}^N$ to $\mathbb{R}$ representing some domain information of the data. The output of the Mapper algorithm is the Mapper network, which is considered a meaningful summary of the data, representing clusters and relations between the clusters in the data.

To define the Mapper network, we need to explain the nerve of a cover. Let $\mathcal{X}$ be a point cloud in $\mathbb{R}^N$. A *cover* of $\mathcal{X}$ is a set of open sets in $\mathbb{R}^N$, $\mathcal{U} = \{U_i\}_{i=1}^m$ with $\mathcal{X} \subset \bigcup_i U_i$. The 1D nerve of $\mathcal{U}$ is a graph and is denoted as $\eta_1(\mathcal{U})$. Each node v_i in $\eta_1(\mathcal{U})$ represents a cover element U_i, and an edge exists between two nodes v_i and v_j if $U_i \cap U_j$ is nonempty for the corresponding cover elements. Figure 2.1a illustrates a case where $\mathcal{X}$ is a 2D point cloud and the cover $\mathcal{U}$ is formed by rectangles in the plane.

While it is possible to use multiple scalar functions, to keep the exposition clear, we describe the Mapper construction with a single scalar function $f : \mathcal{X} \to \mathbb{R}$. We start with a finite cover of $f(\mathcal{X}) \subset \mathbb{R}$ using intervals, that is, a cover $\mathcal{I} = \{I_k\}_{k=1}^n$ of $f(\mathcal{X}) \subset \mathbb{R}$ such that $f(\mathcal{X}) \subset \bigcup_k I_k$, see Figure 2.1. This induces a cover $\mathcal{U}$ of $\mathcal{X}$ by considering the clusters induced by points in $f^{-1}(I_k)$

for each k as a cover element. The 1D *Mapper network* of $(\mathcal{X}, f)$, denoted as Γ, is nothing but the 1D nerve of $\mathcal{U}$, i.e. $\Gamma_{\mathcal{X}} := \eta_1(\mathcal{U})$.

For a point cloud $\mathcal{X}$, several choices are to be made to compute the Mapper network $\Gamma_{\mathcal{X}}$. The first and most important one is the lens function $f : \mathcal{X} \to \mathbb{R}$. Another choice is the clustering method for the point cloud. Finally, there are two tuning parameters. The first one is called *resolution*, which is the number of intervals $\{I_k\}$ to cover $f(\mathcal{X})$. The second one is called *gain* which is the percentage of overlaps of these intervals. Note that increasing the resolution gives a finer summary by increasing the number of nodes in the Mapper network and makes the clusters smaller. On the other hand, increasing gain adds more relation (edges) between the nodes (clusters). In Figure 2.1, the resolution is 6, and the gain is the fixed intersection amount (for example, twenty percent) between the intervals I_k and I_{k+1}.

Note that the Mapper construction can be generalized to higher dimensions by choosing the lens function f as a multivariate function, i.e., $f : \mathcal{X} \to \mathbb{R}^d$ (for $d \geq 2$). In most cases, $d = 2$, and the resulting Mapper network is referred to as a *2D Mapper network*, where the corresponding cover elements of $\mathbb{R}^2$ become rectangles.

2.1.3 Illustrative Example of Mapper

We give a toy example for TDA Mapper networks in Figure 2.1 where for the point cloud $\mathcal{X}$ we use the height function $f : \mathcal{X} \to \mathbb{R}$ as a lens. We use 6 intervals $\mathcal{I} = \{I_1, \ldots, I_6\}$ to cover $f(\mathcal{X})$ (Figure 2.1b). By using the chosen clustering algorithm, we detect the clusters in each $f^{-1}(I_k)$ for each k (Figure 2.1a). These clusters form elements of a cover of $\mathcal{X}$. In particular, as illustrated in Figure 2.1a, $f^{-1}(I_2)$ induces two clusters of points that are enclosed by the blue rectangles, while $f^{-1}(I_1)$ produces a single cluster enclosed by an orange rectangle. The Mapper network in Figure 2.1c is the nerve of this cover, where each node represents the corresponding cluster and each edge represents that the clusters have nonempty intersections. The final Mapper network gives a rough summary or sketch of the whole point cloud [42]. In TDA Mapper graphs, nodes are associated with clusters of data points, and edges are drawn between nodes based on the overlap

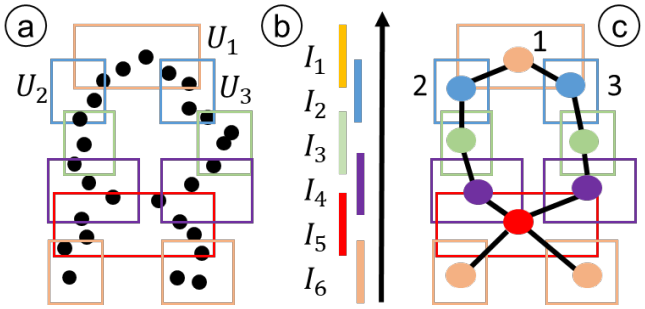


Figure 2.1: **Toy Example of Mapper.** For a point cloud $\mathcal{X}$ (a), we define a lens function $f : \mathcal{X} \to \mathbb{R}$ (b), and the induced covering defines a Mapper network where nodes represent clusters and edges represent related clusters (c).

or commonality of data points between the clusters. The edge weight reflects the strength of this connection, indicating how many data points are included in the overlapped area.

2.1.4 Integration with Machine Learning

Topological data analysis has moved beyond its theoretical foundations to become a practical component of modern machine learning. Its methods provide ways to extract global structure, reduce complexity, and improve robustness when working with challenging datasets. The following directions illustrate how TDA has been incorporated into learning pipelines.

- **Feature Extraction.** TDA methods generate representations that capture the shape and organization of data, serving as informative features for machine learning models or as tools for improving model design and interpretability [44]. The most widely used TDA approach for feature extraction is *persistent homology*, which produces *persistence diagrams* that summarize multi-scale topological features such as connected components, holes, and voids. These diagrams have been transformed into several machine-learning-compatible formats, including persistence landscapes [26], persistence images [25], and kernel-based embeddings for use in classifiers and regressors. Alternative scalar summaries such as total persistence [45], p-norms [46], and persistent entropy [47] provide compact quantitative characterizations of topological information. Rieck et al. [48] proposed *Betti curves* [49] to describe the evolution of topological features across filtration scales, while Chevyrev et al. [50] combined persistence diagrams with ensemble

and kernel-based models such as Random Forests and Support Vector Machines. Bubenik et al. [26] introduced persistent landscapes as functional summaries, and Zhao et al. [51] applied TDA-based descriptors for graph classification, showing the suitability of topological features for structured data analysis. Beyond persistent homology, the *Mapper* algorithm [36] offers an alternative TDA framework that constructs low-dimensional graph representations of high-dimensional datasets. These Mapper graphs can be encoded as structural features for supervised or unsupervised learning, capturing global data geometry that complements traditional local feature-based representations. Together, these methods demonstrate that TDA-derived features provide expressive, domain-agnostic representations that enrich the input space of machine learning models, improving both performance and interpretability across diverse learning tasks.

- **Robustness to noise.** A distinguishing property of persistence diagrams is their stability: small perturbations in the data lead only to small changes in the topological signature [58]. This robustness makes TDA particularly effective when learning from noisy, irregular, or incomplete datasets, where conventional features may be unreliable.

- **Handling sparsity and compression.** Many real-world datasets are large, sparse, or high-dimensional, which creates challenges for scalability. Mapper addresses this by compressing data into simplified graph structures that preserve the global organization of the dataset [36]. Such summaries not only reduce computational cost but also provide interpretable representations that can guide learning algorithms.

- **Integration into neural architectures.** Beyond preprocessing, TDA has been embedded directly into learning models through differentiable layers. Topological signature layers [59] and PersLay [27] allow persistence diagrams to be optimized within deep networks, enabling task-specific representations of topological structure. Recent approaches extend this idea by incorporating both persistent homology and Mapper into neural pipelines, showing benefits for sparse and high-dimensional data while raising open challenges related to scalability and efficiency [60, 61].

2.1.5 Applications and Future Potentials of TDA in Machine Learning

Topological data analysis has become a practical tool for addressing several challenges in modern machine learning. One important application is improving computational efficiency. Methods such as Mapper provide compressed representations of complex datasets, enabling faster training and inference while preserving global structural information [36, 38]. This ability to reduce dimensionality without discarding essential structure highlights the potential of TDA as a scalable preprocessing strategy.

Another contribution lies in enhancing model expressiveness. Persistence-based descriptors capture multi-scale connectivity and higher-order patterns that complement conventional feature extraction methods [25, 26]. By encoding structures beyond local statistics, TDA-derived features can enrich representations used by classifiers and regressors, leading to stronger predictive performance. Recent tutorials further emphasize how these methods can be adapted for diverse data types, including graphs, images, and time series [62].

TDA also plays a central role in robustness. Persistence diagrams are stable to perturbations, which makes them reliable for learning from noisy, irregular, or incomplete data [58]. This property is particularly valuable in domains where uncertainty and measurement error are unavoidable. Mapper-based reductions also contribute to robustness by producing interpretable low-dimensional summaries that remain informative under data variability [62].

A further potential lies in supporting multi-view and multi-scale learning. Since TDA methods summarize data at different resolutions, they naturally provide multiple complementary perspectives that can be integrated into unified models. This ability to combine topological summaries across views has been noted as a promising direction for improving adaptability and generalization [61, 62]. Overall, the applications of TDA extend beyond single domains. Its ability to improve efficiency, enrich representations, provide robustness, and enable multi-view modeling positions it as a versatile tool for future machine learning systems.

2.2 Graph Learning

In this section, we explore the taxonomy and key concepts of graph data, graph learning tasks, graph neural networks, and graph foundation models.

2.2.1 Graph Datasets

Graph data provides a robust framework for modeling complex relationships and interactions among entities in various domains. Unlike traditional data structures based on Euclidean spaces, graph data is inherently non-Euclidean, which allows it to represent complex structures with arbitrary connectivity and no fixed neighborhood geometry [63]. This flexibility makes graphs an ideal representation for a wide range of applications, from social networks and biological systems to transportation networks and recommendation systems.

In social networks, graph data is used to model relationships between individuals, capturing interactions and influence patterns [64]. In biological networks, nodes can represent molecules such as proteins or genes, while edges capture interactions or pathways, crucial for understanding cellular processes and drug interactions [65]. In transportation networks, nodes represent locations or intersections, and edges model routes or connections, allowing for the analysis of traffic patterns and optimization of routes [66].

Graphs can be classified into several overlapping types, each suited to different kinds of data and tasks:

- **Homogeneous graphs** have nodes and edges of the same type, simplifying modeling and analysis but potentially overlooking the complexity of certain datasets [67].
- **Heterogeneous graphs** incorporate multiple types of nodes and edges, enabling the modeling of complex systems where diverse entities and relationships coexist, such as in bibliographic networks and e-commerce systems [68].

- **Hypergraphs** permit edges to connect more than two nodes, capturing higher-order interactions and group relationships prevalent in many real-world networks [69].
- **Dynamic graphs** evolve over time with nodes and edges that may appear or disappear, crucial for modeling changes in networks like social or communication systems [70].
- **Temporal graphs** specifically focus on the timing and sequence of changes within dynamic graphs, making them ideal for analyzing patterns over time and forecasting future developments in dynamic systems.

The diverse nature of graph structures across different domains presents challenges in developing universal models that can effectively learn and generalize across various graph types and applications.

2.2.2 Graph Learning Tasks

Graph learning tasks have gained significant attention due to their wide applicability across various fields such as social networks, biology, chemistry, and more. These tasks can be generally classified based on the type of prediction they make, whether at the node level, edge level, or graph level, and the learning methods they use, including supervised, semi-supervised, and unsupervised learning. This section examines the various types of graph learning tasks and the paradigms employed to address them.

Table 2.1: Summary of graph learning tasks by prediction level, output type, and example applications.

Target	Output Type	Example Task
Node	Categorical	Node Classification
Node	Continuous	Node Regression
Edge	Categorical	Link Prediction, Edge Classification
Edge	Continuous	Edge Regression
Graph	Categorical	Graph Classification
Graph	Continuous	Graph Regression
Graph	Structural	Graph Property Prediction

Types of Graph Learning Tasks

Graphs can be analyzed at different levels of granularity, leading to various learning task formulations depending on the target of prediction. Broadly, these tasks are categorized into *node-level*, *edge-level*, and *graph-level* learning. Node-level tasks focus on inferring properties of individual

nodes, edge-level tasks involve understanding relationships between pairs of nodes, and graph-level tasks capture global properties or behaviors of the entire graph. The following subsections outline these categories in detail.

Node-Level Tasks.

- **Node Classification**: This task involves predicting the labels of individual nodes in a graph [67]. It is widely used in social network analysis [71], where nodes represent users and labels might denote categories such as interests or demographics. For instance, in a citation network, nodes represent papers, and node classification can help predict the research field of a paper based on its content and citation patterns [72].

- **Node Regression**: This task predicts continuous values for nodes. For instance, in a social network, node regression can be used to predict the influence score of users based on their interactions [72].

Edge-Level Tasks.

- **Link Prediction**: This task aims to predict the establishment of edges between nodes in a graph. It is beneficial in social networks, recommender systems, and biological networks. For example, in a social network, link prediction can suggest potential new friends, while in a biological network, it can predict interactions between proteins [72, 73, 74, 75].

- **Edge Classification**: This task involves classifying the type of relationships between nodes. It can be used in various scenarios, such as predicting the type of interaction between proteins or classifying the nature of a transaction between entities in a financial network [76, 77].

- **Edge Regression**: Similar to edge classification but involves predicting continuous values for edges. An example could be predicting the strength of interaction between two proteins [72].

Graph-Level Tasks.

- **Graph Classification**: This task involves assigning a label to an entire graph [78]. It is commonly used in chemistry and biology, where graphs represent molecular structures, and the task is to predict properties such as toxicity or activity. For example, predicting whether a molecule can act as a drug based on its graph representation [79, 80].

- **Graph Regression**: Similar to graph classification, but instead of predicting a categorical label, the task involves predicting a continuous value. This is useful in applications such as predicting the solubility of molecules or the strength of materials [72, 81, 82].

- **Graph Property Prediction**: This involves predicting properties that are intrinsic to the entire graph, such as the average clustering coefficient, graph diameter, or growth trends [17]. This is useful for understanding the overall structure and characteristics of the graph [83, 84].

Learning Paradigms and Task Settings in Graph Learning

Graph learning tasks can be categorized based on the prediction target, such as nodes, edges, or entire graphs, and the learning paradigms employed, namely supervised, semi-supervised, or unsupervised learning. Additionally, these tasks can be approached through inductive or transductive learning frameworks, which affect how models generalize to new data.

In supervised learning, models are trained on labeled data to predict specific outcomes. A key task in this paradigm is node classification, where the goal is to assign labels to nodes based on their features and the graph structure. This task can be performed in a transductive setting, where all nodes are known during training, or an inductive setting, where the model generalizes to unseen nodes or entirely new graphs. Another important task is edge prediction, also known as link prediction, which involves predicting the existence or attributes of edges between nodes. Graph classification and regression focus on predicting labels or continuous values for entire graphs. An example of this is molecular property prediction in cheminformatics, where models predict chemical properties of molecules to aid in drug discovery and materials research [84].

Semi-supervised learning leverages both labeled and unlabeled data, making it particularly ef-

fective in graph contexts due to the relational information inherent in graph structures. In node classification, for instance, the graph's connectivity can be utilized to propagate label information from labeled to unlabeled nodes, enhancing prediction accuracy even with limited labeled data [85]. This approach can significantly improve performance by capturing the underlying patterns in both labeled and unlabeled data. Edge and graph-level tasks also benefit from semi-supervised learning by incorporating unlabeled edges or graphs during training, which helps in refining the model's predictions and generalization capabilities.

In unsupervised graph learning, models aim to learn meaningful representations of the graph structure and node features without relying on manual labels. One fundamental method in this paradigm is graph representation learning, which focuses on learning low-dimensional embeddings for nodes, edges, or entire graphs that capture both structural and feature information. Traditional techniques like DeepWalk [86], node2vec [87], and Graph Autoencoders [88] generate these embeddings by exploiting random walks or reconstructing adjacency information, effectively capturing the essence of the graph's topology.

Another approach is contrastive learning, which employs contrastive objectives to maximize the agreement between different views or augmentations of the graph data. This method enables the model to learn invariant and discriminative features that are robust to changes in the input data [89]. Self-supervised learning creates surrogate tasks using the graph's inherent properties, such as predicting masked nodes or edges, context prediction, and clustering assignments [90]. By training models end-to-end on these proxy tasks, meaningful representations that are useful for downstream applications are learned without the need for manual labeling.

Graph clustering is another important task in unsupervised learning, where nodes or subgraphs are grouped based on similarity in their learned representations. This helps in uncovering community structures and functional modules within the graph, providing insights into the underlying organization of the data [91]. Anomaly detection involves identifying nodes, edges, or patterns that deviate significantly from the norm by modeling the typical structure and feature distributions within the graph. This is particularly important in applications such as fraud detection and network

security, where unusual patterns may indicate malicious activity.

Regarding learning frameworks, transductive learning involves training and evaluating the model on the same graph. The model learns embeddings specific to the nodes present during training and cannot directly generalize to new nodes or graphs without retraining. In contrast, inductive learning allows the model to learn transferable patterns from training graphs that can be applied to unseen nodes or entirely new graphs, enabling generalization without the need for retraining. This distinction is important for developing models that are robust and adaptable to new data.

Understanding these paradigms and task settings is crucial for developing graph foundation models that can capture universal patterns across diverse graph-structured data. By leveraging the strengths of each learning paradigm and selecting the appropriate framework between inductive and transductive approaches, researchers can develop powerful models that advance the field of graph learning.

2.2.3 Evaluation Metrics

The evaluation of temporal graph models in this thesis follows two complementary perspectives. The first examines the *predictive performance* of individual models on single temporal networks, while the second evaluates *cross-network generalization* and *relative ranking* when models are trained on multiple networks. Together, these evaluation schemes provide a consistent understanding of model accuracy, robustness, and transferability across both classification and regression tasks.

Single-Network Performance Metrics

In temporal graph learning, individual network performance is typically assessed using well-established classification and regression metrics that measure how effectively a model captures the evolving dynamics within a single temporal network. These measures are also widely used in traditional machine learning models for classification and regression tasks.

Area Under the ROC Curve (AUC). The AUC quantifies the ability of a classifier to distinguish between positive and negative samples [92]. It is widely adopted for evaluating classification models in both static and temporal settings. Formally,

$$\mathrm{AUC} = \Pr\Big(\hat{y}_i > \hat{y}_j \,\Big|\, y_i = 1,\, y_j = 0\Big),$$

where $\hat{y}_i$ denotes the predicted score and $y_i \in \{0, 1\}$ represents the ground truth label. A higher AUC value indicates a stronger discriminative capability of the model.

Mean Absolute Error (MAE). For regression-based prediction tasks, the MAE measures the average magnitude of prediction errors [92]:

$$\mathrm{MAE} = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i|,$$

where $\hat{y}_i$ and y_i denote the predicted and true values, respectively, and N is the total number of samples. Lower MAE values indicate higher regression accuracy and better model fit.

Average AUC and Average MAE. To summarize model behavior across several datasets, results are averaged over the set of temporal graphs $\mathcal{D}$:

$$\mathrm{AvgAUC}(m) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \mathrm{AUC}(m, d), \quad \mathrm{AvgMAE}(m) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \mathrm{MAE}(m, d).$$

These aggregate values provide a holistic view of how well a model performs across multiple independent temporal graph datasets.

Multi-Network and Ranking-Based Metrics

When temporal graph models are pre-trained or jointly trained on multiple networks, performance must be evaluated beyond individual predictive accuracy. In this context, ranking-based and comparative metrics are used to assess how consistently and effectively a model generalizes across

diverse networks.

First-Place Count. The first-place count measures how often a model achieves the top performance across all datasets. Let $\mathcal{D}$ denote the datasets and $\mathcal{M}$ the set of competing models. For dataset d, define

$$m^*(d) = \arg\max_{m \in \mathcal{M}} \mathrm{Perf}(m, d),$$

where $\mathrm{Perf}(m, d)$ is the task-specific performance metric (AUC for classification, MAE for regression). The first-place count of model m is then

$$\mathrm{First}(m) = \sum_{d \in \mathcal{D}} \mathbf{1}[m = m^*(d)].$$

This metric indicates how frequently a given method dominates others across datasets.

Average Rank. Each model is ranked by its performance per dataset, and the mean rank is computed as

$$\mathrm{AvgRank}(m) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \mathrm{rank}(m, d),$$

where smaller values correspond to more consistent high-ranking performance across multiple networks.

Relative Gain Across Tasks. Relative gain measures the improvement achieved by a multi-network model, such as Hydra, compared to the strongest baseline methods $\mathcal{B}$. It quantifies generalization strength across multiple unseen temporal networks.

For classification tasks (where higher is better):

$$\mathrm{Gain}_t = \frac{\overline{\mathrm{AUC}}_{\text{Hydra}}(t) - \max_{m \in \mathcal{B}} \overline{\mathrm{AUC}}_m(t)}{\max_{m \in \mathcal{B}} \overline{\mathrm{AUC}}_m(t)} \times 100\%.$$

For regression tasks (where lower is better):

$$\mathrm{Gain}_t = \frac{\min_{m \in \mathcal{B}} \overline{\mathrm{MAE}}_m(t) - \overline{\mathrm{MAE}}_{\text{Hydra}}(t)}{\min_{m \in \mathcal{B}} \overline{\mathrm{MAE}}_m(t)} \times 100\%.$$

Finally, the average gain across a set of tasks $\mathcal{T}$ is computed as

$$\mathrm{Gain}_{\mathcal{T}} = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \mathrm{Gain}_t.$$

This measure provides a quantitative assessment of cross-network transferability and improvement, capturing how well a temporal graph model benefits from large-scale multi-network training.
These evaluation metrics are used throughout this thesis to compare temporal graph learning models and traditional classification-based machine learning models under both single-network and multi-network experimental settings.

2.2.4 Graph Neural Networks

Graph neural networks have emerged as powerful tools for addressing challenges associated with graph-structured data. GNNs offer scalable solutions that traditional neural networks, designed for grid-like data, could not. The pioneering work on GNNs introduced a method to view nodes as feature vectors (embeddings), a feature aggregation method, and an update function [93]. The development of GNNs revolutionized data analysis, expanding the applicability of neural networks to complex, real-world graph data.

Geometric Priors for GNNs

Graph Neural Networks are guided by several geometric priors that allow them to capture the structural properties of graphs effectively. First, **permutation invariance/equivariance** ensures that the node ordering in a graph does not affect the model's output, treating all nodes symmetrically. This is critical for respecting the arbitrary node orderings in graph data. Second, **locality** plays a central role, as GNNs are built on the assumption that nodes are primarily influenced by their local

neighbors, reflecting real-world structures such as social networks and molecular graphs. Another important prior is **homophily**, which assumes that connected nodes tend to share similar features or labels. This allows GNNs to aggregate information more effectively, leveraging the similarity between neighboring nodes. Additionally, **positional/structural encoding** is employed to capture the relative positions or roles of nodes within a graph, addressing the lack of a natural coordinate system in graph data. Techniques like Laplacian eigenvectors or shortest-path distances can be used for this purpose. Moreover, many graphs have **multi-scale/hierarchical structures**, where information exists at different levels of granularity. GNNs that respect this prior can aggregate information from local to global scales, capturing relationships across clusters or communities. **Substructure awareness** or motif recognition is also important, as many graphs contain repeating patterns or subgraphs, such as triangles or cliques. GNNs that can detect and leverage these motifs can capture higher-order relationships in the graph. Lastly, the **continuity** prior ensures that the embeddings of nodes vary smoothly across the graph, promoting consistency within clusters or communities. Regularization techniques are often used to enforce this smoothness, leading to more coherent node embeddings. Together, these geometric priors enable GNNs to generalize across different graph structures and tasks, making them a powerful tool for graph representation learning.

As GNNs have developed, more use cases have emerged, proving them as powerful tools in fields including social networks [72, 94], communication networks [72], traffic [95], and biology [96]. Their profound ability to predict various graph learning tasks has led to further development of GNNs. The advent of Temporal GNNs has expanded their capabilities, establishing GNNs as essential tools for analyzing dynamic data [10].

Types of Graph Neural Networks

Graph Neural Networks can be categorized into two main types: spectral and spatial methods.

- **Spectral-Based GNNs:** Spectral-Based GNNs perform convolutions in the spectral domain by utilizing the graph Laplacian matrix and Fourier transforms, making them well-suited for tasks

such as graph classification. They leverage the spectral properties of graphs, often through eigen-decomposition techniques, but face challenges with high computational complexity and scalability, especially for large graphs. They predominantly apply to undirected graphs, reducing their uses in real-world graph datasets that often include directed edges. Prominent spectral-based GNNs include Spectral Convolutional Neural Networks and ChebNet [97, 98].

- **Spatial-Based GNNs:** Spatial-Based GNNs define convolutions based on nodes' neighborhoods, operating on graphs in the spatial domain using message-passing mechanisms. This approach makes them particularly adept at handling large-scale graphs. These networks iteratively gather and update information from neighboring nodes, resulting in richer node representations that effectively capture relational and structural information, proving particularly useful in social network analysis. Popular spatial-based GNN architectures include Graph Convolutional Networks, GraphSAGE, and Graph Isomorphism Networks [97, 99].

- **Message Passing GNNs (MPNNs):** Message Passing Neural Networks (MPNNs) are a foundational subset of spatial-based GNNs. They propagate node features through iterative information exchanges between neighboring nodes, known as the message-passing mechanism (see Figure 2.2). Each node updates its state throughout the message-passing process based on aggregated features from its local neighborhood, leveraging aggregation functions such as sum or average to capture the local graph structure [100, 101]. Through this iterative process, GNNs can achieve lower computational complexities than spectral approaches and the ability to process directed graphs [102].

Prominent Graph Neural Network Architectures

- **Graph Convolutional Networks (GCN):** GCNs are one of the pioneering architectures in graph neural networks. Introduced by Kipf and Welling in 2017 [104], GCNs extend convolutional neural networks to graph-structured data by aggregating node features through convolutional layers, capturing both node attributes and the graph's topology. Originally derived from spectral

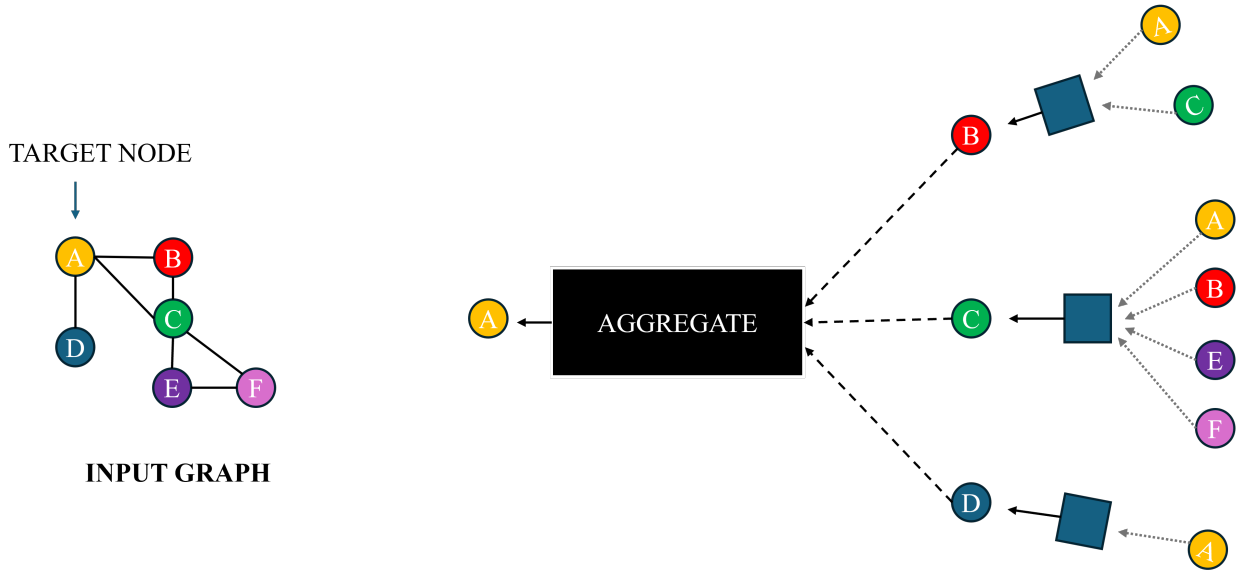


Figure 2.2: An illustration of the message-passing mechanism in Graph Neural Networks. In this process, node A aggregates information from its neighboring nodes by receiving messages that contain their features and structural information. These messages are combined using an aggregation function (e.g., sum, mean, or max), allowing node A to update its representation based on its local neighborhood. This mechanism enables node A to capture the influence of its immediate surroundings, effectively learning a representation that reflects its own attributes and the context of connected nodes. Source: Hamilton et al. [103].

graph convolutions, GCNs define convolution operations in the spectral domain using the eigenvalues and eigenvectors of the graph Laplacian. To make GCNs computationally efficient and scalable to large graphs, the authors approximated the spectral convolution using a first-order approximation. This simplification led to a convolution operation that aggregates information from a node's immediate neighbors. Despite their spectral origins, GCNs are often implemented as spatial methods due to this local aggregation. GCNs are known to be computationally efficient and scalable to large graph datasets but can suffer from over-smoothing and do not handle multidimensional edge features well [104].

- **Graph Attention Networks (GAT):** The GAT architecture uses attention mechanisms to assign importance values to nodes within a neighborhood, employing multi-head attention for more complex data patterns. Although adaptable and highly efficient, GATs may struggle with noisy data and require careful hyperparameter tuning [105].

- **Graph Isomorphism Network (GIN):** GINs employ a sum-based aggregation function for node and graph information, followed by a multi-layer perceptron (MLP) to capture structural information. GINs effectively differentiate between graph structures, being at most as powerful as the Weisfeiler-Lehman graph isomorphism test. [106] However, despite being a spatial-based GNN, they suffer from high computational complexities and low scalability to large graphs [107].

2.2.5 Temporal Graph Learning

Temporal graph learning focuses on modeling and understanding graphs whose structures, nodes, and edges change over time. Unlike static graph learning, which assumes a fixed network topology, temporal graph learning captures both the spatial relationships among connected entities and the temporal dynamics that describe how these connections appear, disappear, or evolve. This approach enables the analysis and prediction of real-world systems such as social interactions, financial transactions, biological processes, and communication networks, where relationships are inherently time-dependent. Temporal graphs are generally categorized into two main types, Discrete Time Dynamic Graphs (DTDGs) and Continuous Time Dynamic Graphs (CTDGs), which differ in how they represent and process temporal information, as explained in the following section.

Discrete Time Dynamic Graphs

Discrete time dynamic graphs represent temporal networks as a sequence of discrete graph snapshots observed at fixed time intervals. Formally, a DTDG is defined as

$$\mathcal{G} = \{G_1, G_2, \ldots, G_T\},$$

where T denotes the total number of discrete time steps. Each snapshot $G_t = (V, E_t, X_t)$ represents the state of the graph at time t, where V is the set of nodes (assumed fixed or slowly varying), $E_t \subseteq V \times V$ is the set of edges observed at time t, and X_t denotes the node and edge attributes at

that time step. As time evolves, edges (and occasionally nodes) may appear or disappear, reflecting changes in the underlying system [108, 109, 110, 111, 112].

This discrete formulation models temporal evolution as a sequence of transitions between consecutive snapshots. Each G_t captures the structural and attribute information for a given period, and the transition from G_t to G_{t+1} encodes the temporal dependency between successive states. DTDGs are particularly effective in domains where data are naturally collected or aggregated at regular intervals, such as social interaction graphs, financial transaction networks, transportation systems, or sensor-based infrastructures. By treating time as a series of ordered steps, DTDGs provide a structured and computationally efficient framework for capturing temporal regularities while maintaining compatibility with conventional graph representation learning approaches.

Continuous Time Dynamic Graphs

Continuous time dynamic graphs extend the temporal graph framework to settings where interactions occur asynchronously in continuous time. Instead of discrete snapshots, a CTDG models graph evolution as a sequence of time-stamped events that update the network state. Formally, a CTDG is defined as

$$\mathcal{G} = (V, E, \mathcal{T}),$$

where V is the set of nodes, $E \subseteq V \times V$ is the set of possible edges, and $\mathcal{T}$ denotes the set of observed temporal events. Each event is represented as a tuple $(u, v, t, x_{uv}, x_u, x_v)$, where $u, v \in V$ are interacting nodes, $t \in \mathbb{R}^+$ is the timestamp of the interaction, x_{uv} represents the event or edge features, and x_u, x_v denote the features of nodes u and v, respectively [113, 114, 115, 116, 117, 118].

In this continuous formulation, the graph evolves through event-driven updates, where new interactions modify the topology or attributes of the network. This representation allows the modeling of fine-grained temporal dynamics and irregular interaction patterns that cannot be captured by fixed time intervals. CTDGs are well suited to systems where relationships emerge asynchronously, such as communication networks, online social platforms, streaming financial markets, and bio-

logical interaction systems. By maintaining temporal continuity, CTDGs enable precise reasoning about the timing, duration, and frequency of events, offering a high-resolution perspective on the evolution of dynamic graphs.

2.2.6 Temporal Graph Neural Network

Temporal Graph Neural Networks (TGNNs) extend traditional graph neural networks to model systems in which the graph structure, node attributes, or edge features evolve over time. Unlike static GNNs that process a fixed topology, TGNNs incorporate temporal dependencies into their message-passing mechanisms, allowing them to capture both spatial and temporal dynamics within a unified framework [114, 116, 110, 108, 119]. This temporal modeling ability makes TGNNs suitable for a wide variety of tasks, including link prediction, node classification, dynamic representation learning, and event forecasting in evolving networks. Early TGNN architectures were inspired by the success of recurrent and convolutional neural models. Recurrent-based approaches integrate time through recurrent units such as GRUs or LSTMs, enabling temporal memory while preserving structural context at each timestamp [111, 112]. Examples include Temporal Graph Convolutional Networks (T-GCNs) [120] and EvolveGCN [110], which update node embeddings over time by learning temporal transitions of graph convolutional parameters. Convolutional variants extend spatial convolutions with temporal filters to jointly model local dependencies in both space and time. Attention-based TGNNs further improve expressiveness by dynamically weighting historical interactions. Models such as Temporal Graph Attention Networks and HTGNs [119] leverage attention mechanisms to focus on relevant temporal neighbors, enabling the network to adaptively capture long-range dependencies in both topology and sequence. Continuous-time approaches incorporate temporal point processes or time-aware message passing to handle asynchronous interactions [113, 114, 115, 117, 118], while discrete-time methods rely on ordered graph snapshots to model sequential transitions [108, 109]. TGNNs have been successfully applied across diverse domains where relational structures evolve in time. In finance, they are used for live fraud detection and temporal anomaly monitoring [121]; in transportation, they enable real-time traffic

forecasting and flow optimization [120]; and in recommender systems, they support adaptive user-item interaction modeling [122]. Beyond applications, theoretical analyses have also examined the expressive power of TGNNs. Several studies extend the classical Weisfeiler–Lehman framework [107, 123, 124] to temporal settings [109, 116], revealing relationships between temporal message passing and graph isomorphism tests. These analyses highlight both the strengths and limitations of TGNNs in capturing dynamic structures and emphasize the need for architectures capable of reasoning over evolving temporal and structural patterns. TGNNs represent a critical step toward modeling real-world dynamic networks by unifying temporal reasoning and graph representation learning. Through recurrent, convolutional, and attention-based mechanisms, they provide the foundational machinery for understanding temporal dependencies in complex relational data.

Prominent Temporal Graph Neural Network Models

TGCN [125] is a combination of GCN and GRU. In particular, GCN is used to learn complex topological structures, while GRU is used to model embedding dynamically to capture temporal dependence.

HTGN [126] leverages the power of hyperbolic geometry, which is well-suited for capturing hierarchical structures and complex relationships in temporal networks. HTGN maps the temporal graph into hyperbolic space and utilizes hyperbolic graph neural networks and hyperbolic gated recurrent neural networks to model the evolving dynamics. It incorporates two key modules that are hyperbolic temporal contextual self-attention (HTA) and hyperbolic temporal consistency (HTC)- to ensure that temporal dependencies are effectively captured and that the model is both stable and generalizable across various tasks.

GraphPulse [17] addresses the challenge of learning from nodes and edges with different timestamps, which many existing models struggle with. It combines two key techniques: the Mapper method from topological data analysis to extract clustering information from graph nodes and Recurrent Neural Networks (RNNs) for temporal reasoning. This principled approach helps

capture both the structure and dynamics of evolving graphs.

GCLSTM [127] combines a graph convolutional network and Long Short-Term Memory (LSTM) units to handle both the structural and temporal aspects of evolving networks. The GCN is used to capture the local structural properties of the network at each snapshot, while the LSTM learns the temporal evolution of these snapshots over time.

EvolveGCN [128] is designed to capture the temporal dynamics of graph-structured data. Instead of relying on static node embeddings, EvolveGCN evolves the parameters of a graph convolutional network over time. By using a recurrent neural network to adapt the GCN parameters, this model is capable of dynamically adjusting during both training and testing, allowing it to handle evolving graphs, even when node sets vary significantly across different time steps.

ROLAND [108] is a dynamic graph learning framework that models node representations as hierarchical states, updated recurrently to capture temporal dependencies in evolving graphs. It supports scalable training using techniques like truncated backpropagation through time and meta-learning. In our DTDG setting, we use ROLAND to benchmark its performance and adaptability across diverse temporal networks.

WinGNN [129] uses a simple GNN to model topological information from the graph as other models existing in the literature. However, to model temporal dependencies, WinGNN proposes a novel mechanism of random gradient aggregation and meta learning strategy. In particular, WinGNN computes the frame-wise loss of the current snapshot and passes the loss gradient to the next to model graph dynamics without using RNN-based modules. Then it introduces the randomized sliding-window to acquire the windowaware gradienton consecutive snapshots, and the calculated two types of gradient are aggregated to update the GNN modules.

2.3 Graph Foundation Models

Graph Foundation Models (GFMs) represent an emerging direction that applies the foundation model paradigm to graph-structured data [130]. GFMs aim to pre-train on extensive graph datasets

and adapt to various graph-related tasks. Similar to foundation models in NLP and CV, Graph Foundation Models are characterized by a unified architecture, transferable representations, pre-training and adaptation, homogenization, and emergent abilities. Like other foundation models, GFMs are expected to exhibit emergence and homogenization capabilities [130].

Definition. A graph foundation model is a unified neural network architecture designed to learn and process graph-structured data in a generalizable and transferable manner to any new, previously unseen graph [131].

Challenges: Graph Foundation Models face several unique challenges that are not as prominent in foundation models for natural language processing or computer vision. One of the key challenges is the **non-Euclidean nature of graph data**, which lacks the regular structure of text or images, making it difficult to apply traditional deep learning techniques. Additionally, GFMs must deal with the **variety of graph types**, such as homogeneous, heterogeneous, dynamic, and hypergraphs, requiring the model to generalize across diverse structures. Unlike NLP and CV, graphs do not come with a natural **positional encoding**, forcing GFMs to generate or infer positional information using techniques like shortest-path distances. Moreover, many graphs exhibit **multi-scale and hierarchical structures**, necessitating models that can capture both local and global patterns simultaneously. GFMs must also account for **higher-order relationships and motifs**, which are prevalent in graphs but less so in NLP or CV, requiring the model to capture complex substructures like triangles or cliques. Another challenge is **heterophily**, where connected nodes may be dissimilar, making aggregation of neighboring information more difficult. Additionally, the **size variability** of graphs, ranging from small molecular graphs to massive social networks, demands models that can scale efficiently. Unlike NLP and CV, there are fewer **standardized benchmarks and evaluation metrics** for graphs, complicating the assessment of model performance.

Emergence

Emergence refers to the spontaneous appearance of novel capabilities in a model as its scale increases, either in terms of the number of parameters or the amount of training data. These capa-

bilities are not explicitly programmed into the model but rather arise from the complexity of the system. In the context of artificial intelligence models, such as large language models or graph foundation models, emergence occurs when certain behaviors or properties become evident only after the model surpasses a critical scale.

Formally, emergence can be characterized by the following conditions:

- **Model Size:** Emergent behavior becomes apparent when the model contains a large number of parameters, allowing for deeper layers of abstraction.
- **Training Data:** The model must be trained on an extensive and diverse dataset to capture the complexity required for emergent properties.
- **Scale Effects:** Emergence typically manifests when the model's parameters or training data surpass a threshold, leading to qualitatively different behaviors. For instance, zero-shot learning, where a model performs tasks without explicit training, is an example of emergent behavior.

In the context of GFMs, examples of emergent abilities might include graph reasoning or zero-shot graph generation, which only become feasible when the model is sufficiently large and complex [131, 132]. These capabilities highlight the model's ability to generalize and adapt across various downstream tasks, exhibiting properties that were not explicitly engineered.

2.3.1 Model Families in Graph Foundation Learning

There are three main approaches to implementing GFMs. GNN-based models use graph neural networks as the backbone architecture (see section 2.2.4), leveraging their ability to process graph-structured data [130, 133, 88]. They typically employ contrastive or generative pre-training strategies. On the other hand, LLM-based models transform graph data into a format suitable for large language models, either through tokenization or natural language descriptions [134, 135]. It leverages the powerful text-processing capabilities of LLMs. Lastly, Hybrid-based models combine graph neural networks and language models to leverage the strengths of both architec-

tures [136, 137]. They aim to capture both structural information from graphs and semantic information from associated text. GFMs hold significant promise for enhancing the expressive power, transferability, and applicability of graph models to more complex data and tasks [130]. However, significant research challenges remain in designing effective architectures, pre-training strategies, and adaptation methods for graph foundation models. GFMs hold significant promise for enhancing the expressive power, transferability, and applicability of graph models to more complex data and tasks [130]. By moving beyond single-network supervised training toward large-scale pretraining and cross-domain adaptation, these model families attempt to bring the paradigm of foundation models to graph learning. However, significant research challenges remain in the pre-training strategies, and adaptation methods for graph foundation models.

Pre-Training Phase

Pre-training GNNs is an essential phase in building models capable of excelling across diverse graph-based tasks. The fundamental concept of pre-training is to leverage unlabeled data to develop a model that learns comprehensive representations of graph structures and relationships. This involves training the GNN to discern and encode the underlying patterns and semantics of the graph data, leading to the creation of embeddings that effectively capture key properties of nodes and graphs. By establishing a solid foundation through pre-training, these models can be fine-tuned for specific tasks or applied to new, unseen graphs with improved performance. This approach minimizes the need for extensive retraining and allows for more efficient adaptation to various tasks and datasets. In general, there are two main approaches for pre-training graph-based models to enable their adaptation for various tasks: supervised learning and self-supervised learning.

Supervised pre-traning In the supervised method of pre-training foundation models, a large amount of graph data, along with associated labels, is used to train a model on specific tasks such as graph-level, link-level, or node-level tasks. This approach helps the model learn to perform inferences on new, unseen network structures within the same task domain. One key benefit of

this method is its potential for task transferability, which allows the trained model to be adapted for similar tasks or new tasks that are closely related to the original ones. This adaptability can significantly streamline the fine-tuning process. However, effective task transfer requires careful alignment of the task formulations. For example, Jin et al. [138] showed that using link prediction directly as a pretext task could result in negative transfer when applied to node classification. On the other hand, positive transfer was achieved by reconfiguring node classification as a link prediction problem, where the class membership of a node is interpreted as the probability of a link between the node and label nodes. Also, further advancement has been made by proposing a sub-graph view that reformulates node classification into an ego graph classification format, thus improving transferability across tasks. [139, 140]

Self-supervised pre-traning Self-supervised learning offers powerful methods for pre-training Graph Neural Networks without relying on labeled data. Among these methods, contrastive pre-training and generative pre-training stand out for their ability to extract meaningful graph representations.

Contrastive Pre-Training focuses on learning invariant features by maximizing mutual information between different views of the same graph. This approach encourages the model to distinguish between varied representations of a graph while ensuring that the underlying semantic information remains consistent across these variations. By training the model to recognize and preserve essential features despite different perspectives, contrastive learning helps the GNN generalize effectively to new, unseen tasks. This invariance is crucial for transferring the learned knowledge to diverse applications without the need for task-specific labels [89, 141].

Generative Pre-Training, on the other hand, involves teaching the GNN to understand and reconstruct the overall structure and attributes of graphs. This approach includes methods like graph reconstruction, where the model learns to rebuild graphs from encoded data. By doing so, it captures both the structural and attribute-based semantics of the graphs, creating a robust foundation for various downstream tasks. Generative pre-training supports task versatility by equipping the model with comprehensive graph representations that can be fine-tuned for specific applica-

tions [88]. These self-supervised methods enhance the adaptability of GNNs, making them well-suited for a range of tasks by providing a strong, generalized understanding of graph data.

Adaptation Strategies

Once a GFM is pre-trained, adapting it to new tasks or data involves several strategies that leverage its learned representations. These adaptation methods ensure that the model can efficiently handle diverse and evolving graph-related tasks. Key strategies for adapting foundation models include fine-tuning, zero-shot learning, few-shot learning, and prompt-based methods.

Task-specific fine-tuning is a widely used adaptation strategy where a pre-trained GFM is further trained on a task-specific dataset. This process involves adjusting the model's parameters to optimize performance for the new task while retaining the knowledge gained during pre-training. Fine-tuning allows the model to specialize in the nuances of the new task, leveraging the foundational knowledge acquired previously. This approach is highly effective when there is sufficient labeled data available for the target task. It is particularly useful for tasks that are closely related to the original tasks the model was trained on [131].

Domain-Specific fine-Tuning is another strategy where the pre-trained model is further refined using domain-specific data. This involves tailoring the model to specific applications or industries, such as social networks, biological data, or financial systems. By incorporating domain-specific knowledge into the fine-tuning process, the model can better understand and respond to the unique characteristics of the target domain, improving performance on specialized tasks [131]

Zero-shot learning enables the model to perform tasks without any additional training on task-specific data. This approach relies on the model's ability to generalize from its pre-trained knowledge to new, unseen tasks. In zero-shot settings, the model applies its learned representations to make predictions or infer patterns in new contexts based on descriptions or semantic cues provided during inference. This method is valuable when labeled data for the new task is scarce or unavailable, allowing the model to leverage its broad understanding of graph structures to address novel problems [142].

Few-shot learning involves adapting the pre-trained model with only a small number of labeled examples from the new task. The model leverages its pre-existing knowledge to make accurate predictions despite limited data. Few-shot learning methods often use techniques such as meta-learning or transfer learning, where the model quickly adapts to new tasks by fine-tuning on a few examples. This strategy is effective when acquiring labeled data is costly or impractical, providing a way to efficiently adapt the model with minimal additional training [143].

Prompt-based methods involve using textual or structured prompts to guide the model's behavior for specific tasks. In the context of graph models, prompts can be formulated as queries or instructions that direct the model to perform certain operations or provide specific types of insights. This approach capitalizes on the model's pre-trained knowledge to interpret and respond to prompts in a task-relevant manner. Prompt-based methods are particularly useful for quickly adapting the model to new tasks by leveraging natural language or structured prompts to specify desired outcomes [144].

These adaptation strategies enhance the versatility of foundation models, allowing them to effectively address a wide range of graph-related tasks and applications.

Effectiveness of the GNN Backbone

GNN-based graph foundation models have demonstrated effectiveness across a wide range of graph-learning tasks due to their inherent ability to efficiently process graph-structured data. This efficiency stems from GNNs' capacity to capture complex relationships between nodes with relatively low computational costs, making them strong candidates for developing graph foundation models.

Despite their promise, the GNN backbone faces limitations, particularly in processing textual data. In Text-Attributed Graphs (TAGs), natural language attributes provide critical context for understanding a graph's structure and functionality. GNNs typically struggle with integrating this crucial information, leading to a loss of valuable context in graph understanding tasks. Moreover, GNN-based models are often trained in narrow domains, limiting their ability to generalize across

different typse of graphs, reducing their performance in graphs that require a broader contextual understanding. To address these shortcomings, LLM-based models can improve performance in many graph-learning tasks by enabling models to handle textual attributes and improving generalization across domains.

LLM Models

Large language models have gained much traction due to their natural language processing prowess. Researchers have explored leveraging LLMs for graph data by expressing graph-structured data through natural language descriptions. This approach promotes flexibility, scalability, and compatibility in various graph-learning tasks. [145] However, LLMs pose unique challenges in representing graph information and modifying existing LLMs for accurate predictions. Additionally, LLMs' input length limitation can result in loss of graph information. Although it has little analysis on graph-learning tasks, the NLGraph benchmark [146] provides a comprehensive analysis of LLMs in graph-reasoning tasks such as shortest path and maximum flow. Recent models such as GPT4Graph [147], InstructGLM [145] and GIMLET [148] have proven the potential that LLMs have in various graph-learning tasks.

Preparing Graph Data

Given that LLMs are not designed to process graph-structured data, it poses a unique challenge of how to provide graph-structured information to the model. Two main approaches address this:

- **Graph-to-token:** Graph-to-token converts graph information into tokens to fit the traditional transformer model input format. This approach was first proposed in GIMLET [148] with generalized position embeddings for molecule prediction tasks. InstructGLM [145] creates a unique token for each node, setting the feature vector as the embedding. These approaches typically use the LLM's pre-trained word token embeddings to handle connectivity. Although structural information is preserved, conveying underlying graph structures remains a challenge.

- **Graph-to-text:** Graph-to-text describes graphs using exclusively natural language. Graph-Text [149] proposes the "graph-syntax tree" as a method to describe graph attributes and structure using natural language. Other approaches have utilized edge lists to describe graph structures, such as in LLMtoGraph. [135] TextForGraph [150] designed full text and reduced text prompts to describe graph information while reducing prompt length effectively. Despite their promise, graph-to-text methods are not wholly effective and struggle to capture underlying graph structures.

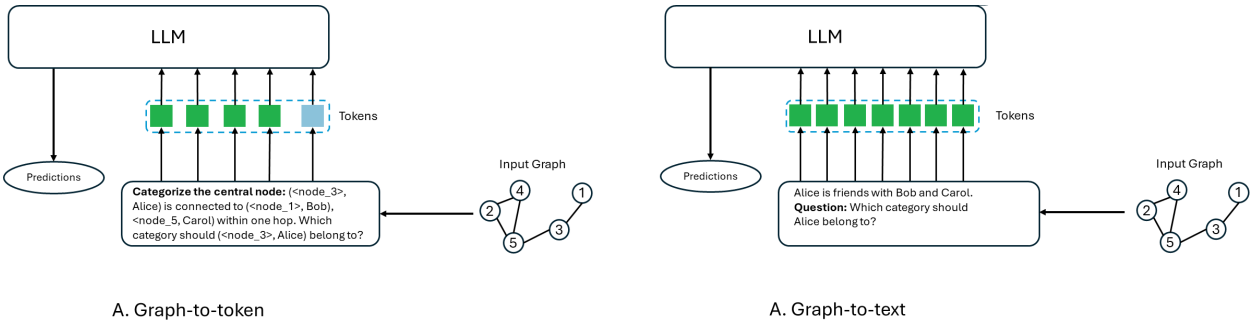


Figure 2.3: Diagram illustrating the pipeline to prepare graph-structured data for LLM processing. Graph-to-token transforms data to unique node tokens (seen as blue tokens) with text tokens (seen as green tokens) used to represent connectivity. Graph-to-text transforms graph data to only natural language using text tokens only (seen as green tokens).

Pre-Training Phase

To improve LLMs' capabilities in graph-learning tasks, they are pre-trained on vast amounts of data using either Language Modeling or Masked Language Modeling.

- **Language Modeling (LM):** This common self-supervised task in natural language processing is widely employed by many state-of-the-art LLMs, such as GPT-3. LM typically refers to auto-regressive LM or unidirectional LM, which involves estimating probability distributions of the next word. A drawback of unidirectional language modeling is that the contextual information of each token is based solely on the preceding tokens and the token itself. Many models, such as LLMtoGraph [135], TextForGraph [150], and GraphText. [149]

- **Masked Language Modeling (MLM):** MLM was designed to address limitations in unidirectional language modeling. After randomly masking tokens in a given input, the model predicts the masked tokens by analyzing contextual information in surrounding text. Subtypes of MLM include Sequence-to-Sequence MLM (Seq2Seq MLM), which uses masked sequences as input for a neural encoder, and Enhanced MLM (E-MLM), which integrates external knowledge into MLM or extends the mask prediction task to language modeling tasks. [131]

Adaptation Strategies

To increase LLMs' ability to understand graph data, graph-to-token and graph-to-text methods are accompanied by adaptation techniques to improve prompts.

- **Manual Prompting:** To form a comprehensive prompt, manual prompting methods utilize graph representations to prompt a LLM for desired outputs. The Graph Description Language (GDL) syntax provides a way to define graph structures and perform queries and operations on graphs. [147] This approach, among others such as change-of-thoughts prompting [151] promote more rich representations of graph information.

- **Automatic Prompting:** For graphs with complex structures and relationships, automatic prompting generates more context or eliminates irrelevant information from the input. GPT4Graph employs prompts generated by the LLM through graph summarization, exploration, and completion. [147] This approach enhances efficiency by providing more detailed and relevant information to the LLM for graph-learning tasks.

Effectiveness of the LLM Backbone

Employing LLMs as independent predictors has shown they can perform graph-learning tasks. However, they still require further development to match the predictive power of GNN models. Despite demonstrating strong preliminary effectiveness, data loss and inaccurate predictions still need to be addressed. [152] Over time, as more experiments are conducted, new approaches to uti-

lizing LLMs as backbone architectures for graph-learning tasks are likely to become more efficient, effective, and accurate.

Hybrid Models

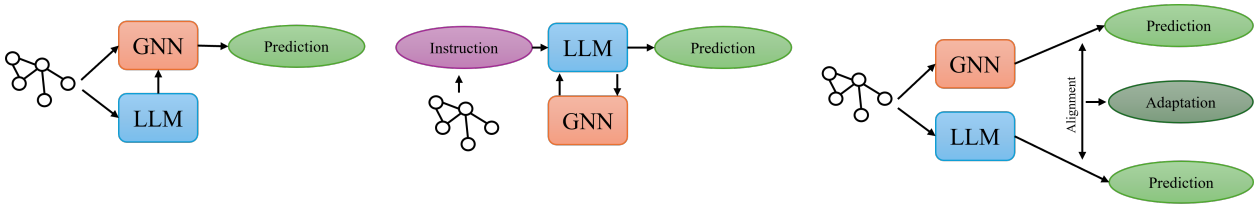
GNN-based models excel at processing graph-structured data but often struggle with text-attributed graphs and cannot effectively interpret natural language instructions. Meanwhile, LLM-based models show promise in graph-reasoning tasks using natural language but have difficulties with higher-level reasoning and mathematical computations. Therefore, to address limitations from both approaches, researchers have been prompted to integrate GNNs and LLMs, aiming to create more accurate and robust models.

Integrating GNN and LLM

To successfully combine the strengths of GNNs and LLMs, a framework is needed to determine how information moves and how each part is utilized.

- **GNN-centric Methods:** GNN-centric models allow the GNN to be the forefront, using LLMs to process textual information. Some models such as GraD [153] utilize LLMs that provide usable node embeddings, creating a GNN model that utilizes node embeddings from the LLM as input. Other GNN-centric approaches [154, 140] utilize LLMs to augment, enhance, or describe graph data, enriching the inputs for the GNN and allowing it to leverage TAGs effectively.
- **LLM-centric Methods:** LLM-centric models primarily depend on an LLM to process information and use a GNN to enhance their performance. This poses some limitations in that they struggle to process mathematical computations and advanced graph reasoning tasks. Models take different approaches as to how they use a GNN to support their LLM, but most [155, 156] use the GNN to provide context for underlying graph structure.
- **Symmetric Methods:** These models treat GNNs and LLMs equally important, unifying them to create a model that learns from graph structures and textual information. For example,

Text2Mol [157] uses an LLM to process molecule descriptions, while a GNN learns the molecular structure. Similarly, CLAMP [158] uses Graph-Text Contrastive Learning (GTCL) to align embeddings in a shared latent space. These approaches create a model with a richer understanding of both graph and text data.



(a) GNN-Centric Methods (b) LLM-Centric Methods (c) Symmetric Methods

Figure 2.4: Diagrams illustrating the general flow of GNN-centric, LLM-centric, and Symmetric hybrid methods

.

Pre-Training Phase

To effectively handle both graph and text-based information, the model must be pre-trained on large datasets that include both graph and text data. Pre-training strategies are generally categorized into GNN or LLM-based and alignment-based methods.

- **GNN or LLM-based:** In this approach, the GNN and LLM components are trained separately before being integrated. GNNs are typically pre-trained on graph-learning tasks to learn graph structures. For example, GALM [159] performs graph reconstruction to learn graph information. Meanwhile, LLMs use techniques such as masked language modeling, as seen in models like GraD [153] and GraphFormers [160], to learn textual information. Other models may use language modeling, as in GLEM [161] and ReLM [155] or Text-Text Contrastive Learning (TTCL), such as SimTeG [162].

- **Alignment-based:** Symmetric pre-training methods balance the role of GNNs and LLMs to provide an equal representation of data. Some models, like Text2Mol[157] and CLAMP[158],

employ Graph-Text Contrastive Learning (GTCL) to align graph and text embeddings in a shared latent space, facilitating richer comprehension of both data types. Other models such as GIANT[163] and TextGT[164] generate multimodal embeddings that unify the embedding spaces, capturing relationships and information from graph and text data.

Adaptation Strategies

To optimize GNN+LLM-based models for graph-learning tasks, different adaptation strategies can be employed:

- **Fine-tuning:** There are various approaches to fine-tuning; each aims to adjust the model to be as efficient and accurate as possible. Vanilla fine-tuning methods adjust many parameters at a time, a computationally intensive process for many models [165, 166, 167]. More efficient approaches such as Text2Mol [157] and CLIP [168], choose to align the embedding space of GNNs and LLMs, while others, like PATTON [169] and GraphTranslator [170], utilize TAGs with classification tasks or generate text captions.
- **Prompt-tuning:** Prompt-tuning leverages pre-trained LLMs for graph-learning tasks by fine-tuning the set of parameters associated with the LLM's prompts. This highly efficient approach is employed by many works, as seen in models like GraphPrompt [140] and GraphGPT [154]. Another approach sometimes employed is Tuning-Free Prompting, also known as zero-shot or few-shot prompting. Instead of fine-tuning the model's parameters, prompts are modified to receive accurate outputs from the model. It requires no additional computational resources and relies on the quality of prompts used, making it a good choice for some, but not all, models, as demonstrated in Text2Mol [157] and SimTeG [162].

Effectiveness of GNN+LLM

By combining GNNs and LLMs, models can achieve more robust performance. Models using GNNs and LLMs leverage the ability of GNNs to learn graph structures and LLMs' ability for

graph reasoning. This approach allows developers to overcome previous shortcomings in the separate models but can have challenges.

One of the primary challenges lies in aligning the representational spaces of GNNs and LLMs. These two types of models are designed to process fundamentally different types of data - GNNs excel at handling structured graph data, while LLMs are optimized for processing sequential text data. Bridging this gap requires developing sophisticated techniques to ensure that the graph structure and node features are accurately represented in a format that LLMs can effectively process.

Another significant challenge is maintaining the structural integrity of graph data when converting it into a format suitable for LLMs. Graph data inherently contains complex relationships and topological information that may not be easily captured in a linear text format. There's a risk of losing critical structural information during this conversion process, which could negatively impact the model's performance on graph-specific tasks.

The computational complexity of combining these two powerful model types is also a considerable challenge. Both GNNs and LLMs can be computationally intensive on their own, and integrating them may lead to models that are prohibitively expensive to train and deploy, especially for large-scale graphs or in resource-constrained environments.

Furthermore, there's the challenge of balancing the contributions of the GNN and LLM components. It's crucial to design architectures and training procedures that allow both components to contribute meaningfully to the final output, rather than having one dominate the other. This balance is essential for leveraging the strengths of both model types - the GNN's ability to capture graph structure and the LLM's proficiency in processing textual information and leveraging world knowledge.

There's also the issue of interpretability. While GNNs offer some level of interpretability through their message-passing mechanisms, LLMs are often considered "black boxes" due to their complexity. Combining these models may further complicate the interpretation of results, making it challenging to understand how the model arrives at its predictions or decisions.

Lastly, the challenge of data efficiency and transfer learning needs to be addressed. While

LLMs are typically pre-trained on vast amounts of textual data, obtaining large-scale, high-quality graph data for pre-training can be more challenging. Developing effective strategies for transfer learning and few-shot learning in this combined setting is crucial for the practical application of GNN+LLM models.

In conclusion, while the combination of GNNs and LLMs holds great promise for advancing graph foundation models, it also presents significant technical and conceptual challenges. Addressing these challenges will require innovative approaches in model architecture, training techniques, and data representation.

2.3.2 Levels of Generalization in Graph Foundation Models

Now that we've discussed various implementations for graph foundation models, this chapter focuses on graph-based models that can be generalized and used as GFMs. One of the key challenges in this area is the diversity of graph structures, as graphs can vary widely in their connectivity, feature structure, and scale. Standard graph neural networks fall short of being truly "foundational" because they typically perform well only on graphs with the same type and dimension of features. Meanwhile, graph heuristics like Label Propagation [171] or Personalized PageRank [172], which can operate on any graph, do not qualify as Graph FMs since they lack a learning component.

As much as LLMs have advanced, the idea of parsing graphs into sequences for LLM processing, as seen in approaches like GraphText [173] or Talk Like A Graph [174], still raises concerns about whether this method effectively retains graph symmetries and can scale beyond small datasets. We leave the exploration of LLMs in graph contexts for a separate discussion.

The most critical aspect of designing Graph FMs is the creation of transferable graph representations. In the case of LLMs, any text in any language can be reduced to tokens from a fixed-size vocabulary [175]. Similarly, Video-Language FMs rely on patches that can be consistently extracted from images (since every image or video has RGB channels). However, finding a universal featurization scheme for graphs is less straightforward, given the diverse nature of graphs. For instance:

A large graph with node features and some node labels is typical in node classification tasks. A large graph without node features or classes, but with meaningful edge types, as seen in link prediction and knowledge graph reasoning. Many small graphs, with or without node or edge features, and with graph-level labels, as in graph classification and regression tasks. These examples illustrate the complexity of developing a foundational approach that can be generalized across such varied graph structures and tasks.

As shown in 2.5, an ideal graph foundation model would be capable of handling any graph, regardless of its node, edge, or graph features, and performing any task at the node, edge, or graph level.

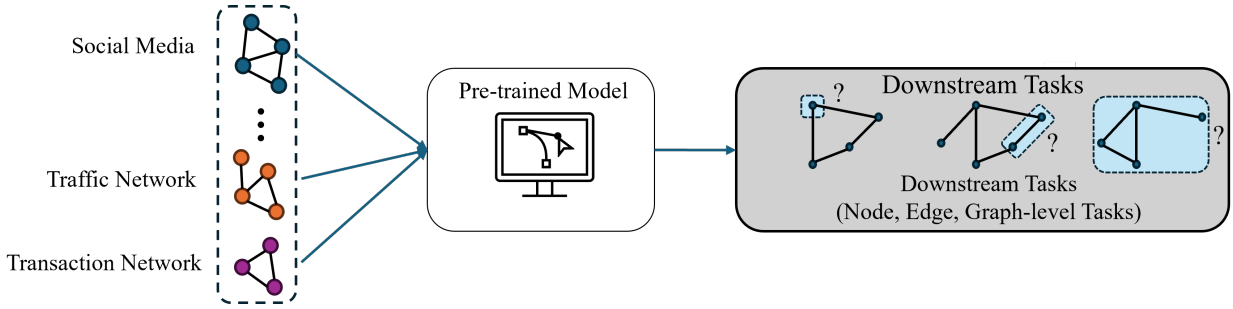


Figure 2.5: In an ideal graph foundation model, the model is pre-trained on a specific task and can then be applied to different datasets across various domains for inference, making it adaptable to a wide range of downstream tasks. We call these models as general foundation models.

Given the vast array of domains and downstream tasks, developing a model that functions as an ideal GFM is extremely challenging. Such a model would need to effectively operate across various domains and be adaptable to all new tasks. Despite this, domain-specific and single-task models remain highly valuable as they can generalize to unseen networks within their specific domains.

Three types of graph models show potential for generalizing to unseen data. The first type is domain-specific models focused on specific tasks. These models can be applied to new, unseen graphs within the same domain without additional training, as long as the target task remains consistent with the training setting. We refer to these as domain-specific, task-targeted models.

The second type consists of models trained on graphs from various domains but designed to

perform a single foundational task. These models have been trained on diverse domain datasets but are specialized in one graph downstream task. They can be adapted to unseen datasets for that particular task.

The third and most ambitious category is models that are both domain- and task-independent. Although models with strong performance across multiple tasks can be highly beneficial, it's important to recognize that all types of such models are still in their early stages of development. While a general model that can be applied to all downstream tasks is certainly valuable, generalized models within each segment are extremely important as well. These models can generalize to unseen data, paving the way for more complex, universal models in the future.

Moreover, within each category, generalizable models significantly reduce the need for extensive training. Since domain-specific models are typically optimized for particular tasks, they often deliver superior performance. These models are particularly useful in scenarios where precision and performance are critical, and the outcomes must be reliable.

Currently, all three categories are actively being explored by the research community, but they remain in their early stages. In this section, we will review the existing graph foundation models and those that can be considered as such within each segment. 2.6 shows the roadmap of the GFM evaluation.

Specific GFMs

These models are built for specific purposes within a particular domain, where the aim is to answer a well-defined question using similar types of input graphs. The model is trained to perform exceptionally well on that specific task, and it's typically used for the same task during testing. These models aren't meant to be applied to other tasks or domains. They're particularly useful when the task is critical and demands high accuracy and performance. Such models are used in fields like knowledge graphs, geometry, finance, and medicine. Once trained on a large amount of domain-specific data, they can be applied to new, unseen networks within the same domain, which helps reduce the need for additional training. The following section highlights some models in this

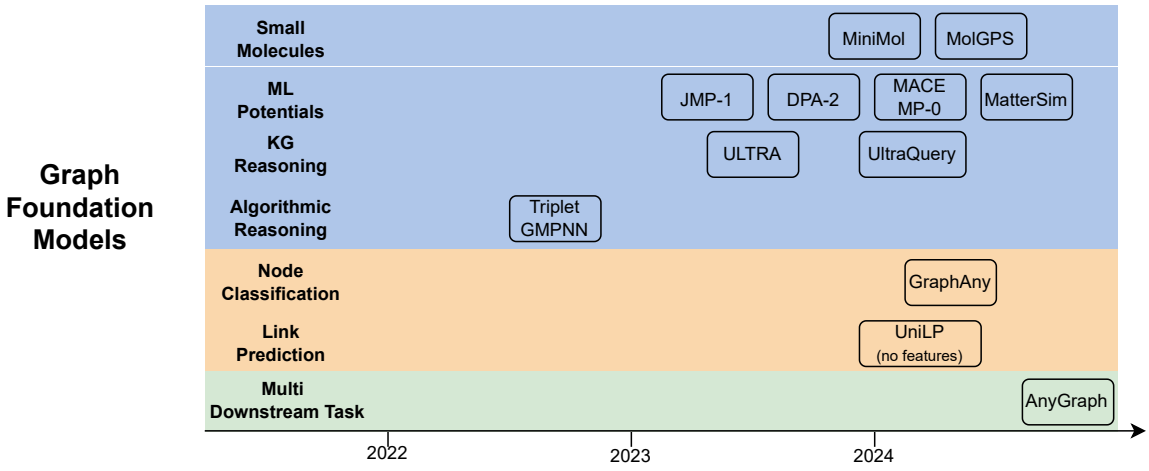


Figure 2.6: Timeline of graph foundation models categorized by specificity: Purple represents specific models, orange indicates partially specific models, and green denotes general models.

category developed within the graph foundation model literature.

Reasoning in Knowledge Graph

Knowledge graphs (KGs) organize interconnected information across diverse entities and relationships. The specific meanings of these relationships often differ between KGs, making it difficult for traditional reasoning models to adapt to new or unseen KGs with different entities and relations.

ULTRA [176] is a pioneering foundation model for KG reasoning, capable of performing zero-shot inference on any multi-relational graph, regardless of size or vocabulary. ULTRA works by first taking a query, such as asking about the genre of a specific artist and then constructing a smaller graph that focuses on the relationships relevant to that query. This smaller graph captures the interactions between different relations in the original, larger knowledge graph. The model then uses the information from this focused graph to enhance the relational features in the larger graph, which helps it to accurately answer the query. ULTRA has shown superior performance across 57 different graphs, surpassing models fine-tuned for individual KGs. The model has also evolved into UltraQuery [177], which handles complex logical queries like conjunctions, disjunctions, and negations on unseen graphs, further showcasing its adaptability. ULTRA's key innovation lies in its

ability to model relational interactions that transcend specific relation identities. This allows it to generalize and apply its learned representations effectively across various domains, from general encyclopedic knowledge to specialized biomedical applications.

Neural algorithmic reasoning

Neural Algorithmic Reasoning (NAR) [178] aims to replicate the behavior of standard algorithms—such as sorting, searching, or dynamic programming—within a neural network's latent space. The central idea is to represent these algorithms as processes that can be generalized to inputs of varying sizes and complexities. In this context, the input to NAR is typically a graph G, where nodes and edges carry features that represent the elements and relationships involved in the algorithm. For example, nodes might represent elements in a list to be sorted, or vertices in a shortest path problem, while edges could represent connections or relationships between these elements.

The output of NAR corresponds to the correct solution for the algorithmic task, such as a sorted list, the shortest path between nodes, or the convex hull of a set of points. The NAR model processes this graph-based input through a graph neural network, which serves as a "processor." Inside the GNN, the network updates the latent representations of the nodes and edges iteratively, simulating the steps of the algorithm. The GNN leverages message passing, where information is exchanged between connected nodes, to mimic the control flow of the original algorithm.

The key to NAR's transferability lies in the homogeneous feature space and the shared control flow among similar algorithms. The Triplet-GMPNN [179] is a universal processor neural net that exemplifies this approach. the model processes triples of nodes and their features, akin to mechanisms like Edge Transformers. The GNN learns to generalize across similar algorithms by recognizing patterns in how algorithms share structural similarities, despite minor differences in execution steps. The network's ability to handle multiple tasks in a shared latent space allows it to improve performance across a range of algorithmic challenges, outperforming specialist models trained on individual tasks.

Geometric and scientific deep learning

Graph foundation models are gaining prominence in Geometric Deep Learning and various scientific disciplines because they effectively address complex problems. These models excel when there is a common framework, such as the names of atoms in small molecules or the types of amino acids in proteins. This shared terminology simplifies the organization and representation of relationships in the data. Despite their effectiveness, these models face challenges due to the intricate nature of the entities they study. Atoms and molecules have detailed three-dimensional structures and physical properties like energy, which are governed by principles from chemistry, physics, and quantum mechanics. In graph terms, atoms are represented as nodes, and their interactions, such as chemical bonds, are shown as edges. This creates a comprehensive framework for understanding interactions and properties. Nonetheless, graph foundation models have proven to be valuable tools. The following section will explore the graph foundation models that have emerged in this area of research.

- **Machine learning potentials:** are predictive models used to estimate the potential energy and forces within chemical compounds based on their three-dimensional structures. These models take the 3D coordinates of atoms as input and output the total potential energy and per-atom forces, enabling applications in fields like molecular dynamics, drug discovery, and materials science. They are highly versatile, generalizing across a wide range of atomistic structures using atomic information from the Periodic Table. Recent advancements include models like JMP-1 [180] and DPA-2 [181], which excel in tasks involving both organic molecules and inorganic crystals. These models have shown outstanding performance in various benchmarks and are designed to match or exceed the accuracy of traditional specialized models. State-of-the-art models like MACE-MP-0 [182] and MatterSim [183] focus on inorganic crystals, handling complex tasks such as multicomponent alloys and combustion processes. These ML potentials leverage equivariant graph neural networks, which effectively process spatially varying features like Cartesian coordinates alongside invariant features like atom types. This architecture allows the

models to accurately predict energy and forces by capturing the intricate relationships between atoms, enhancing the understanding and representation of complex chemical systems.

- **Molecular property prediction:** In the molecular property prediction task, the objective is to predict various molecular properties based on a molecule's 2D graph structure, where nodes represent atoms and edges represent bonds. A key advantage in this domain is the transferability of atom types from the periodic table and bond types across different tasks. Unlike traditional molecular fingerprints, which were used primarily to assess molecular similarity, modern approaches use graph neural networks or Transformer encoders to learn representations that can predict physical properties. Recent advancements in graph foundation models, such as MiniMol [184] and MolGPS [185], have been pivotal in this area. These models are trained on extensive datasets of molecular graphs, learning universal molecular representations that can be fine-tuned for a variety of downstream tasks. While a single pre-trained model from these approaches may not perform zero-shot inference on every possible task, they offer a robust foundation for task-specific fine-tuning. This makes them both efficient and versatile in molecular property prediction, significantly reducing computational costs.

Partially Specific GFMs

Partially specific foundation models are those that are not fully adaptable to new tasks and domains but still offer some flexibility in their use. Partially specific foundation models offer a significant advantage over completely specific models by providing a balance between adaptability and specialization. While they may not be fully flexible across all tasks and domains, they are designed to perform efficiently in their targeted areas. This focused approach allows them to generalize well within certain constraints, either across multiple tasks in a single domain or across multiple domains for a specific task. These models fall into two main categories: domain-specific models and task-specific models.

Domain Specific GFMs

Domain-specific GFMs are designed to generalize across a variety of downstream tasks within a specific domain, such as finance, biology, or social networks. These models focus on capturing the unique structural and dynamic properties of the data within a single domain, allowing them to perform efficiently on tasks like node classification, link prediction, and graph-level predictions. For example, a domain-specific GFM trained on financial data could be applied to tasks like stock market forecasting (node-level prediction), detecting transaction fraud (link prediction), or assessing the risk profile of entire financial systems (graph-level task). By honing in on domain-specific characteristics, these models can adapt to multiple tasks within their field without needing to be retrained for each new dataset. The strength of domain-specific GFMs lies in their ability to exploit specialized knowledge from a given domain, leading to better task performance compared to more general models. However, to the best of our knowledge, no work has yet successfully created a domain-specific GFM that fully generalizes various tasks within one domain without retraining. While GNNs exist for specific tasks in particular domains, a GFM that can seamlessly perform multiple node, link, and graph-level tasks within a single domain remains an open research challenge. The development of such models would have a profound impact on areas like healthcare, finance, and transportation, where domain-specific data structures and patterns are critical to solving a wide array of complex tasks.

Task Specific GFMs

task-specific foundation models involve training graph foundation models on a diverse range of datasets across various domains but with a focus on solving specific downstream tasks. These models are designed to handle different types of input data, leveraging their extensive training to adapt and apply their knowledge in various contexts. Although they are exposed to a wide array of data during training, their primary goal remains to excel at particular tasks. The strength of these models lies in their ability to generalize and perform consistently on these specialized tasks, even when encountering new and unseen graphs, demonstrating their adaptability and efficiency in

meeting specific objectives. In this section, we will explore graph foundation model advancements that have been developed within these categories.

Node level tasks Node classification is a machine learning task applied to graph-structured data, where the goal is to assign labels or categories to nodes based on their attributes and the overall graph structure. This task is crucial in various applications, such as identifying user types in social networks or predicting protein functions in biological networks. Traditionally, node classification required specialized models tailored to specific datasets, meaning each new graph necessitated a unique model configuration, which could be time-consuming and computationally expensive. In traditional GNNs, models had to be trained specifically for each dataset, like the Cora dataset, requiring significant computational resources to adapt to each graph's unique features and label dimensions. This process was both labor-intensive and limited the scalability of GNNs, making it challenging to apply these models to new, unseen graphs. The need for dataset-specific models hindered the widespread use of GNNs across diverse graph structures.

To the best of our knowledge, GraphAny [186] is the first GFM introduced to overcome these limitations by providing a single pre-trained model capable of performing node classification across any graph without additional training or adaptation, regardless of differing feature dimensions or class numbers. GraphAny generalizes across various graphs, outperforming standard models like GCN and GAT by eliminating the need for fixed prediction heads tailored to specific class counts. This flexibility marks a significant advancement in handling diverse datasets efficiently. GraphAny employs semi-supervised node classification without being constrained by feature dimensions or class counts. Instead of relying on a universal latent space, GraphAny uses spectral filter predictions, applying high-pass and low-pass filters similar to those in Simplified Graph Convolutions. The model optimizes predictor weights through a least squares method, enabling robust predictions for unknown node labels. Additionally, GraphAny refines its predictions using an inductive attention matrix, which adjusts prediction weighting based on the specific graph's structure. This parametric attention mechanism, parameterized through a Multi-Layer Perceptron, reduces dependency on predefined class numbers, allowing GraphAny to perform rapid inference

across various graph-related tasks efficiently, demonstrating its adaptability and efficiency.

Link level tasks Link prediction is a fundamental task in graph analysis that involves predicting whether a connection exists between two nodes in a graph. This task is essential in various applications, such as recommending friendships in social networks, identifying interactions in biological networks, or uncovering missing links in knowledge graphs. Traditionally, models for link prediction had to be trained separately for each graph, which was both time-consuming and computationally intensive due to the unique structures and features of each graph.

To the best of our knowledge, the Universal Link Predictor (UNILP) [187] is the first GFM designed for universal link prediction. UNILP overcomes the limitations of traditional models by providing a single pre-trained model capable of performing link prediction across any graph without additional training or adaptation, regardless of whether node features are present. This universality marks a significant advancement, as it allows for efficient and scalable application of link prediction to new and unseen graphs.

UNILP works by leveraging structural patterns within the graph using advanced labeling techniques to distinguish between nodes, even in the absence of node features. It addresses the challenge of automorphic nodes—nodes that occupy identical structural positions and are thus indistinguishable by standard graph neural networks. UNILP employs labeling tricks like Double Radius Node Labeling to break these symmetries. The model processes a query link alongside sampled positive and negative in-context links from the target graph through a shared subgraph GNN encoder. An attention mechanism then computes the similarity between the query link and the in-context links, resulting in a contextualized representation of the query link based on the graph's structure.

By capturing transferable structural patterns and utilizing an attention-based mechanism, UNILP generalizes learned relational patterns to effectively predict links in various graph types. Its ability to function without reliance on specific node features and to generalize across different graphs makes UNILP an ideal Graph Foundation Model for universal link prediction. This approach significantly reduces the need for retraining models for each new graph, enhancing scalability and

efficiency compared to traditional link prediction models.

Graph Level tasks Graph-level tasks focus on analyzing and predicting based on an entire graph rather than individual nodes or edges. These tasks include graph classification [78], which aims to categorize a graph into predefined classes based on its overall structure and features, and graph property prediction [17], which involves forecasting specific properties of the entire network. In both cases, understanding the global characteristics of the graph is essential, as it surpasses the importance of examining individual components.

Graph-level foundation models extend the capabilities of GNNs to learn graph representations that are versatile and applicable across various domains. These models aim to generalize across different types of graph data, enabling seamless application to graph-level tasks like classification or property prediction, even in unseen networks. Their ability to adapt without retraining facilitates efficient processing and analysis in diverse fields, potentially improving predictions and insights across complex datasets. However, to the best of our knowledge, the development of graph-level foundation models for broad, cross-domain application remains an emerging field, with limited work and exploration in this specific area so far.

General GFMs

Graph Foundation Models designed for general domains and multiple tasks are developed to work with any graph, regardless of its structure or features. These models are not limited by specific input graphs and can handle a variety of downstream tasks, such as node classification, link prediction, and graph-level tasks, across different domains. The adaptability of these models makes them highly valuable, as they allow users to perform inference on a wide range of tasks with minimal computational resources. This flexibility and efficiency are critical in real-world scenarios where multiple tasks may need to be executed without training separate models for each graph or domain.

One of the key benefits of these generalist models is their ability to transfer learning across tasks and domains, significantly reducing the need for retraining. AnyGraph [188], the first work in this area, has shown the potential of these models. AnyGraph has been designed to handle node,

edge, and graph-level tasks, making it one of the first models to be evaluated across all three task types. This capability allows it to seamlessly adapt to new tasks and datasets while maintaining high performance. AnyGraph's methodology centers on a Mixture-of-Experts (MoE) architecture. This architecture uses a collection of graph expert models, each responsible for handling specific structural or feature-based challenges in graph data. When presented with a new graph, AnyGraph's lightweight routing mechanism quickly identifies the most relevant expert model, which is then applied to the task. This design allows for efficient adaptation to new datasets without requiring extensive retraining, as demonstrated by its impressive performance across 38 diverse graph datasets.

While models like AnyGraph show great potential, they still face some challenges. These models need to handle the differences in structure and features across various domains and ensure they can scale as data complexity grows. Despite these hurdles, GFMs' ability to adapt to different tasks could revolutionize graph learning, opening up exciting possibilities in areas like biology, finance, and social networks.

2.3.3 Homogenization as a Challenge in Foundation Models

One of the fundamental challenges in building foundation models is achieving *homogenization*. Homogenization refers to a model's ability to generalize and be applied uniformly across different tasks and data formats. In the context of artificial intelligence, and particularly in large-scale architectures such as language, vision, and graph foundation models, homogenization describes the goal of learning a common representation space that can flexibly adapt to heterogeneous downstream applications.

In temporal and graph foundation models, achieving homogenization is particularly demanding due to the wide variability of graph structures, temporal resolutions, and learning objectives. Unlike static models that operate on fixed feature spaces, temporal graph models must reconcile differences in node and edge semantics, time dynamics, and topological sparsity while preserving a consistent model design.

Formally, the challenge of homogenization in foundation models can be described along three dimensions:

- **Task Generalization:** Developing a single model that can support diverse tasks—such as node classification, link prediction, and graph-level regression—without structural modification or retraining for each task.
- **Cross-Domain Adaptation:** Ensuring that the same pre-trained model can operate across distinct domains (for example, social, financial, or molecular graphs) while maintaining performance stability and representation fidelity.
- **Architectural Uniformity:** Preserving a consistent architectural backbone and training pipeline that allows task-specific fine-tuning through minor adaptations rather than full reconfiguration.

In graph foundation models, this challenge manifests as the tension between specialization and universality. Traditional graph neural networks are typically optimized for narrow objectives and domains, whereas GFMs aim to encode universal graph principles that transfer across datasets and learning tasks [131]. Homogenization thus represents both a design aspiration and an open research problem—balancing the need for a unified, domain-agnostic model structure with the flexibility required to handle diverse temporal behaviors and topological variations.

2.4 Temporal Graph Foundation Models

Temporal graph foundation models are a novel category of graph deep learning techniques that aim at modeling the development of the various graphs that exist over time. Static graphs are structures that display relationships among entities at a particular instant of time. Graphs that change over time and become more complex with the addition of new nodes and edges and their interactions are termed temporal graphs. Temporal GFMs would still be composed in a way that they will be able to transfer learning between multiple temporal graphs and tasks rather than having one graph

to one task as in the case of static GFMs. However, as of today, there are no comprehensive works or models that can be considered temporal GFM.

2.4.1 The Challenge of Temporal Graphs

Despite the growth of static graph foundation models, where GFM is exceptionally powerful, it is still a challenge to construct Temporal GFMs due to the complications involving the aspect of time. Each of these challenges becomes particularly difficult when dealing with different datasets and domains:

- **Dynamic Structures:** This is particularly important for a temporal graph where the specification of the graph changes with time and thus the model has to learn not only the connections that the nodes have but also how these connections vary with time [114]. The difference between a static GFM and a temporal model is that temporal models do not learn within a time frame but rather capture changes within a time frame through patterns or sequences. This is particularly problematic for datasets where there is constant presence of structural changes, as the frequency and type of changes vary across domains, for instance, social domains tend to change at a fast pace while the biological domain changes slowly.

- **Memory and Computation:** The requirements of the graph go even further as it also includes management of past activities, which therefore calls for quite a lot of memory space as well as computation power [114]. New data points are being generated on a continuous basis and so it becomes necessary to restrict inference to a smaller but discrete number of data points to sustain performance and bandwidth. Putting this into practice across different datasets becomes even more difficult as some domains will accommodate live changes in the information such as in financial markets while others will accommodate changes after a certain period.

- **Temporal Dependencies:** Such dependency exists over time among the temporal vertices in temporal graphs. That is, some events leading to the interaction of two nodes may happen at different time frames [189]. A GFM with a temporal constraint must manage quick as well as

deferred dependencies. This becomes a significant challenge when applied to different domains, as the nature of these dependencies may differ drastically—for example, recent interactions in a communication network may be more important for this thinking whereas old interactions in biological systems may be more influential in this regard.

- **Transferability Across Domains:** In static GFMs, transferability arises from the ability to generalize over the graph structures. For temporal graphs, however, such a generalization is not clear as their relationships are more complex and temporal which is very much different across domains. A Temporal GFM that has been trained using social data might not perform well on a biological system because biological interactions are modeled completely differently.

- **Modeling Temporal Features:** To learn efficiently, temporal graph models must rely on both time and structures of the graph, including timestamps and the duration of interactions. Such temporal information must be well-fitted into the graph model if the intention is to improve the representations produced. The problem of achieving this becomes quite complicated especially when dealing with different datasets where temporal features may assume different levels of significance. For instance, some interactions might be very essential in social networks whereas less important in some scientific datasets.

Each of these challenges suggests why it is more complicated to develop Temporal GFMs in comparison to static GFMs. Especially when such Temporal GFMs are extended to real datasets and domains which have very unpredictable temporal dynamics and relationships.

2.4.2 Applications of Temporal GFMs

TGFMs can be useful for a variety of tasks and domains much like static GFMs have demonstrated. These models can be designed to be specific, partially specific, or general-purpose for handling time-evolving data. An advantage of TGFMs is they can both i) handle unseen data for a new inference without any retraining and ii) operate in environments where training new models is impractical or infeasible. For these reasons, TGFMs will be indispensable for real-time modeling

where there is demand for forecasts of new data over arbitrarily long timescales which are not near what the model has been trained on.

- **Specific TGFMs:** Specific TGFMs are trained for a particular domain and task, offering high-performance predictions on completely unseen networks in a zero-shot setting without any additional training. These models serve as an important first step in advancing the understanding of TGFMs by providing insights into how to train models that generalize across both the temporal and spatial aspects of a network. They are particularly valuable in real-world scenarios where multiple datasets exist within the same domain, and training a separate model for each network is impractical or impossible. Specific TGFMs offer a streamlined solution, making them highly applicable in such contexts.

- **Partially specific TGFMs:** Similar to static GFMs, partially specific TGFMs are models that can be adapted to new tasks or domains without full retraining. These models are divided into two categories: task-specific and domain-specific. Each type has significant impact and practical applications in real-world predictions. Task-specific TGFMs are highly optimized for a particular function, such as temporal node classification, link prediction, or anomaly detection. These models can generalize across unseen graphs in the same task category without the need for retraining. For instance, a TGFM designed for real-time stock price forecasting could predict future trends in various financial markets without needing to be retrained on each new dataset. Similarly, task-specific TGFMs can be deployed in fraud detection in banking networks, where the rapid adaptation to new transaction patterns is critical to catching fraudulent activities as they occur. The ability of task-specific TGFMs to handle such problems without retraining saves valuable computational resources and speeds up decision-making. Domain-specific TGFMs go beyond a single task and are designed to generalize across multiple tasks within a particular domain. For instance, in the financial sector, a TGFM could be trained to handle tasks such as stock market prediction, transaction analysis, and risk management across various datasets from different markets and institutions. The TGFM's ability to learn generalized patterns within the financial domain allows it to be applied across different companies, markets, or time periods

without retraining. This capability is particularly valuable for sectors like logistics or transportation, where models can track the evolution of traffic patterns, supply chains, and deliveries over time, adapting to new networks and timelines on the fly.

- **General TGFMs:** The most ambitious and far-reaching type of TGFM is a general model that can be applied to any domain or task. Such models would operate similarly to large language models in the realm of natural language processing but for graphs. A general TGFM could analyze any form of temporal graph data—social networks, biological systems, financial networks, or knowledge graphs—without needing domain-specific adaptation or retraining. This universality makes them extremely powerful, as they can scale to handle a wide range of applications, from forecasting economic trends in different countries to predicting disease outbreaks or analyzing the evolution of global transportation systems. The ability of a general-purpose TGFM to adapt to any domain and task without further learning ensures high scalability and efficiency. For example, in a real-time financial forecasting scenario, the model could predict stock market movements or detect irregularities across multiple financial systems simultaneously, without requiring separate models for each market. The model's zero-shot learning capabilities make it invaluable in domains where data scarcity or time constraints prevent the training of specialized models, such as during the early stages of a crisis or disaster management scenario, where real-time decisions must be made based on incomplete data.

The importance of TGFMs becomes apparent in temporal and real-time forecasting, especially in situations where data gathering and model retraining are not feasible. In many real-world applications, graphs evolve, and it is often impractical to continuously collect data or re-train models with each new sequence of events. TGFMs are adept at analyzing sequences of graphs and understanding the dynamics inherent in the changing data, allowing for accurate predictions based on historical trends. Their ability to function effectively without constant retraining makes TGFMs indispensable for applications such as recommendation systems, fraud detection, and social network analysis, where real-time insights can significantly impact decision-making. By capturing the temporal dependencies in graph data, TGFMs provide valuable tools for understanding complex

interactions and making informed predictions in rapidly changing environments.

2.5 Graph Learning Dataset and Benchmarks

Benchmark datasets have become fundamental for advancing graph machine learning, providing a common ground to evaluate models and facilitate the development of graph foundation models. Early graph ML studies often relied on a handful of small, static benchmark graphs (e.g., citation networks like Cora/Citeseer and molecular graphs from the TU collection [190]). Repositories such as the Stanford SNAP dataset collection and the Network Repository cataloged many graphs for research use, but without standardized tasks or unified evaluation protocols [191, 192]. The lack of consistency in tasks and splits made it difficult to compare algorithms fairly. This motivated community efforts to create dedicated benchmark suites that are large-scale, diverse, and standardized.

2.5.1 Static Graph Benchmarks

Static graph benchmarks focus on time-agnostic graph tasks (node classification, link prediction, graph classification) on fixed networks or sets of graphs. A seminal effort in this direction is the Open Graph Benchmark (OGB) [193], introduced at NeurIPS 2021. OGB provides a diverse collection of realistic graph datasets spanning domains such as social networks, citation networks, molecular graphs, knowledge graphs, and more. Importantly, OGB defines consistent evaluation protocols with meaningful train/validation/test splits (e.g., avoiding overly easy random splits) and unified metrics, addressing issues of reproducibility and out-of-distribution evaluation. The OGB suite includes challenging datasets like a citation network with hundreds of thousands of nodes (OGBN-Arxiv), a protein interface graph (OGBL-PPA) for link prediction, and molecule datasets for graph property prediction [193]. By benchmarking various GNN models on these datasets, Hu et al. showed that OGB tasks remain far from solved and identified key challenges such as scalability to large graphs and generalization under realistic splits. Following OGB's release, it

has become a standard evaluation framework in graph ML research, with a public leaderboard tracking state-of-the-art results on each task. Building on OGB, the community pushed toward even larger-scale benchmarks to catalyze foundation model-level advances. Hu et al. organized the OGB Large-Scale Challenge (OGB-LSC) in 2021, as part of KDD Cup and later reported at NeurIPS 2021 [194]. OGB-LSC introduced three exceedingly large graph datasets, each targeting a core task at unprecedented scale: (1) a node classification task on a heterogeneous academic graph with over 240 million nodes and 1.2 billion edges (MAG240M), (2) a link prediction task on a massive knowledge graph with 90 million entities (WikiKG90M), and (3) a graph regression task on a molecular dataset with 4 million molecular graphs (PCQM4M) [194]. These datasets are orders of magnitude larger than prior benchmarks. The challenge drew 500+ teams, yielding innovative, scalable GNN approaches and significant performance improvements. Notably, the best-performing models in OGB-LSC employed techniques like deep transformer-based GNNs and aggressive neighbor sampling to handle scale (e.g., the Graphormer model, a Transformer tailored to graphs, won the molecular track) [195]. The OGB-LSC findings highlighted that expressive models can significantly outperform simpler, scalable baselines on large graphs, but also that many standard GNNs fail to even run at this scale without specialized training algorithms [194]. The annual OGB-LSC benchmark (continued in 2022) serves as a graph analog to ImageNet challenges, steering the community towards scalable graph learning techniques and pretraining strategies suited for extremely large data. Another notable effort for static graphs is the Benchmarking Graph Neural Networks study by Dwivedi et al. [196]. Rather than introducing new data, the work systematically evaluated popular GNN architectures on a curated set of existing graph datasets (including both classical node/graph classification benchmarks and synthetic graph tasks). It revealed inconsistencies in prior evaluations and underscored the need for standard benchmarks like OGB. Overall, these static benchmark initiatives (OGB and others) have greatly improved the rigor and comparability of graph model evaluation. They also supply the data foundation for graph representation learning, for instance, OGB datasets are commonly used to pre-train GNNs or evaluate graph foundation models via fine-tuning.

2.5.2 Temporal Graph Benchmarks

While static benchmarks assume a fixed graph structure, many real-world graphs are dynamic, evolving over time with new nodes or edges (e.g., social interactions, transaction networks, communication networks). Until recently, research on temporal graph neural networks relied on individually curated datasets and inconsistent protocols. For example, Kumar et al. (KDD 2019) introduced dynamic interaction graphs for Reddit and Wikipedia to evaluate embedding trajectory prediction [2], and various works used their own splits of social network data (e.g., user-item interaction logs, citation dynamics) to benchmark temporal GNN models. This fragmented landscape made it hard to gauge progress in learning on temporal graphs. Recognizing this gap, Huang et al. proposed the Temporal Graph Benchmark (TGB), first released in 2023 [14]. TGB (accepted at NeurIPS 2023) is a unified collection of large-scale temporal graph datasets with standardized tasks and evaluation pipelines. It covers diverse domains, including social networks, communication, trade, finance, and transportation, reflecting the broad applicability of temporal graph learning. TGB defines two primary task categories: dynamic link prediction (predicting the future existence or properties of edges) and dynamic node property prediction (predicting future attributes or labels of nodes). Each dataset consists of a sequence of graph snapshots or time-stamped edge events spanning multiple years. For example, TGB includes a Wikipedia co-editing network (users editing pages over time), an E-commerce review network (Amazon user–product interactions over 20+ years), a Reddit reply network (users replying to each other over time), and an air traffic network (temporal flights between airports), among others. Crucially, TGB provides consistent train/validation/test splits along the timeline and an automatic evaluation pipeline, enabling reproducible benchmarking of temporal graph models. In their extensive experiments, Huang et al. found that the performance of popular temporal GNN architectures varies wildly across different TGB datasets, and intriguingly, on certain dynamic node prediction tasks, simple time-series models can outperform complex temporal GNNs [14]. These insights point to open challenges and the need for better inductive biases in temporal models. TGB is an actively main-

tained project with a public leaderboard, poised to drive research in temporal graph representation learning. Recently, TGB 2.0 was introduced at NeurIPS 2024 to extend this benchmark to multi-relational dynamic graphs, i.e., temporal knowledge graphs and heterogeneous graphs with various edge types [197]. TGB 2.0 contributed eight new datasets spanning domains like social media, biomedicine, and communications, with up to tens of millions of time-stamped edges [197]. The inclusion of temporal knowledge graphs (e.g., evolving knowledge bases of events) bridges graph ML with temporal reasoning tasks from the knowledge graph community. Experiments in TGB 2.0 underscored that leveraging edge type (relation) information is crucial for high performance on multi-relational tasks, and again noted that simple heuristic methods can sometimes rival more sophisticated models [197]. However, most existing methods struggled to scale to the largest TGB 2.0 graphs, reinforcing the necessity for new scalable temporal GNN techniques. Together, TGB and its extension provide a comprehensive platform to evaluate how well algorithms handle the evolving, temporal aspect of graphs, complementing the static benchmarks like OGB.

2.5.3 Blockchain Graph Benchmarks

One emerging application domain for graph learning is blockchain networks, which pose unique challenges and opportunities for benchmarks. Blockchains (like Bitcoin and Ethereum) can be represented as graphs (e.g., transaction networks where addresses are nodes and transactions are edges), often large-scale, dynamic, and multi-layered. For instance, the Bitcoin network continuously grows with each block of transactions, and Ethereum's smart contract calls form complex interaction graphs. Traditionally, machine learning on blockchain graphs was limited by data accessibility and labeling: researchers relied on isolated datasets curated for specific tasks, with no unified benchmark. A notable example is the Elliptic illicit transaction dataset [198], introduced in 2019, which consists of a Bitcoin transaction graph with $\sim$200K nodes labeled as licit or illicit. This dataset has been widely used to evaluate graph-based fraud detection and anti-money laundering models, and it established a baseline task of classifying illicit transactions on a large transaction graph. Other works have compiled datasets for tasks like Ethereum phishing address detection or

DeFi fraud, but each dataset was used in a siloed manner in its respective paper [199]. To advance graph ML for blockchain data, Shamsi et al. introduced Chartalist [16], the first comprehensive repository of labeled blockchain graph datasets, which was published in the NeurIPS 2022 Datasets and Benchmarks track. Chartalist organizes blockchain data (from both UTXO-based blockchains like Bitcoin and account-based blockchains like Ethereum) into ML-ready graph datasets, complete with labels and task definitions. Importantly, it addresses the considerable preprocessing burden: raw blockchain ledgers are massive and require domain expertise to convert into meaningful graph features and labels. Chartalist provides cleaned and annotated graphs, including dynamic multi-layer networks extracted from blockchains. For example, it curates the evolving transaction graph of Bitcoin with ground-truth labels for certain known illicit events (like ransomware addresses), and similarly for Ethereum, it tracks address interaction networks with identified scams or anomalies [16]. By incorporating major blockchain events and annotating addresses (e.g., hacks, frauds), Chartalist enables supervised learning tasks such as address classification, transaction link prediction, temporal anomaly detection, and forecasting on blockchain graphs. This was a significant step, as previously no public benchmark existed for graph ML on blockchain data [16]. Chartalist's datasets are large-scale (the Bitcoin network graph in 2025 exceeds 400 million edges) and dynamic by nature, reflecting months or years of blockchain activity rather than static snapshots. Recent benchmark efforts have further expanded blockchain graph datasets and tasks. One example is the "Multi-Chain Graphs of Graphs" dataset proposed by Luo et al. [200]. This work goes beyond single-chain analysis by constructing a hierarchical Graph-of-Graphs: local transaction graphs for multiple cryptocurrencies are connected via a higher-level graph that captures interactions between those cryptocurrencies (for instance, if one token is used to purchase another). Their dataset includes detailed labels at the token level and links across blockchains, supporting novel tasks like cross-chain link prediction and anomaly detection. This approach recognizes that modern blockchain ecosystems are interconnected (e.g., users swapping assets across chains), and analyzing them requires considering a network-of-networks structure. Another notable dataset is EX-Graph by Wang et al. [201], introduced at ICLR 2024, which bridges blockchain data with

social networks. EX-Graph links the Ethereum transaction graph with the Twitter (X) social graph by identifying accounts that are active in both networks. It contains 2 million Ethereum addresses (nodes) and 30 million transaction edges, alongside 1 million Twitter user nodes and their following relationships, with over 30,000 address–social account linkages. By combining on-chain and off-chain (social) information, this benchmark allows researchers to study how incorporating external social features can improve blockchain analytics, for example, using Twitter interactions to predict cryptocurrency address behavior or to detect coordinated illicit activities. The introduction of EX-Graph underscores a trend of creating hybrid benchmarks that connect blockchain graphs with other data modalities to enrich learning signals. It is worth noting that blockchain graphs also appear in the aforementioned broad benchmarks: for instance, the Temporal Graph Benchmark includes a cryptocurrency transaction dataset derived from Ethereum token transfers (a stablecoin transaction network during the 2022 Terra collapse) as one of its dynamic link prediction tasks [14, 16]. Similarly, TGB's dynamic node prediction tasks include a user–token interaction graph where the goal is to predict users' future activity with various cryptocurrencies [14]. The inclusion of these in TGB indicates a convergence where domain-specific efforts (like Chartalist) feed into general benchmark frameworks (like TGB). Going forward, we anticipate more blockchain-specific benchmarks to emerge, potentially covering areas like smart contract vulnerability graphs or transaction network simulations, given the growing interest in applying GNNs to cryptocurrency ecosystems. For now, Chartalist and its derivatives represent the state-of-the-art in providing public, labeled blockchain graph benchmarks for machine learning research.

2.5.4 Benchmarks and Graph Foundation Models

The development of these benchmarks has been closely intertwined with progress on graph foundation models and training algorithms. By graph foundation models, we refer to large, general-purpose graph neural networks (or related architectures) that are trained on broad graph data (often via self-supervised learning) and can be adapted to a wide range of downstream tasks, analogous to NLP's pre-trained language models. High-quality benchmark datasets are a prerequisite

for training and evaluating such models. For example, the massive node classification graph in OGB-LSC (MAG240M) and the huge molecular graph set in OGB have spurred research into pre-training GNNs on unlabeled graph data at scale [194]. Likewise, the diverse tasks in OGB (node, link, graph prediction across domains) provide natural downstream evaluations for a foundation model's versatility. We have started to see the emergence of self-supervised GNN training frameworks leveraging these benchmarks. Notably, Hu et al. proposed GPT-GNN [202], a generative pre-training method for GNNs using an attribute-masked graph generation task, which they demonstrated on the billion-edge Open Academic Graph (a subset of MAG) and an Amazon reviews graph. Their pre-trained model achieved significant gains on downstream node classification, showing the promise of foundation models on large graphs. Similarly, contrastive learning approaches like GraphCL [203] and graph autoencoders like GraphMAE [204] have been applied to OGB datasets to learn transferable representations. These algorithms create task-agnostic embeddings by maximizing agreement between differently perturbed versions of the same graph or by reconstructing masked features, enabling the model to capture generic graph structure and semantics.

Finally, benchmarks like TGB are driving advances in temporal graph learning algorithms that will feed into foundation models capable of handling dynamic data. The surprising observation that simple models sometimes beat complex temporal GNNs on TGB [14] suggests current architectures are not fully capturing temporal information; this has led researchers to rethink model designs (e.g., incorporating memory modules or temporal attention mechanisms) and training procedures for dynamic graphs. A foundation model that can jointly understand structure and temporal evolution might be trained by self-supervision on large temporal graphs (many of which are now available through TGB and related efforts). The multi-relational focus of TGB 2.0 [197] also pushes the development of models that can handle richly attributed graphs (multiple edge types, dynamic attributes), which is relevant for heterogeneous graph foundation models.

The ecosystem of graph and blockchain benchmarks, from static collections like OGB, to dynamic suites like TGB, and domain-specific data like Chartalist, provides the critical testbed and

training ground for graph foundation models. These benchmarks cover a broad spectrum of graph scenarios that a foundation model should excel in: large-scale static networks, evolving temporal graphs, and complex multi-relational or cross-domain graphs (as in blockchains). By benchmarking new algorithms on these datasets, researchers can identify generalization gaps and scalability issues, guiding the design of more powerful graph neural network architectures. The continued expansion of benchmark datasets (especially at top venues like NeurIPS) ensures that, as graph ML enters the foundation model era, it does so on a firm, well-evaluated base.

Our MiNT benchmark introduces a novel scale and structure for temporal graph learning by assembling 84 real-world ERC20 token transaction networks and 8 social interaction graphs, enabling both within- and cross-domain transfer studies. Unlike prior benchmarks such as TGB [14] and OGB [193, 206], which offer diverse but isolated dynamic or static graph tasks, MiNT focuses specifically on the challenge of learning transferable temporal representations across independent dynamic graphs.

Table 2.2: Comparison of temporal graph benchmarks.

Dataset	# temporal graphs included	# newly introduced graphs
EdgeBank [205]	13	6
ROLAND [108]	8	0
TGB [14]	9	8
GraphPulse [17]	9	7
MiNT	92	84

Each token network in MiNT is temporally disjoint, semantically distinct, and characterized by varying lifespan, novelty, and transactional behavior, making it unsuitable for naive aggregation or multi-label classification. This independence supports rigorous investigation of zero-shot generalization and pre-training on long-range temporal structures. Moreover, MiNT introduces network-level property prediction tasks, shifting the focus from local node- or edge-level tasks to macro-scale graph dynamics forecasting. It is the first benchmark to reveal neural scaling trends in temporal graph learning, demonstrating how increasing the number of training networks improves performance on unseen graphs. These properties position MiNT as the first foundation-model-oriented benchmark for temporal graph learning, complementing prior efforts by enabling systematic pre-training, ablation, and transfer evaluations across a controlled, large, and heterogeneous collection of temporal graphs.

Chapter 3
Topological Forest

Murat Ali Bayir, **Kiarash Shamsi**, Huseyincan Kaynak, and Cuneyt Gurcan Akcora.

Published in IEEE Access, 2022

In machine learning, ensemble methods have become a cornerstone for building robust and accurate predictive models by combining the outputs of multiple base learners. Among these methods, the *Random Forest* [207] stands out as one of the most influential ensemble techniques, widely used for both classification and regression tasks. A Random Forest constructs a set of decision trees, each trained on a random subset of data and features, and aggregates their predictions using simple policies such as majority voting, mean, or median, depending on the task type. This aggregation enhances model stability and reduces overfitting, making Random Forests a reliable choice for a broad range of machine learning applications.

Due to their simplicity, interpretability, and efficiency, Random Forests have been successfully employed across various domains, including patient health prediction, image classification, emotion recognition, malware detection, user response prediction, travel analytics, and cybersecurity [208, 209, 210, 211, 212, 213]. Unlike more complex models such as gradient-boosted trees [214] or deep neural networks [215], Random Forests are straightforward to parallelize: individual trees can be trained and evaluated independently, allowing training and inference to scale effectively across multiple machines [216, 217]. This property makes them well-suited for large-scale applications that require efficient processing of big data [218, 219, 220, 221].

Another important characteristic of Random Forests is their interpretability [222, 223], which has motivated extensive research on explainable ensemble models [224, 225, 226]. Each decision tree provides a transparent, rule-based structure that can be converted into human-readable formats [227], helping to clarify the relationship between features and predictions. This level of interpretability has contributed to the widespread adoption of Random Forests in domains where transparency and accountability are essential, such as healthcare and finance.

Despite these advantages, Random Forests are not without limitations. The ensemble may include a large number of trees, some of which contribute little to predictive performance while increasing inference time and memory usage. Previous studies have investigated the optimal number of decision trees and their collective effect on ensemble performance [228, 229, 230, 231], aiming to balance efficiency and accuracy. However, most of these approaches focus on adjusting

ensemble size rather than analyzing the individual quality or similarity of decision trees to improve model diversity and reduce redundancy. Topological forest propose a training process that takes into account the similarity and quality of decision trees in a random forest. By carefully selecting decision trees in this way, we achieve similar performance on prediction tasks with a smaller ensemble size. This would reduce the size of the model and improve computational efficiency during inference.

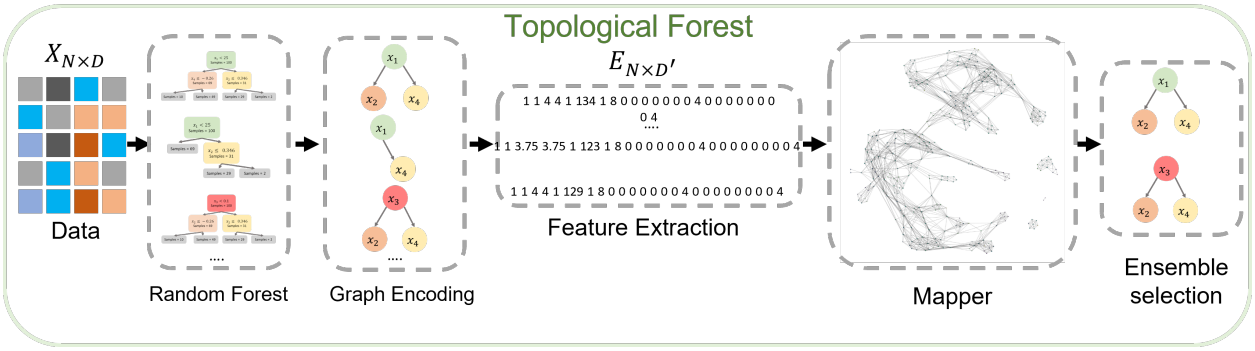


Figure 3.1: The building blocks of the Topological Forest.

We propose the Topological Forest model, which uses a smaller ensemble of decision trees to perform regression and classification tasks. The Topological Forest model has a multi-step training process that begins by training a vanilla random forest. Next, we transform each tree in the random forest into a feature graph and extract key features from the graphs as high-dimensional vectors. We then use TDA Mapper [32] to transform these high-dimensional feature vectors into a topological cluster network. This allows us to identify representative decision trees that can be used to construct a smaller, more efficient random forest. By using this approach, we aim to achieve similar performance on prediction tasks with a smaller ensemble size, reducing the model size and improving computational efficiency.

TDA Mapper is a technique that allows for the soft clustering of high-dimensional data points while constructing a network view where similar clusters are connected by short paths. TDA Mapper transforms the high-dimensional input features into a low-dimensional space (in this case, 2D) using a lens or filter function. Common choices for the filter function include projection onto one or more axes using techniques like TSNE [232] or density-based methods. Once the filter

function is applied, the data in the original feature space is transformed into 2D space. TDA Mapper then constructs a set cover of the projected space in the form of a set of overlapping intervals with constant length.

Next, TDA Mapper constructs a mapper graph with clustered trees as vertices that represent each set in the cover of the projected space. An edge exists between two vertices if the two sets (due to the overlapping intervals) share some trees in common. Finally, we use different strategies to select decision trees from each cluster to discover a more representative tree ensemble with fewer trees. Our experimental results show that trees in a cluster make similar predictions on out-of-bag samples, which makes TDA Mapper well-suited to our tree selection problem. We use topological data analysis to analyze our decision trees and make the random forest more efficient in computational costs of inference with a small compromise on prediction accuracy. Our proposed approach is unique because topological random forests can reduce the ensemble size and improve the speed of predictions compared to vanilla Random Forest at the cost of reducing at most 2% in the AUC metric. Topological Forest also keeps the diversity of decision trees by selecting individual decision trees that belong to different similarity classes.

In particular, we list the contributions of this work as follows:

- We propose a new machine learning model called Topological Forest, which uses a significantly smaller number of decision trees than traditional random forest models while achieving similar prediction quality (within 2%). This reduction in the number of decision trees allows for a smaller model size and improved computational efficiency.

- We introduce a novel graph representation of decision trees that can be used in various applications to take advantage of tree features and decision tree similarities. This graph representation allows for more effective analysis and interpretation of decision trees and can be used in a variety of contexts.

- We use TDA Mapper to map similar decision trees into the same clusters, allowing us to construct a more effective random forest by applying various ensemble selection strategies.

- From our experiments on several binary classification tasks, we find that the Topological Forest model significantly improves computational efficiency during inference, with minimal compromise on prediction quality.

3.1 Topological Random Forest

In this section, we'll explain the details of the Topological Forest: We present a high-level overview of the training process for the Topological Forest. After that, we briefly describe a vanilla Random Forest and introduce the notation we use throughout the chapter. Next, we explain the graph encoding of decision trees and discuss how a Random Forest ensemble is constructed from a topological clustering of encoded decision tree representations.

3.1.1 Overview of the Training Process

Building a Topological Forest from the training data is a multi-step process, as shown in Figure 1. In the first step, we train a vanilla random forest by using the original dataset with all its features. After that, each decision tree in vanilla random forest is transformed into the graph representation by using relationships between features. In the next step, we extract features from the graph representation of decision trees as N-dimensional vectors. These vectors have topological features like the number of edges or the average degree of vertexes.

Once feature extraction is completed, we use TDA Mapper to transform and cluster N-dimensional feature vectors into a topological network that has a cluster of nodes (each node represents a decision tree). TDA mapper step first invokes a feature embedding task by leveraging TSNE as a filter function to transform N-dimensional feature vectors in 2D. Once compressed features in 2D space are computed, TDA mapper generates a mapper graph with clustered trees as vertices by leveraging a set cover of the projected space in the form of overlapping intervals with fixed lengths.

As Figure 3.1 shows, we employ two types of graphs to build Topological Forest: a graph representation of a decision tree and a TDA Mapper produced graph (aka mapper graph) to cluster

decision trees. The first graph is relatively smaller than the mapper graph and it contains parent and child relationships of features. On the other hand, the mapper graph is a topological network of soft decision tree clusters, and an edge between two clusters exists if they share common decision trees. In the final step of the entire process, we employ an ensemble selection task to build a topological forest from the mapper graph. In the ensemble selection task, we used different selection algorithms to pick decision trees to construct a topological forest from the cluster of decision trees in the mapper graph. In the following sections, we give details of each step.

3.1.2 Training Vanilla Random Forest

We consider a binary classification scheme for ease of exposition, but our approach can be trivially generalized to multi-label classification. Let $\{x_n\}_{n \in Z^+}$ be a set of data points, and let each point x_n be associated with a pair $(\vec{x}_n, y_n)$, where $\vec{x}_n \in \mathcal{R}^D$ is a feature vector and $y_n \in \{0, 1\}$ is a label. There are D features for each data point x_n, and a total of N data points in the training set.

Definition 1 (Random Forest) *A Random Forest is an ensemble classifier composed of a collection of decision tree classifiers:*

$$\{\, h(X, Y, \Theta_k) \mid k = 1, \ldots, N \,\}, \tag{3.1}$$

where each Θ_k is an independent and identically distributed random vector [207].

Each tree in Random Forest is trained on a randomly selected subset of data through data bagging. Once the Random Forest is constructed, each tree casts a unit vote for a class of a given input $\vec{x}_n \in X$. Here, $Y = \{y_n\}_{n=1}^N$ denotes the set of class labels corresponding to the feature vectors $X = \{\vec{x}_n\}_{n=1}^N$.

3.1.3 Graph Encoding and Feature Extraction

In this step, we convert each decision tree into a graph structure. Our decision to convert a tree into a graph is motivated by two reasons:

- First, creating a graph from a decision tree allows us to generalize parent-child relations as weighted feature-feature edges, which create summary, interpretable representations of large decision trees (see Figure 3.5).

- Second, although extracting features of a decision tree is under-studied in the machine learning community, graph feature extraction is a well-studied problem. To illustrate; graph ML offers features for vertices (e.g., eccentricity [233]), edges (e.g., edge centrality [234]), subgraphs (e.g., modularity [235]) and graphs (e.g., clustering coefficient [236]) which can encode the trees of a random forest from various aspect. The encoding brings us closer to measuring the performance of tree ensembles in a principled way.

In the graph representation of each decision tree $T = V \times E$, each vertex $v \in V$ corresponds to a feature $d \in D$, and a directed edge between two vertices $e = \langle v_i, v_j \rangle \in E$ denotes a parent-child split of $d_i \rightarrow d_j$ in the decision tree. A decision tree can split the same feature multiple times into different levels; however, we do not create a duplicate vertex for each new split. Instead, we add duplicate edges from the parent vertex to the child vertex for new parent-child splits. As a result, the encoded graph may contain duplicate edges between vertex pairs. We also ignore feature values in splits during the entire encoding process because traditional graph features do not utilize attributes of vertexes.

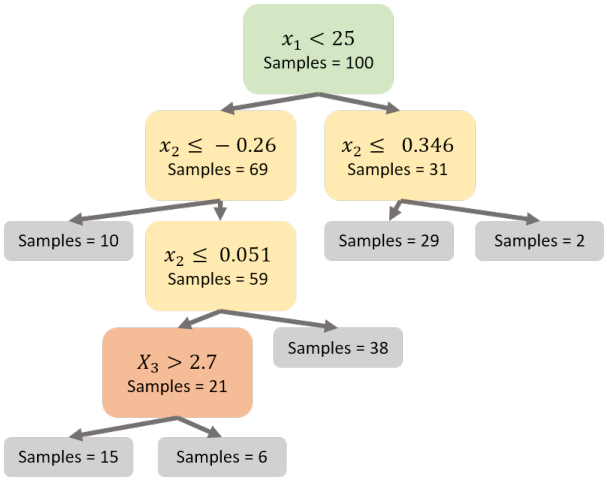


Figure 3.2: An example decision tree built on 100 data points and 3 features of the adult dataset. The tree utilizes features x_1, x_2, and x_3 and creates five feature splits (10 children nodes, where 6 of them are leaf nodes). In six of these splits, a further split is not required (shown as the gray leaf vertices). In four splits, additional feature splits are required.

Consider example in Figure 3.3 to understand how the decision tree is transformed into a graph representation. Since our goal here is to show the mapping between the decision tree and graph representation, we omit the example data set and hyperparameters that are used to construct deci-

sion tree in Figure 3.2. In the decision tree, let's assume that the feature x_1 of the root node has a split value of 25 in Figure 3.2. However, we do not retain this information in the graph representation of this decision tree in Figure 3.3. The reader should notice that there is only one node for each feature in the graph representation in Figure 3.3. However, there are two edges between x_1 and x_2 in Figure 3.3 since there are two child nodes that split on x_2 of the root node, which splits on x_1.

The graph kernel [237] or graph neural network [103] approaches can incorporate the multi-set of split values as vertex features in the learning process. However, such approaches may create high computational costs in the training process. For this reason, we leave the kernel and graph neural network approaches as future work.

We detail the process for encoding a graph from a decision tree in Algorithm 1 where we employ a breadth-first traversal to discover the parent-child relations in feature splits. The algorithm takes a decision tree as an input and outputs a directed graph structure such as the one shown in Figure 3.3. In the edge case where the decision tree has only a root node without any split, the algorithm would create a graph that consists of one vertex with special empty feature label without any edges. In all other cases, the algorithm creates at least one feature vertex since there must be at least one split at root node. In cases where tree depth is more than 1, the algorithm creates at least one edge.

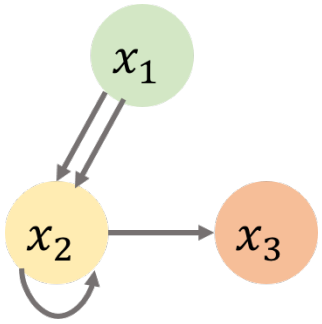


Figure 3.3: A graph representation of the decision tree shown in Figure 3.2. The four parent-child feature relationships are represented as directed edges. Feature x_2 includes a self-loop, indicating that it is used in two consecutive splits.

We outline three approaches to graph encoding: traditional features, graph kernel [237], or graph neural network [103]. Traditional features and graph kernel approach are considered shallow graph encodings because the output is a low-dimensional feature vector. Graph neural networks create deep encodings that are high-dimensional feature matrices. Deep encoders frequently outperform shallow encoders in ML tasks (see Tables 3 and 4 in [78]); however, the performance

Algorithm 1 Graph creation from a decision tree.

Require: Decision tree T
Ensure: Directed multi-graph $\mathcal{G}(V, E)$

Initialize $\mathcal{G}$ as a directed multi-graph
Initialize $\mathcal{S}$ as a multiset
$r \leftarrow$ the root of T ▷ Insert root into multiset $\mathcal{S}$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{r\}$
while $\mathcal{S}$ is not empty **do**
    $r_x \leftarrow$ first element in $\mathcal{S}$
    $\mathcal{S} \leftarrow \mathcal{S} \setminus \{r_x\}$
    if r_x is not a leaf **then**
        if $r_x \notin \mathcal{G}.V$ **then**
            $\mathcal{G}.V \leftarrow \mathcal{G}.V \cup \{r_x\}$
        end if
        if $r_x.left$ is not a leaf **then**
            if $r_x.left \notin \mathcal{G}.V$ **then**
                $\mathcal{G}.V \leftarrow \mathcal{G}.V \cup \{r_x.left\}$
                $\mathcal{S} \leftarrow \mathcal{S} \cup \{r_x.left\}$
            end if
            $\mathcal{G}.E \leftarrow \mathcal{G}.E \cup \langle r_x, r_x.left \rangle$
        end if
        if $r_x.right$ is not a leaf **then**
            if $r_x.right \notin \mathcal{G}.V$ **then**
                $\mathcal{G}.V \leftarrow \mathcal{G}.V \cup \{r_x.right\}$
                $\mathcal{S} \leftarrow \mathcal{S} \cup \{r_x.right\}$
            end if
            $\mathcal{G}.E \leftarrow \mathcal{G}.E \cup \langle r_x, r_x.right \rangle$
        end if
    end if
end while
return $\mathcal{G}$

comes at a great computational cost which we want to avoid for this task.

We used two main approaches to encode a graph in traditional graph features. We extract individual vertex features, such as vertex degree, in the first approach. Next, we design a readout function, which can be as simple as averaging values to pool vertex features. In the second approach, we ignore vertex features and directly extract graph-level features such as graph diameter and the number of strongly connected components. We follow a combination of both approaches and extract the following features on both vertex and graph levels:

- **Vertex-level metrics:**
  - In-degree, out-degree, and total degree of a vertex
  - Path distance to all vertices
  - Betweenness centrality
- **Graph-level metrics:**
  - Diameter of the graph
  - Number of vertices and edges
  - Directed three-vertex motif counts (16 types) [238]
  - Clustering coefficient

Additionally, we have tested several vertexes and graph features such as numbers of strongly and weakly connected components and hub/authority scores [239]. For reporting purposes, we only provide features that improve the performance of Topological Forest on out-of-bag test data.

Computational cost: Extracting graph encodings of an individual decision tree is a non-trivial operation. Features like betweenness centrality requires a time complexity of $\mathcal{O}(|V| \times |E|)$ [240] and motif counting requires $\mathcal{O}(|E|)$ [238]. We use the vanilla clustering coefficient implementation of the Jung library [241] whose time complexity is $\mathcal{O}(|V|^3)$. Most datasets on the UCI repository have less than 50 features. As a result, the graphs that we extract from decision trees are quite small, with less than 50 vertices. Furthermore, decision tree depth is bounded by the number of training data points. As a result, there are less than 200 parent-child relationships recorded as edges in these graphs. For these reasons, the graph representation creates quite small graphs (Figure 3.5) (i.e., $|V| < 50$ and $|E| < 200$ for most datasets). As a result, computational costs of graph representation and encoding in Topological Forest are negligible.

We average vertex features to find the corresponding graph feature (e.g., average vertex degree in the graph). Combining five averaged vertex features and 20 graph features, we create a decision tree representation $\vec{e} \in \mathcal{R}^{D'}$ where $D' = 25$. This representation allows us to compare and contrast decision trees which may use different features, split values, and tree depth.

3.1.4 Topological Clustering

We employ the highly customizable TDA tool Mapper [242] to analyze and cluster decision trees in the original vanilla random forest. Mapper was formally described in Section 2.1.2, where we introduced its construction in detail. Here, we briefly summarize only the components necessary for clustering decision tree encodings. The key idea behind TDA Mapper is as follows: Let T be a total number of observed trees and $\{\vec{e}_t\}_{t=1}^{T} \in \mathcal{R}^{D'}$ be a data cloud of tree encodings. For our dataset, $D' = 25$. We employ the t-distributed stochastic neighbor embedding (t-SNE) [232] as a lens to reduce the data into a two-dimensional space. The t-SNE converts similarities between data points to joint probabilities and minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. Next, we select a function $\xi : \{\vec{e}_t\}_{t=1}^{T} \to \mathbb{R}$ that filters data in one of the two dimensions.

Let I be the range of ξ, that is, $I = [m, M] \in \mathbb{R}$, where $m = \min \xi(\vec{e}_t)$ and $M = \max \xi(\vec{e}_t)$ in the dimension d'. We place data into overlapping bins by dividing the range I into a set S of smaller overlapping intervals of uniform length and let $u_j = \{t : \xi(\vec{e}_t) \in I_j\}$ be trees corresponding to features in the interval $I_j \in S$. For each u_j we perform a k-means clustering to form clusters $\{t_{jk}\}$.

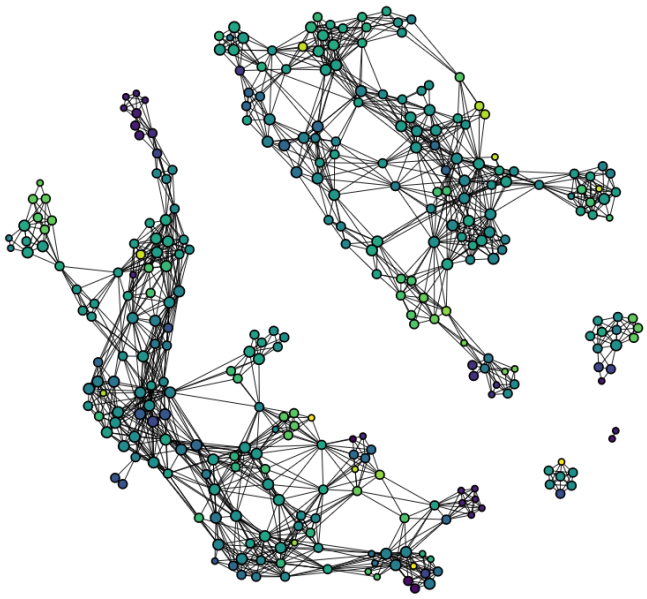


Figure 3.4: Mapper network of the Adult dataset with $cls = 5$, $n_{cubes} = 10$, and $overlap = 0.6$. Each node represents a soft cluster of decision trees from the Vanilla Forest (a tree can belong to multiple clusters). Edges indicate shared decision trees between clusters.

We analyze the empirical distribution of edge lengths where each cluster is merged to find the number of clusters. The merging criteria are based on the rationale that internal distances (i.e., within a cluster) are expected to be lower than external distances (i.e.,

in-between clusters), and distributions of internal and external distances are disjoint. Let $\{t_{jk}\}$ denote the k-th cluster of the j-th interval. We construct a cluster graph by transforming each cluster into a node and adding an edge between two nodes k and p if clusters $\{t_{jk}\}$ and $\{t_{lp}\}$ contain overlapping data points, i.e., $\{t_{jk}\} \cap \{t_{lp}\} \neq \emptyset$. Formally, the graph is called a TDA Mapper graph or a topological network.

After the graph transformation of decision tree clusters, TDA Mapper produces a low dimensional representation of the underlying data structure in the form of "cluster tree" graph $\mathcal{CT}$ where each "cluster" is a branch of some single connected component rather than a disconnected component on its own as in conventional clustering analysis. In Fig. 3.4, we show an example of "cluster tree" graph that is constructed from Adult Dataset [243].

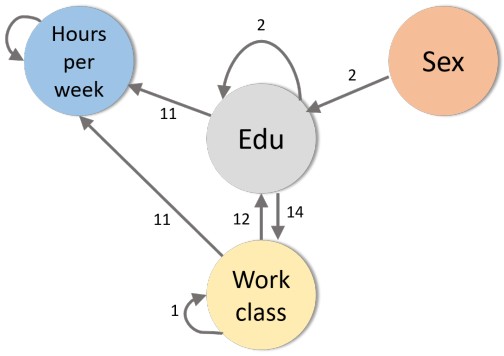


Figure 3.5: A sample graph representation of a single decision tree from the Adult dataset. Edge labels indicate the number of parent–child splits within the tree. Higher edge values represent more frequent splits from the source feature to the destination feature.

In the underlying mapper library, we can control the bin count with the n_{cubes}, and interval overlap with the $overlap$ parameters. A high $overlap$ value creates more edges between vertices, whereas higher n_{cubes} and k values create more vertices in the graph. As explained in Section 3.1.5, Topological Forest allows fine-tuning the inference process on the mapper network with various selection strategies that consider vertex sizes and edges. We consistently report good performance results across various datasets in our experiments with appropriate mapper parameters.

3.1.5 Ensemble Selection

As the previous section 3.1.4 explained, the TDA mapper network contains vertices that are clusters of decision trees and edges that show the relationship between neighbor clusters. This type of mapper network encodes topological insights into the trained decision trees. In particular, the network shape and the positions of vertices on the network convey helpful information about the

diversity and quality of decision trees. For example, consider the disconnected components of the mapper graph in Figure 3.4. Although we use a high overlap value of 0.6, which increases the probability of establishing an edge between two clusters, three groups of vertices (i.e., clusters) at the bottom right have no edges to the rest of the network; they form disconnected components. This phenomenon is inherently due to the dissimilarity of some decision trees and their encodings with respect to the rest of the decision trees. However, this observation of dissimilarity does not tell us anything about the utility of such isolated clusters and their decision trees. In one (and most likely) scenario, trees of the isolated vertices may have been built on useless features or noisy data points that add no predictive power to the forest. In a second scenario, the isolated trees may have been built on the most predictive features and data points. We design topological cluster selection strategies to test such hypotheses and evaluate the predictive power of clusters. We will use Figure 3.6 to explain three selection strategies: random, greedy mapper, and quality. In all three strategies, we first compute a cluster graph $\mathcal{CT}$ with a set of clusters defined as vertices on the graph. For simplicity, we will use graph notation and refer to a cluster k as $v_k \in \mathcal{CT}$.

Random: We randomly select n clusters and build a Random Forest from the union of the trees of the chosen clusters.

Greedy Mapper: We select n clusters that yield the highest AUC individually and create an ensemble from the union of the decision trees. We build an ensemble from the trees of each cluster and test the predictive power of the ensemble on validation data. Next, we test the ensemble on out-of-bag test data.

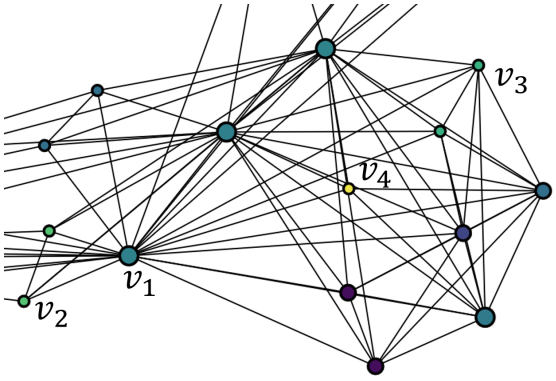


Figure 3.6: An inset of the Mapper network shown in Figure 3.4 (bottom right). Vertex size is proportional to the number of decision trees in each cluster: v_2, v_3, and v_4 each contain a single tree, while v_1 contains 13 trees. Vertex colors are artifacts of the Mapper library (binned mean IDs of the decision trees).

Quality: The Quality Strategy uses a homogeneity metric-based selection which we define as the average tree agreements over correct and incorrect predictions for all validation data points. If most

trees agree on a label, homogeneity will be high. However, the trees may agree on true (correctly predicted) and false (incorrectly predicted) labels. Clusters whose trees are homogeneous on true labels are preferable to those of false labels. In both cases, we hypothesize that if decision trees of a cluster contradict each other in classification (i.e., low homogeneity), the cluster must have been formed out of "bad trees".

We split the set of data points in $X = \{X_i \cup X_f \cup X_t\}$ w.r.t. the random forest classification. X_t comprises data points classified correctly by the random forest by majority voting, whereas X_f comprises data points that are misclassified by random forest by majority voting. Lastly, X_i includes points where the decision trees vote equally for true and false labels. Formally, we define true homogeneity as follows:

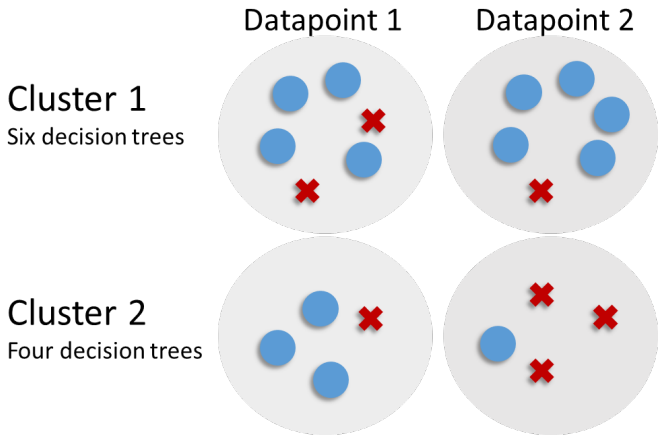


Figure 3.7: Decision tree votes of two clusters for two data points. Green circles indicate correct classifications, while red crosses represent incorrect ones.

Definition 2 (True homogeneity of cluster v_k) *Let h denote the number of decision trees in cluster v_k that correctly classify data point x. Then, the true homogeneity of v_k is defined as:*

$$H_{True} = \frac{1}{|X_t|} \sum_{x \in X_t} \frac{|\{h \in v_k \mid h(x, \Theta) = y\}|}{|v_k|} \tag{3.2}$$

Similarly, we define false homogeneity as follows:

Definition 3 (False homogeneity of cluster v_k) *Let h denote the number of decision trees in cluster v_k that misclassify data point x. Then, the false homogeneity of v_k is defined as:*

$$H_{False} = \frac{1}{|X_f|} \sum_{x \in X_f} \frac{|\{h \in v_k \mid h(x, \Theta) \neq y\}|}{|v_k|} \tag{3.3}$$

We calculate a cluster quality score for all existing clusters based on the true and false homogeneity scores as follows:

Definition 4 (Cluster Quality Index) *The Cluster Quality Index (CQI) measures the overall consistency of a cluster by combining its true and false homogeneity scores. It is defined as:*

$$CQI = w_{True} \times H_{True} - w_{False} \times H_{False}, \tag{3.4}$$

where

$$w_{True} = \frac{|X_t|}{|X|}, \quad w_{False} = \frac{|X_f|}{|X|}. \tag{3.5}$$

Example 1 (Homogeneity) *Consider the two topological clusters shown in Figure 3.7. Cluster 1 contains six decision trees, where four of them correctly classify data point 1. Cluster 2 contains four decision trees, and only one of them misclassifies data point 1. Similarly, only one decision tree of cluster 1 misclassifies data point 2, whereas three decision trees of cluster 2 misclassify data point 2.*

In true homogeneity computations, we will use data points 1 and 2 for cluster 1, but only datapoint 1 for cluster 2. Datapoint 2 is misclassified by the majority of cluster 2 decision trees. As a result, datapoint 2 will be used to compute the false homogeneity of cluster 2.

For cluster 1, $H_{False} = 0$ *and* $H_{True} = \frac{1}{2}(4/6 + 5/6) = 3/4$.

For cluster 2, $H_{False} = \frac{1}{1}(3/4) = 3/4$ *and* $H_{True} = \frac{1}{1}(3/4) = 3/4$.

We compute the cluster quality index for cluster 1 as $CQI_1 = (2/2) \times 3/4 - (0/2) \times 0 = 0.75$ *whereas* $CQI_2 = (1/2) \times 3/4 - (1/2) \times 3/4 = 0.0$.

As a result, our quality metric favors cluster 1 in its ensemble selection.

Homogeneity does not punish nor reward Tie cases where decision trees vote equally for correct and incorrect labels. In Tie cases the number of miss-classifying decision trees is equal to number of correctly classifying decision trees in a cluster. After indexing all clusters, we select the highest quality clusters and create an ensemble out of their trees.

Dataset	Instances	Attributes	Majority Class
Diabetes [244]	100K	21	78%
Binary Connect-4 [244]	68K	42	65%
Adult [243]	33K	14	76%
Binary Poker [245]	25K	10	99%
Binary Letter Recognition [246]	20K	16	80%
Nursery [247]	13K	6	50%

Table 3.1: Dataset characteristics.

Quality Top-x: Differing from the Quality approach, after selecting Top-K clusters based on the index score, we limit tree selection to Top-1, Top-2, or Top-5 best trees based on the tree score index in each cluster. For calculating the tree score index inside each cluster, we count the number of positive and negative collaborations of each tree. If a tree contributes to true homogeneity, it is rewarded with $+1$, and if it contributes to false homogeneity, it is penalized with -1. The top 1, 2, and 5 trees with the highest rewards are selected for an ensemble.

3.2 Experimental Results

We have released our source code at `https://github.com/cakcora/MultiverseJ` where we have developed a complete classification Random Forest in Java.

3.2.1 Datasets

For the experiments, we selected six classification datasets from the UCI Machine Learning Repository with two selection criteria to ensure robust classification results: the number of data points in a dataset must be more than 10K, and the dataset should have six or more features. As Table 3.1 shows, the selected datasets have diversified population sizes, attributes, and majority-class percentages. Diabetes, Adult, and Nursery data sets have binary labels while other datasets have more than two classes.

Since our focus in this work is binary classification tasks, we reduce the number of classes for non-binary datasets in the following way. For the Poker dataset *nothing in hand*, *one pair*, *two pair* and *three of a kind* classes were re-labeled as *low probability to win*. Other classes were classified as *high probability to win*. In the Connect-4 dataset, *draw* and *loss* were merged in *not-win class* along with the existing *win* class. In the Letter Recognition dataset, the first 13 letters were merged in a 0 class, and the last 13 letters were merged in a 1 class.

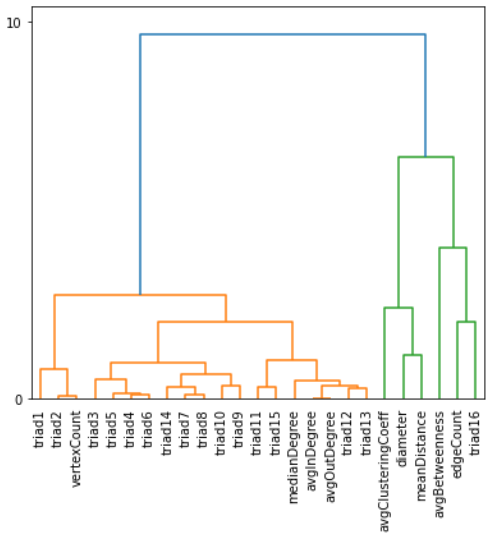


Figure 3.8: Hierarchical clustering of the 25 features used in graph encodings on the Adult dataset. The y-axis shows distances between features. Triad 16 co-clusters with edge count, while other triads mostly cluster together. Distances are relatively small (e.g., between triads 12 and 13) compared to those between edge count and triad 16.

3.2.2 Features and Graph Encodings

Dataset	Greedy Mapper	Quality	Q-Top1	Q-Top2	Q-Top5	Random	Vanilla Forest
Diabetes	**0.63±0.008**	0.56±0.010	0.56±0.010	0.56±0.010	0.56±0.010	0.59±0.020	0.65±0.006
Binary Connect-4	**0.81±0.010**	0.79±0.010	0.78±0.010	0.80±0.010	0.79±0.010	0.77±0.020	0.85±0.005
Adult	**0.89±0.005**	0.88±0.005	0.88±0.005	0.88±0.006	0.88±0.005	0.87±0.010	0.90±0.004
Binary Poker	**0.76±0.100**	0.54±0.150	0.53±0.130	0.53±0.160	0.52±0.150	0.41±0.170	0.77±0.099
Binary Letter Recog.	0.94±0.008	0.94±0.009	0.95±0.009	**0.95±0.008**	0.94±0.010	0.92±0.020	0.95±0.006
Nursery	0.53±0.010	0.52±0.010	**0.54±0.010**	0.53±0.010	0.52±0.010	0.47±0.010	0.48±0.015

Table 3.2: AUC of different methods using 10% of trees. Greedy Mapper AUC is, on average, 98% of the Vanilla Forest AUC.

During the training phase (See Section 3.1.3), we extract a graph from each decision tree of the vanilla Random Forest for all data sets. Next, we extract features from each graph and use the TDA Mapper to create a topological network.

We use hierarchical clustering to show the similarity of features over trees. Although hierarchical clustering is not used by proposed approach in the topological forest, we use graph features extracted from each decision tree and pass them to the TDA mapper. If these features are very correlated and clustered together in the feature space, this will reduce the efficiency of TDA mapper step as it process these features as an input. Therefore, visualization of this clustering information

is very critical from experimental results perspective.

In Figure 3.8 we show the relationship between the features where we hierarchically cluster the extracted features from graphs. Triads are the directed 3-vertex motifs [238], and they co-cluster well, except for triad 16, which is the strongest (closed) triangle motif. Unsurprisingly the median, average in (avgIn), and average out-degree (avgOut) co-cluster with the triads 12 and 13, which are closed triangle motifs. However, these degree-statistics-based features do not co-cluster with betweenness or edge count. This behavior arises because we one-hot encode categorical features of the data, creating many vertices (i.e., new features) in the graph. Such graphs have many edges, but they exhibit low clustering coefficients and few connected triads.

3.2.3 Tuning for TDA Mapper

We have experimented with a set of TDA Mapper parameters for each dataset and reached the best overall AUC performance for $n\_cubes = 10$, $perc\_overlap = 0.6$ and $number\_of\_clusters = 5$. The performance is not sensitive to the $number\_of\_clusters$ within a cube or num_cubes. However, $perc\_overlap$ determines how connected the topological network will be. A more connected network implies more shared decision trees between clusters. As a result, clusters include less similar decision trees that create lower classification homogeneity (i.e., trees vote differently on data points).

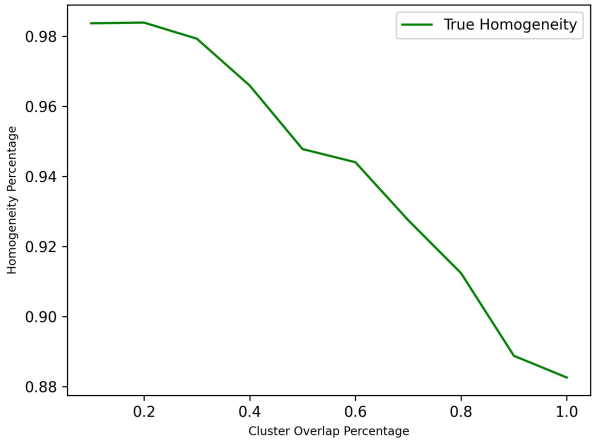


Figure 3.9: Homogeneity in the Adult dataset with varying cluster overlap percentages ($perc\_overlap$).

Figure 3.9 shows that our methods are robust against the overlap percentage. An increasing overlap percentage lowers homogeneity, but the decrease is not drastic (from 0.99 to 0.89), which shows that features of decision trees are diversified enough to create separate clusters even when

we allow for a higher overlap. As a result, trees in a cluster are similar and vote similarly. The high homogeneity is a significant result because, as we show in Section 3.2.4 it will enable us to use fewer decision trees from a cluster but reach similar performance in classification.

3.2.4 Classification Results

Our experiments partition each dataset into 80% training, 10% validation, and 10% out-of-bag test subsets. We ran each experiment 30 times, where we used random seeds to select the partitions in each replica. A vanilla forest has been created on the training subset in each replica. The vanilla forest has been tested for any possible overfitting by comparing the overall accuracy of the train and test set, which were close to each other.

In the next step, we employ the Kepler Mapper [33], a Python implementation of TDA Mapper, with built-in visualization, dimensionality reduction, and clustering options on the vanilla forest decision trees to create and visualize our TDA results.

We report our results in terms of ROC AUC and run-time performance. We compared six ensemble selection strategies with vanilla random forest. The definition of each strategy is explained in Section 3.1.5. Table 3.2 shows the mean and standard deviation of AUC results over the 30 replicas. As Table 3.2 shows, we find that Greedy Mapper has the best AUC on four datasets among all ensemble selection strategies. Greedy Mapper also loses only 2% of AUC with significantly smaller (10x, 30 trees) Topological Forest size compared to the vanilla Random Forest, which has 300 decision trees. In the Poker and Letter Recognition datasets, where we have substantial class imbalances (99% and 80%, respectively), Greedy Mapper and Q Top5 AUC are close to the Vanilla Forest in terms of AUC, which offers evidence that selecting the best clusters can also help with class imbalance.

We also reported the prediction quality of three random forest policies that have 5%, 10% and 100% of decision trees as proxy for all approaches that tune the number of decision trees. Table 3.3 shows the results for randomly picking up decision trees. Since topological forest has 10% of decision trees, comparing these three policies gives an idea of the upper and lower bound on

prediction quality when number of trees is purely tuned. From these new experimental results, we find that the quality and diversity of decision trees are very important factors in further improving the prediction quality of random forest. As shown in Table 3.3, the Greedy Mapper approach produces random forest with 10% of decision trees, which is better than random policy that uses 10% of decision trees.

Dataset	5% trees	10% trees	100% trees
Diabetes	0.58±0.007	0.60±0.005	0.65±0.006
Binary Connect-4	0.76±0.005	0.79±0.007	0.85±0.005
Adult	0.87±0.005	0.88±0.006	0.90±0.004
Binary Poker	0.40±0.14	0.55±0.16	0.77±0.10
Binary Letter Recog.	0.91±0.006	0.93±0.009	0.95±0.006
Nursery	0.45±0.014	0.46±0.015	0.48±0.015

Table 3.3: AUC for random tree selection policies.

Table 3.4 shows the computational cost of the inference task from our best approach, Greedy Mapper, and the Vanilla Random Forest when deployed on the test data. Here, the computational cost during the inference task is defined as the total CPU time spent for all computations, including the cost of inference from individual decision trees. In the best case, Greedy Mapper improves computational cost 217 times compared to Vanilla Forest in the Adult dataset, whereas the improvement is around 22 times for the Binary Poker dataset. On average, Greedy Mapper reduces the inferring time on the test data with such high values for two reasons. First, the number of decision trees in the ensemble is reduced by 90% or more after applying ensemble selection strategies. Second, trees that are built on *low quality* features grow too complex to fit the training data better. However, such deeper trees are unlikely to appear in the best-performing clusters. As a result, such trees are excluded in Greedy Mapper, contributing to better run-time results.

We also compared the computational cost of all approaches for the training task in Table 3.5. Here, the computational cost during the training task is defined as the total CPU time spent for all computations, including training individual decision trees and any other downstream steps like graph encoding of decisions trees or running the TDA Mapper. While, on average, Greedy Mapper is 4.1 times slower than Vanilla Forest for the training task, it is 113 times faster than Vanilla Forest in the inference task. Thus, the trade-off between the cost of training and the computational

performance in the inference task significantly justifies the use of Topological Forest.

Dataset	Vanilla Forest	Greedy Mapper	Speed Up Ratio
Diabetes	842.16±146.34	4.76±3.85	177×
Binary Connect-4	382.33±63.60	2.53±1.07	112×
Adult	294.53±56.38	1.36±2.98	217×
Binary Poker	36.76±2.16	1.68±0.47	22×
Binary Letter Recog.	145.10±7.00	1.14±0.69	127×
Nursery	44.03±9.27	1.73±0.63	25×

Table 3.4: Inference cost (milliseconds).

Dataset	Vanilla Forest	Greedy Mapper	Slow Ratio
Diabetes	7.51±0.41	18.49±0.39	2.46×
Binary Connect-4	1.61±0.08	7.37±0.22	4.57×
Adult	5.04±0.53	14.01±0.45	2.77×
Binary Poker	1.38±0.13	6.11±0.22	4.42×
Binary Letter Recog.	4.40±1.11	11.73±0.88	2.66×
Nursery	0.58±0.09	4.48±0.11	7.72×

Table 3.5: Training cost (seconds).

The efficiency of the Topological Forest is mainly related to the computational time savings in inferral. Topological Forest uses 10% of the trees and yields comparable performance to Vanilla Forest. Furthermore, Topological Forest performs better than the Vanilla forest in Nursery and Letter Recognition datasets (Table 3.2). In this sense, our method has better AUC performance for some datasets as well.

3.3 Conclusion

We have developed an open-source implementation of our novel ML method Topological Forest. Our approach builds on a Vanilla Random Forest implementation but uses topological methods to create a refined ensemble that has a smaller number of decision trees and better trees in the forest. On average, Topological Forest speeds up inference time by more than 100x for a cost of at most 2% reduction in AUC. The results of our experiments suggest that the topological forest is considerably faster than random forest. Moreover it needs less resources and efforts compared to neural networks.

Chapter 4

Chartalist: Labeled Graph Datasets for UTXO and Account-based Blockchains

Kiarash Shamsi, Friedhelm Victor, Murat Kantarcioglu, Yulia Gel, Cuneyt Gurcan Akcora

Published in the Thirty-Sixth Conference on Neural Information Processing Systems (NeurIPS 2022), Datasets and Benchmarks Track

Blockchain is an emerging technology that has already enabled a wide range of applications, from cryptocurrencies (e.g., Bitcoin and Ethereum) and digital asset management to accountable data sharing. Most of the blockchain information can be represented as single and multilayer graphs composed of different types of nodes or edges. For example, for the Ethereum blockchain, the transactions that trade a digital token or asset can be seen to create a unique layer between the involved addresses (i.e., nodes of the transaction graph). Due to the increased popularity, analyzing the graph data stored on blockchains has emerged as an essential data science and machine learning (ML) problem. For example, building ML models (e.g., graph neural networks) using cryptocurrency blockchain graphs has direct applications for ransomware payment tracking, price manipulation analysis, and money laundering detection.

Analyzing the massive blockchain data today requires significant efforts to extract the underlying graph by running a Bitcoin client or paying for commercial APIs (e.g., etherscan.io) to download transaction data. Even when we extract transaction data by any means, we must allocate considerable resources to construct blockchain graphs. For example, a leading open-source Bitcoin parser BlockSci [248] requires 60GB of memory to build the Bitcoin transaction graph. Ethereum poses even greater challenges as its data is bigger and requires smart contract analysis to discover internal transactions. Combined with data size and complexity issues, blockchain research also lacks labeled data for many significant problems – these issues plague ML on blockchains and force researchers to develop their pipelines to work with blockchain data.

To enable broader ML research in this emerging area, we created a dataset repository named Chartalist, containing cleaned and labeled data (e.g., known ransomware payment addresses) combined with open-source data loaders and graph extractors for easy analysis and model building using the provided data. To the best of our knowledge, Chartalist is the first attempt to systematically organize blockchain data for the broader ML community and provide a set of ML tasks defined for appropriate blockchain datasets.

In addition to facilitating ML applications on blockchains, our dataset further enables ML model development for dynamic multilayer graphs. In particular, although there are many publicly

available multilayer graph datasets, the existing graphs are either small, static, focus on a specific case when the set of nodes is fixed across layers (i.e., multiplex networks), or lack ground truth information. To our knowledge, *no dynamic multilayer graphs with ground truth for anomaly detection are publicly available*. In contrast, our dataset provides large-scale, dynamic multilayer networks where nodes, edges, and edge weights evolve. Furthermore, adopting and curating major blockchain events allow us to complement such multilayer graphs with the ground truth information on event anomalies. The graph data given in Chartalist can be used for a broad range of ML tasks, such as forecasting, link prediction, and node classification with graph neural networks (GNNs).

Sustainability of Chartalist. Data Security and Privacy Lab of UT Dallas and Future Data Lab of the UManitoba are committed to hosting, managing, and updating the Chartalist Data Repository at least quarterly.

Chartalist contributes to graph ML research in the multifold ways:

- Chartalist removes the burden of downloading, parsing, and cleaning blockchain data, the key bottleneck for analysis of blockchain data by the broader ML community. Chartalist aims to be a reliable supplier of Bitcoin and Ethereum blockchain graphs and plays a vital role in emerging graph machine learning.
- Chartalist intends to be the focal point for curated address and edge labels that are collected from the community.
- Chartalist provides transaction and block extraction and graph creation tools to facilitate graph machine learning on blockchains.

4.1 Overview of Chartalist Data Commons

Chartalist has three main components: 1) A holistic view of blockchains and formulations of graph machine learning tasks. 2) A comprehensive ecosystem of tools and community resources

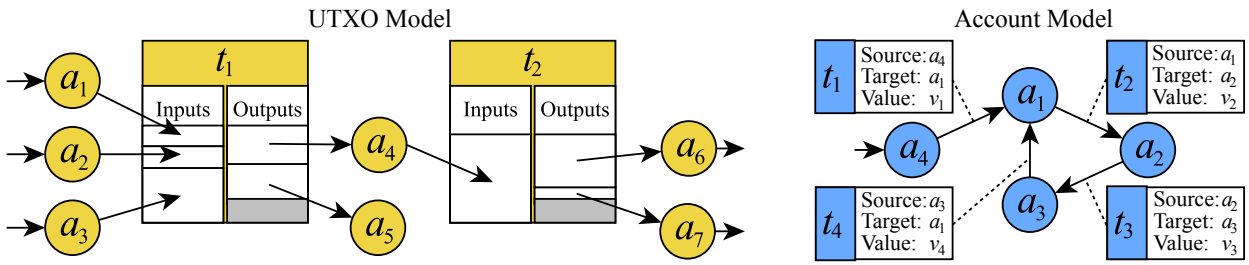


Figure 4.1: In the UTXO model, transactions (i.e. t_1, t_2) between addresses (a_i) can consist of multiple inputs and outputs. In the account model, every transaction has only one source and target address.

to support blockchain data analytics. 3) A set of boards to support performance comparison and benchmark for the tasks. As a first step, we now provide an overview of blockchain types that shape our approach in ML model building using the blockchain transaction network.

4.1.1 Blockchain Data Types

Due to the numerous advantages of blockchain technology, blockchain-based platforms are growing very fast. Although the fundamental concepts behind all these networks are very similar, They may use various architectures with different data types to implement their network. For instance, Bitcoin (2008) [249] is the first implementation of blockchain technology where coin transaction are the primary data to be stored; However more recent platforms such as Ethereum has been created to store more complicated structures like software code, called smart contracts.

Blockchains can be categorized into two broad lines in terms of transaction type: unspent transaction output (UTXO)-based (e.g., Bitcoin, Litecoin) and account-based (e.g., Ethereum) blockchains [250]. The difference between UTXO and account-based models profoundly impacts blockchain networks [251]. Figure 4.1 illustrates UTXO and account model transaction networks. An analogy to account-based blockchains is bank accounts that can be used to make payments and keep a remaining balance. In these blockchains, a transaction has exactly one input and one output address (an address is a unique account identifier on the blockchain transaction network). An address may be used to receive and send coins multiple times. The resulting network is similar to traditional social networks, which implies that social network analysis tools can be directly applied

to account networks. Smart contracts are a more recent concept mainly available on account-based platforms [252]. These contacts have their own account addresses which can be called to enable certain actions such as buying/selling digital tokens [253]. An account-based smart contract platform employs code accounts to manage application-specific states, such as asset balances. These state changes can result in a diverse set of asset networks. The networks may share the same set of account addresses, allowing us to link activity, observe asset flows in the network, and create multilayer networks (i.e., flows associated with each asset result in a different layer).

In turn, the second type (i.e., UTXO-based blockchains), such as Bitcoin, have made design choices to blur the association between input and output addresses of a transaction [254]. First, transactions may involve more than two addresses. Users often use many addresses, and it is not straightforward to establish which input addresses of a transaction sent coins to a specific output address. Consequently, the ways of processing data are entirely different across networks. Thus, knowing the data types and network interactions is essential for implementing ML models on top of these networks.

4.1.2 Machine Learning Datasets and Learning Tasks

Chartalist is structured around blockchain types and currently serves UTXO and account-based blockchains. In the following, we describe a set of problems that can be addressed based on the datasets we provide and framed as blockchain graph ML tasks.

1. **Address Clustering**: This task aims to identify which addresses are co-owned by an entity [255]. To approach this task, protocol design choices and application-specific user behaviors may be exploited.

2. **Address and Transaction Type Classification**: This task aims to determine whether addresses and transactions belong to a specific class type. For example, certain addresses behave like cryptocurrency exchanges or certain transactions exhibit a malicious character [256, 257, 7, 258].

3. **Anomalous Transaction Pattern Detection**: This task aims to identify anomalous transactions so that, i.e., their distinct temporal patterns can be summarized. For example, ransomware payments typically involve similar, and the receiving addresses use certain transaction types [259, 260, 261, 262, 263].

4. **Multilayer Network Analysis**: This task aims to predict events and phenomena that emerge through the simultaneous use of multiple assets, which can be studied as multilayer networks [264].

5. **Coin Tracking Across Blockchains**: This task aims to uncover transaction flows between blockchains. For example, Bitcoins can be temporarily exchanged for privacy coins such as Monero or Zcash for obfuscation purposes.

6. **Price Analytics**: The goal of this task is to use observable blockchain activity, such as within the transaction network, to predict asset or coin prices [265, 266, 267, 268, 269, 270, 271, 272, 273] and the associated analysis of volatility [274, 275].

4.1.3 Tools and Frameworks for Blockchain Data Analytics

We curate our transaction network data by installing a blockchain node and connecting it to the P2P network. Afterward, we use a tool such as Bitcoin4J or Blockchain-ETL (`https://github.com/blockchain-etl`) to parse Bitcoin, Dash, and Ethereum. Although blockchain data can be accessed publicly, creating the transaction network is arduous. First, blockchain sizes have become prohibitively large to run data analytics in a single machine. Second, linking transactions to develop a blockchain network requires domain expertise, which many ML practitioners lack. Third, as BlockSci warns "cryptocurrencies continue to update their protocols, they may lose compatibility with BlockSci parsers". Chartalist aims to be a reliable supplier of blockchain graphs and meet a vital role in graph machine learning to combat this. Many blockchain analytics companies offer REST APIs to access blockchain data and networks. Examples are etherscan.io, blockcypher.com, and infura.io. However, APIs are costly or allow limited access to the API through a

rate limit. For example, etherscan.io allows only five queries per second for the Ethereum network.

We provide datasets for the aforementioned tasks on Bitcoin, Dash, and Ethereum. **With the community's help, we aim to provide a labeled dataset for each task on Bitcoin, Ethereum (and its asset networks), Monero, Zcash, Dash, and Ripple** and continuously expand the number of blockchains covered. All datasets contain transaction time, metadata, provenance information, and curated labels. Chartalist datasets vary in size between 50 thousand and 40 million edges.

In addition to transaction graphs, we have curated address labels for the tasks. They originate from the domains of ransomware attacks, asset networks, and decentralized finance. In particular, we have used the label cloud of `etherscan.io` to curate Ethereum address labels (see [257]). We have curated our Bitcoin ransomware address labels from Montreal [276], Princeton [277] and Padua [278] studies. We have provided our code and data under a CC BY-NC (Attribution-NonCommercial) license.

Limitations. New Blockchain addresses are created daily. Hence new address labels may be missing from our labeled Ethereum dataset. On Bitcoin, many entities pay the ransom without disclosing the payment [279]. As such, our ransomware dataset cannot be complete. Our goal in sharing the dataset is to encourage detecting such undisclosed payments.

Ethics and Privacy. We must stress that all transaction data is theoretically already available to users with the resources (fast SSDs, large RAM, and disk space) to synchronize blockchain clients with the respective networks and manually extract information. An individual user's privacy depends on whether personally identifiable information (PII) exists about any of their blockchain addresses, which serve as account handles and are understood to be pseudonymous. Our data does not contain any PII. However, it is possible that if such PII were to be obtained from other sources, the datasets we are proposing here could lead to further privacy-related implications. This scenario cannot be prevented *because all raw data is already accessible to users controlling significant hardware resources*. Real-life identities may be discovered by using IP tracking information, *which we do not have nor share*. Furthermore, we curate all labels from public information or publicly available models (e.g., etherscan.io tags for Ethereum). The data does not contain individually

identifiable information or personal attributes. Hence, there is no need for any anonymization of the data.

Most of our labels are published by Blockchain entities themselves. For example, exchanges publish their Ethereum addresses. Ransomed entities publish Bitcoin ransom addresses. Note that our Bitcoin label dataset consists of solely such disclosed addresses. We, for example, have not released a set of suspicious (potentially ransomware) addresses that we found in our previous work [256] for privacy reasons. Furthermore, we provide an online tool to appeal for the exclusion of an address from our dataset with a promise of responding to such requests promptly.

4.2 Tasks and Baseline Experiments

This section describes the first three tasks in more detail and provides baselines for them. We refer the reader to our website for more details on the other tasks.

4.2.1 Ransomware Address Classification on Bitcoin

Blockchain transactions can be created anonymously, and participation in the network does not require identity verification. Hackers can demand a ransomware payment by delivering a public blockchain address (i.e., a short string) to a sender using anonymity networks such as Tor [280].

Why is the task important? Malicious actors have noticed blockchain technology's ease of usage and worldwide transaction availability. The pseudo-anonymity of users in cryptocurrencies such as Bitcoin has attracted the interest of a diverse body of criminals, transnational terrorist groups, and illicit users. Cryptocurrency-related crime and criminal abuse of blockchain technologies are nowadays recognized as the fastest-growing type of cyber-crime. Ransomware has emerged as a critical threat to infrastructure in many countries. Using cryptocurrencies for ransomware payments appears to be substantially more prevalent than has been previously realized. As noted by Hernendex-Castro et al. [279], among the respondents to their survey, "the prevalence of the CryptoLocker ransomware seems much higher than expected". Detecting undisclosed payments and

discovering ransoming actors have become critical tasks in blockchain data analytics.

Ransomware analysis has two sub-problems: identifying undisclosed payment addresses of a known ransomware family and detecting the emergence of a new ransomware family. We formulate and include the second problem as a particular case of the first one and formalize the problem as node label prediction in a directed weighted graph.

Experimental Setup. This task uses the Bitcoin transaction graph in Chartalist. Using a time interval of 24 hours, we extracted daily transactions on the network and formed the Bitcoin graph. We have filtered out the network edges that transfer less than $BTC\ 0.3$ for computational efficiency since ransom amounts are rarely below this threshold. We have labeled 24,486 addresses from 26 ransomware families (see [256]).

Let $f_1, \ldots, f_n$ be labels of known ransomware families observed until time point t. We set f_0 as the label of addresses that are **not** known to belong to any ransomware family, and we assume them to be **white addresses**. Let rs be a known ransomware family of interest. Let $\tilde{Y}_t \subseteq Y_t$ be such that $\forall y_j \in \tilde{Y}_t, y_j \in \{f_0, f_{rs}\}$. Let $\{a_{l+1}, \ldots, a_{l+z}\}$ be a set of addresses whose set of labels $Y_{t'} = \{y_{l+1}, \ldots, y_{l+z}\}$ is unknown. Let $t' > t$, and $t < \min\{t_{a_{l+1}}, \ldots, t_{a_{l+z}}\}$. The problem is to predict all addresses $a_m \in \{a_{l+1}, \ldots, a_{l+z}\}$ such that $y_m = f_{rs}$, using the transaction graph and history (X_t, Y_t).

Baseline. We approach the baseline detection task by extracting six features [256] from the daily Bitcoin transaction network for each address. We have designed the graph features to quantify ransomware operators' specific obfuscation patterns. Afterward, we employ tree bases methods, clustering, and naive similarity search on the feature matrix of all Bitcoin addresses.

Address clustering identifies which blockchain addresses are controlled by the same real-life entity/person. A popular method involves finding input and output address associations in a transaction. For example, on Bitcoin, two heuristics are widely used as baselines [281]. **Co-spending heuristic:** "If two addresses are inputs to the same transaction, the same user controls them". **Transition heuristic:** "If we observe one transaction with addresses A and B as inputs and another with addresses B and C as inputs, then we conclude that A, B, and C all belong to the same

user". Using co-spending and transition heuristics with all history, we discover only 40 unique addresses from CryptoLocker, CryptoWall, CryptoTorLocker2015, CryptoTorLocker2015 families.

Naive similarity search: An interesting benchmark for detecting undisclosed payments from existing families is to compute the similarity of $X_{t'}$ addresses to the past addresses in X_t. If existing families exhibit repeating patterns over time, the similarity search can match new addresses to known ransom addresses. Figure 4.2 shows that this strategy may be effective. Using exact matches to known ransom patterns of the past, a six-day similarity search reveals more than 50 addresses each day while predicting a maximum of 73 false positives on 2013, day 336. These results offer supporting evidence of the utility of our features. However, this naive approach creates 21,371 FP addresses for the period we considered, making it unfeasible for operational use by security analysts.

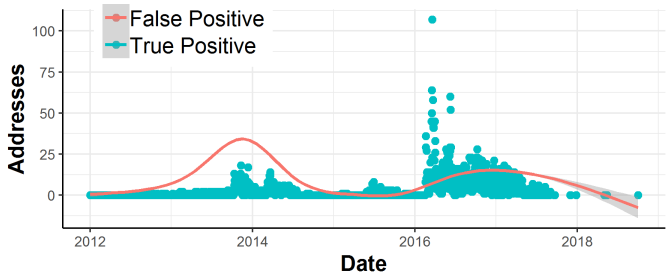


Figure 4.2: Ransomware detection with exact feature matches.

Results.

As Table 4.1 shows, baseline models yield better recall than precision. Similar to our (proposed) topological data analysis method (TDA), which performs the best, the DBSCAN clustering algorithm can ignore data points in its model building; two of the best non-TDA results are delivered by DBSCAN models. In the best TDA models for each ransomware family, **we predict 16.59 false positives for each true positive**. In turn, this number is

Table 4.1: Detecting undisclosed ransomware payment addresses.

RS	Method	TP	FP	FN	TN	Prec	Rec
Locky	TDA	451	2350	50	8221	0.161	0.900
	COSINE	2395	41681	3990	146369	0.054	0.375
Crypto Wall	TDA	217	3087	155	11200	0.066	0.583
	DBSCAN	728	18960	794	16913	0.037	0.478
Crypto Locker	TDA	439	9686	212	22129	0.043	0.674
	DBSCAN	935	42771	295	11316	0.021	0.760
Cerber	TDA	187	5174	459	23027	0.035	0.289
	XGBOOST	1606	47307	7279	374169	0.033	0.181
Crypt XXX	TDA	77	2460	271	11057	0.030	0.221
	COSINE	589	20872	610	42952	0.027	0.491

27.44 for the best non-TDA models.

4.2.2 Address Classification on Ethereum

The Ethereum project [282] was created in July 2015 to provide smart contract functionality on a blockchain. Smart contracts are Turing-complete software codes that are replicated across a blockchain network. Some smart contracts implement mechanisms to allow the trading of digital assets, known as tokens [283], on the blockchain. We refer to such a smart contract as an *asset contract* and use the term asset interchangeably. Like cryptocurrencies, an asset is transferred publicly between accounts (addresses). It may have an associated value in fiat currency which is arbitrated by asset demand and supply in the real world. These asset transfers create a directed, weighted multigraph between Ethereum addresses.

Why is the task important? Blockchains have enabled a new class of decentralized services, such as lending protocols, data oracles, stablecoins, and decentralized exchanges. The entities can employ regular accounts or smart contracts to facilitate information exchange or asset trades among blockchain addresses, lend coins, and provide a forum for financial decision-making. Although blockchain user addresses are pseudonymous, some entities publish or confirm their addresses to foster trust and security in their communities. Furthermore, analytics companies curate addresses of blockchain entities to analyze blockchain dynamics in real-time. However, manual curation is time-consuming, costly, and error-prone in the decentralized finance ecosystem that trades billions of US dollars worth of assets.

By modeling transaction behavior, we automate address identification and address type (e.g., proxy address of an exchange). Here we have formulated this task as identifying the most central nodes in Ethereum token networks (see [257]). Token networks offer more; we have also created a multilayer network out of token networks to detect global temporal anomalies (e.g., when blockchain technology is banned in a country). However, due to space limitations, the anomaly task is not explained here (see [264]).

Experimental Setup. This task uses the token networks dataset in Chartalist. Our address labels

are taken from Etherscan,[1] a prominent Ethereum block explorer. Covering 149 centralized and decentralized exchanges, we have manually collected 296 account addresses that were listed publically in May 2020. Labeled data is scarce; not all the top token networks have multiple exchange nodes taking part in them. For the following evaluation, we only consider those token networks that contain at least ten such labeled exchange nodes, which reduces the number of networks considered to 28. We first evaluate the top 100 networks by the number of transfers.

We evaluate the task of finding top n labeled nodes in token networks, using precision and recall measures (shown as $P@n, R@n$ respectively), where $AP@n$ and $AR@n$ denote the averages across multiple networks. Nodes are ranked by their core membership in descending order.

Baseline. We employ k-core and weighted k-core decomposition as the baseline and add our recent AlphaCore [257] decomposition results for comparison.

Both k-core and weighted k-core are defined for undirected networks. As the token networks are directed multigraphs, we use a node's neighborhood size instead of their degree. For the other networks, degree equals neighborhood size. Furthermore, we compare against modified variations of k-core (rows 8 and 9 of Table 4.2) and weighted k-core (rows 11 – 13), using different input features.

Traditional core decomposition algorithms propose little in alleviating data challenges. For example, weighted k-core suggests using $\alpha = \beta = 0.5$ for combining properties, but improving over this arbitrary choice is left as future work. Most networks' edge weights are skewed, and long tails complicate scaling and normalization issues. For example, max edge weights in token networks can reach 10^{13}, whereas node degrees are typically less than 3. With such a difference in scales, node property combination becomes a challenge in weighted k-core.

Results. The first insight is that data depth-based AlphaCore [257] performs best compared to the other two core decomposition algorithms. The results indicate that addresses of centralized and decentralized exchanges can be discovered by using in and out-degrees of addresses. This is a promising result to automate address type discovery in Ethereum asset networks.

[1] `https://etherscan.io/labelcloud`

Table 4.2: Performance comparison with a ranking task using 28 token networks. Performance is indicated using (average) precision and recall at k. We compare core decomposition methods with a variety of input features. Feature N_{in} refers to the number of neighbors with incoming edges, and S_{in} to the sum of weights (strength) of these edges.

	Algorithm	Input features	AP@10	AP@20	AP@50	AR@10	AR@20	AR@50
1	**AlphaCore**	N_{in}, N_{out}	**0.54**	**0.45**	**0.30**	**0.10**	**0.16**	**0.25**
2	**AlphaCore**	N_{out}	0.50	0.44	0.27	**0.10**	**0.16**	0.23
3	**AlphaCore**	N_{out}, S_{in}, S_{out}	0.42	0.33	0.21	0.07	0.11	0.18
4	**AlphaCore**	$N_{in}, N_{out}, S_{in}, S_{out}$	0.41	0.33	0.22	0.07	0.11	0.19
5	**AlphaCore**	N_{in}	0.50	0.41	0.24	0.10	0.15	0.21
6	**AlphaCore**	N_{in}, S_{in}	0.39	0.30	0.19	0.06	0.10	0.17
7	k-core	N	0.36	0.26	0.14	0.06	0.08	0.11
8	k-core	N_{out}	0.22	0.16	0.11	0.03	0.05	0.10
9	k-core	N_{in}	0.13	0.09	0.06	0.03	0.04	0.06
10	w. k-core	N, S	0.37	0.30	0.19	0.06	0.10	0.16
11	w. k-core	N_{in}, S_{in}	0.33	0.23	0.15	0.06	0.07	0.13
12	w. k-core	N_{out}, S_{out}	0.28	0.29	0.21	0.05	0.10	0.18
13	w. k-core	N_{in}, N_{out}	0.04	0.03	0.02	0.01	0.01	0.02

4.2.3 Edge Classification for Wash Trade Detection on Decentralized Exchanges

Decentralized Finance applications have emerged as one of the critical use cases of smart contract platforms like Ethereum. The complexity of the applications is increasing, and transactions are not only used as a means of value transfer but also to execute smart contract functionality.

Why is the task important? While the hashed representation of executed smart contract function signatures is observable, not all signatures can be translated to a human-readable form because there is a lack of publicly available smart contract source codes. Furthermore, single edges, as well as a collection of edges, may have a higher level of meaning that can range from simple actions such as permission changes to a set of transactions designed to perform arbitrage trades or malicious transactions designed to feign activity where there is none: for example by artificially inflating trading volumes by what is commonly known as wash trading. The latter represents an illegal activity in traditional markets where existing data is difficult to obtain. Stock exchanges usually do not provide information on which accounts have traded with each other. These examples illustrate the need for transaction or edge classification, especially in the Decentralized Finance

setting. In the following, we focus on the wash trade detection example.

Experimental Setup. This task uses the anomalous transaction pattern detection dataset in Chartalist. Decentralized Exchanges (DEX) have grown very popular in the past couple of years and are usually implemented as smart contracts. There are several variants, but those most closely mimicking the centralized exchange model are Limit Order Book (LOB)-based exchanges. Users can place limit orders, directly accept existing orders, and trade one crypto-asset for another. However, as most DEXs do not implement Know Your Customer (KYC) procedures, users can easily wash trade crypto-assets. The Commodity Futures Trading Commission (CFTC) defines it as "entering into or purporting to enter into transactions to give the appearance that purchases and sales have been made, without incurring market risk or changing the trader's market position" [284]. This can be achieved on a blockchain by controlling multiple accounts and trading back and forth between them. The result is fake trading volume, which may draw the attention of unsuspecting investors.

Ground truth on wash trading activity on DEX does not exist. However, we have conservatively found trading patterns conforming to the wash trade definition. To find them, we have extracted all trades from blockchain transaction data of the first popular LOB DEX on Ethereum, namely IDEX and EtherDelta, up until May 4th, 2020. Each trade exchanges one crypto-asset for another, with information on participating accounts, amounts, and timestamps.

We have then constructed a directed, weighted trade graph, where the target node is always the one buying the native asset Ether in exchange for another crypto-asset. We then identify sets of edges that conform to the legal definition of wash trading, which is described in detail in our work [285] and outlined in the following paragraph.

Baseline. The wash trade detection task is approached with a two-step process. First, a candidate set of potential wash trades is determined by repeatedly identifying trade cycles in the form of strongly connected components, which are identified in multiple time windows. In a second step, the candidate set is evaluated on whether each participating account's position (asset balances) has not, or almost not, changed. Performing the second step can be used as an evaluation criterion when a different candidate set generation mechanism is employed. For example, one could devise

a random walk strategy [286] or use graph convolutional networks to identify wash trades. A ML-based approach might learn the same type of patterns that we have manually devised in the original work. To compare with the baseline, the results should additionally be checked for whether they conform to the legal definition of no position change. The primary score is then the number of identified wash trades.

Results. Table 4.3 displays baseline results obtained through the repeated identification of strongly connected components in multiple time windows. We can determine the number of wash trades that meet the legal definition. We can also resolve the total wash trade volume and the number of involved trading accounts. These lower bound numbers can be improved with more advanced methods.

Table 4.3: Wash trades summary for IDEX and EtherDelta.

Metric	IDEX	EtherDelta
# Wash Trades	213,029	69,711
Total Wash Volume (USD)	83,531,254	75,846,518
# Wash Trader Accounts	659	5,533

4.3 Conclusion

In the past ten years, blockchains have gathered a wide variety of data from all walks of life. However, ML and data science methods for analyzing blockchain data tend to be noticeably delayed than the blockchain technology itself. Some domains of blockchain data analytics, such as ransomware payment detection in cryptocurrencies, have emerged as a fundamental societal problem, demanding the development of novel machinery of ML tools. Chartalist makes blockchain data more accessible to the broader ML community and, as such, stimulates the development of innovative ML tools for blockchain data analytics. Chartalist is the first comprehensive platform for blockchain graph datasets with ground truth suitable for a broad range of ML research tasks, from node classification to time-series forecasting.

Chapter 5

GraphPulse: Topological Representations for Temporal Graph Property Prediction

Kiarash Shamsi, Farimah Poursafaei, Shenyang Huang, Bao Tran Gia Ngo,
Baris Coskunuzer, Cuneyt Gurcan Akcora

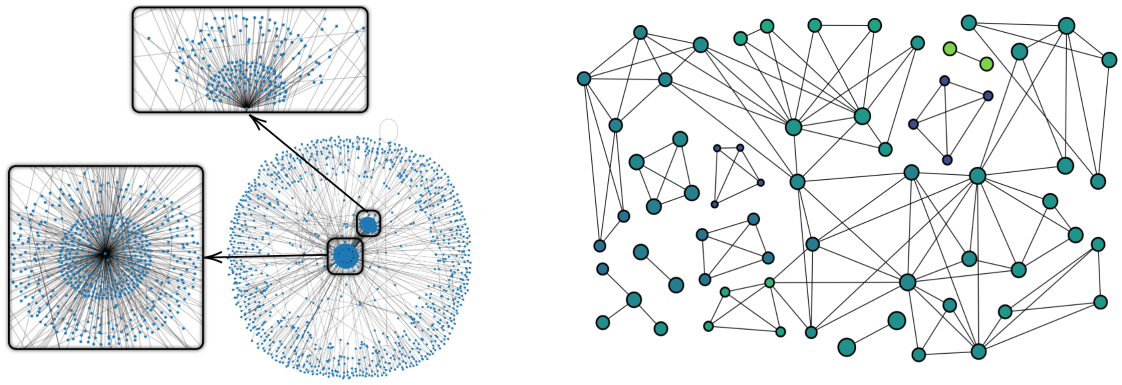
Published in the International Conference on Learning Representations (ICLR 2024)

Real-world interaction networks, such as financial and cryptocurrency networks, experience continuous evolution due to the emergence of new transactions and users. Predicting the dynamic changes in the graph structure of these networks over time poses a significant challenge. While Graph Neural Networks have proven effective for learning graph representations in static graphs [107], temporal graphs differ significantly because nodes and edges continuously change at various timestamps. Classical GNNs are not well-suited for temporal graphs as they do not effectively utilize crucial temporal information, such as alterations in the graph structure over time. Hence, there is a pressing need for a novel approach to adapt GNNs successfully to temporal graph settings, giving rise to a new field known as Temporal Graph Neural Networks (TGNNs).

In order to understand the evolution of a temporal graph, it is essential to capture the global graph structure across time points and then quantify the changes and evolution that the graph has undergone. For instance, in a subsequent time step, the graph might introduce a new node or establish an edge between two existing nodes to significantly shorten the graph diameter. While random graph models such as the Erdos-Renyi graph [287] and the Stochastic Block Models (SBMs) [288] have been studied extensively in the static graph literature [289], there is few theoretical work studying the evolution of temporal graphs in the transition space of graph generative models.

We aim to close this gap and introduce *GraphPulse* to efficiently capture the evolving structure of a graph over time in an innovative temporal framework. Our principled approach posits two hypotheses. First, we suggest that the evolution of a graph can be represented as a temporal trajectory in a Newtonian phase space [290]. Second, we propose that a topological technique, Mapper [31], can be effectively employed to model this trajectory. Mapper generates concise and principled visual representations of data, as exemplified in Figure 5.1. As a third step, we utilize the topological information in a Recurrent Neural Network to predict a graph feature in the future.

We demonstrate the effectiveness of our approach in two types of experiments. First, we empirically prove that Mapper networks can capture temporal trajectories effectively by considering two common phase spaces for graphs: Erdos-Renyi graphs [287] and Stochastic Block Models (SBMs) [288, 289]. Subsequently, we introduce a unique task focused on predicting graph



(a) Daily transaction graph of an Ethereum token (b) TDA Mapper network of the graph in 5.1a.

Figure 5.1: **Illustration of TDA Mapper Network.** The daily transaction graph of an Ethereum token (a) is transformed into a concise Mapper network (b), where nodes represent clusters of token investors, and edges are assigned weights based on the shared number of nodes between clusters. Node sizes indicate cluster sizes and node colors are Disconnected components, highlight node groups with divergent graph characteristics compared to the rest of the nodes.

properties in dynamic graphs and demonstrate that the GraphPulse outperforms existing Graph Neural Network and Temporal Graph Neural Network models in social and cryptocurrency networks. In nine temporal network scenarios within these contexts, our model consistently surpasses state-of-the-art approaches in eight instances.

Our Contributions:

- We present a novel framework, GraphPulse, for analyzing the evolution of temporal graphs, grounded in phase space transition. We empirically show that the TDA Mapper algorithm is highly effective at capturing the evolution of temporal graphs in the phase space.

- We introduce a unique and valuable task involving the prediction of temporal graph properties in the future. GraphPulse takes advantage of sequential modeling to capture temporal variations, and it effectively integrates node feature information into the model using the TDA Mapper algorithm.

- We create seven original cryptoasset networks for the temporal graph property task and publish them to serve as temporal benchmark datasets for future research.

- Empirically, GraphPulse significantly outperforms state-of-the-art temporal graph learning mod-

els in eight out of nine networks from both social and transaction network domains.

- In experiments, we demonstrate that GraphPulse outperforms state-of-the-art temporal graph learning models in the majority of networks from both social and transaction network domains.

5.1 Mapper and Topological Graph Representation

In this section, we describe the first component of GraphPulse, namely the Mapper construction, and its application to a single graph. The formal definition of Mapper, including the notions of lens functions, coverings, and the nerve construction, was presented earlier in Section 2.1.2. Here, we focus on its role in representing graph-structured data within our framework. The core principle underlying Topological Data Analysis revolves around uncovering latent data patterns through systematic analysis of data shapes across multiple resolution scales [21, 57]. TDA provides coordinate-free structural summaries, enabling systematic comparison of patterns derived from diverse data collection frameworks. This property is particularly valuable in temporal graph settings, where graphs from different time periods must be compared in a consistent and geometry-aware manner.

Mapper for graphs. Given a graph $\mathcal{G} = (\mathcal{V}, E)$, we consider the node feature representations $\mathbf{X}_i \in \mathbb{R}^N$ for $v_i \in \mathcal{V}$ and construct the associated point cloud $\mathcal{X}_\mathcal{G} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\} \subset \mathbb{R}^N$. Following the Mapper construction described in Section 2.1.2, we compute the Mapper network $\Gamma_\mathcal{G}$ of this point cloud and use it as a topological summary of the feature geometry encoded in $\mathcal{G}$. The resulting Mapper network captures cluster structure and inter-cluster relationships induced by the chosen lens and covering strategy. The hyperparameters of the Mapper construction, including the lens function, clustering method, resolution, and gain, are discussed in detail in Section 5.4.

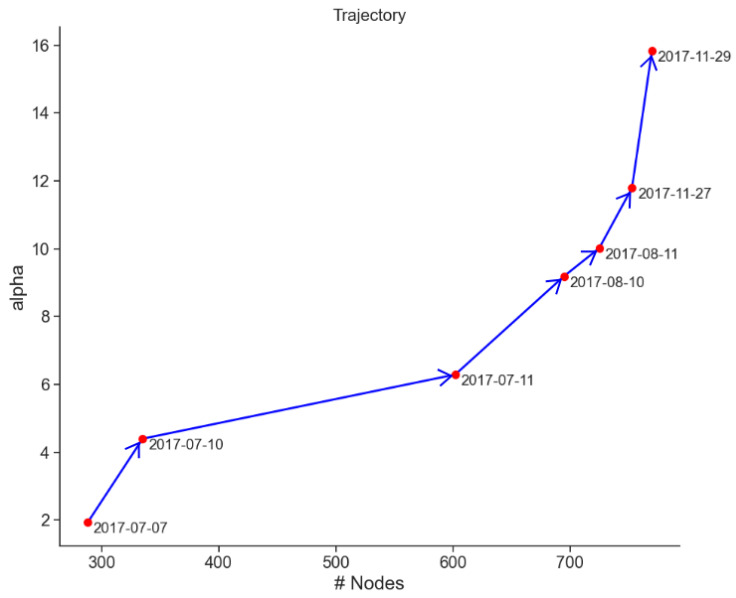


Figure 5.2: **Adex Network Trajectory Example.** A seven day trajectory for the Adex network within the phase space of α and $|V|$. We assume that the network follows a power law model [291], and fit graph data to compute the α exponent of the model. For the network, the power law exponent α moves from 2 to 16.

5.2 Trajectories of Temporal Graphs

In this section we explain the second component of the GraphPulse, where we represent graphs in a space, enabling us to directly compare their Mapper representations. The concepts of phase space and trajectory in this section will serve as the foundation on which we will build a new temporal framework.

We propose a topological model for describing the progression of temporal graphs. Specifically, we posit the presence of a phase space about a family of temporal graphs, wherein the sequence of temporal graphs gives rise to a trajectory (path) within this phase space. Subsequently, for each graph, we induce its topological summary through the Mapper algorithm, which enables us to monitor the evolution of these Mapper networks over time.

5.2.1 Trajectory in a Phase Space

The core idea for our temporal framework is inspired by Newton's laws of motion [290] that specifies how a system's variables change over time in response to forces acting upon it. The idea of *phase space* emerges from Newtonian mechanics as a natural extension of these principles [292]. A phase space $\mathcal{P}$ refers to a multi dimensional space where the state of a system is represented by a set of coordinates corresponding to its variables. In our context, one can imagine this notion as a correspondence map $\varphi : \mathcal{P} \to \mathfrak{S}$ where $\mathfrak{S}$ represents the space of graphs, where any point x in the phase space $\mathcal{P}$ represents a graph $\varphi(x) = \mathcal{G}_x \in \mathfrak{S}$. Hence, **each point in the phase space represents a graph instance**: a specific configuration of the system's variables, capturing its instantaneous or momentary state (see Figure 5.2 for an example). In a dynamical system, the trajectories of a system's evolution over time can be visualized as paths in the phase space. This concept is crucial for analyzing the behavior, stability, and evolution of complex systems by observing how their states change in response to various influences or forces.

Formally, let $\mathfrak{S}_{\mathcal{P}}$ denote a family of graphs parameterized by the parameter space $\mathcal{P}$. To simplify our notation, we will employ Erdős Rényi graphs as an illustrative example, where $\mathcal{P} = \mathbb{N} \times [0, 1]$. In this context, the generative parameters are $(n, p) \in \mathcal{P}$, where n signifies the number of nodes, and p signifies the probability of an edge. This notation can be straightforwardly extended to encompass scenarios involving more than two parameters. Consequently, for $x \in \mathcal{P}$, $\mathcal{G}_x \in \mathfrak{S}_{\mathcal{P}}$ corresponds to the graph within $\mathfrak{S}_{\mathcal{P}}$ associated with that parameter x. We assume a one to one correspondence between the elements of $\mathcal{P}$ and the graphs within $\mathfrak{S}_{\mathcal{P}}$. Consider a dynamic graph family $\{\widehat{\mathcal{G}}^i\}$ whose instances $\mathcal{G}^i_{t_j}$ are in $\mathfrak{S}_{\mathcal{P}}$, i.e., $\widehat{\mathcal{G}}^i = (\mathcal{G}^i_{t_1}, \mathcal{G}^i_{t_2}, \ldots, \mathcal{G}^i_{t_n})$ an ordered sequence of graphs. By using one to one correspondence, we induce a set of trajectories $\{\widehat{\mathbf{x}}^i\}$ with $\widehat{\mathbf{x}}^i = (\mathbf{x}^i_1, \mathbf{x}^i_2, \ldots, \mathbf{x}^i_n)$, i.e., $\mathbf{x}^i_j \in \mathcal{P}$ is the the j^{th} step of the i^{th} trajectory which corresponds to the graph $\mathcal{G}^i_{t_j} \in \mathfrak{S}$.

For any point $\mathbf{x}^i_j \in \mathcal{P}$, we define a vector $\nu^i_j = \mathbf{x}^i_{j+1} - \mathbf{x}^i_j$ where $1 \leq j < n$. The pair $(\mathbf{x}^i_j, \nu^i_j)$ defines a *discrete dynamical system* in the parameter space $\mathcal{P}$ [293]. Our idea is to learn

this induced discrete dynamical system in the parameter space and obtain some useful feature vectors for future instances. In other words, by using the prediction of future trajectory points of $\mathbf{x}^i$ in parameter spaces, we extract useful feature vectors to predict the future instances of dynamic network $\widehat{\mathcal{G}}^i$.

Phase space for token networks. A token network on Ethereum can be conceptualized as a dynamic graph based decentralized ecosystem. Nodes represent participants or entities within the network, including users and smart contracts. Edges symbolize transactions involving tokens.

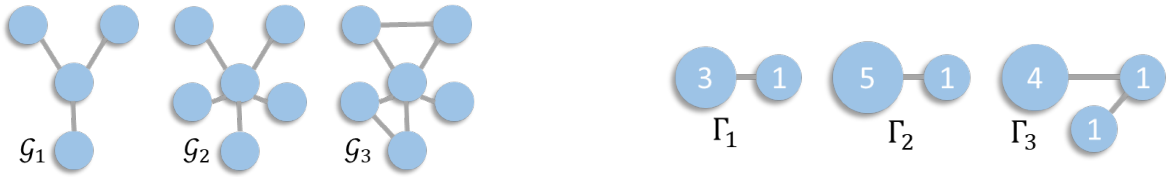
We model token networks as power law graphs [291] where the phase space is given with $|V|$ and the exponent α. In power law graphs, $P(x) \propto x^{-\alpha}$ where x is the node degree. Table 5.1 shows the α values, as described by [294], for token networks. A high α indicates that there are fewer nodes with very high degrees compared to nodes with lower degrees. This leads to a more concentrated distribution, where a small number of nodes have extremely high degrees, and the majority of nodes have relatively low degrees. Different systems exhibit different α values, reflecting the diversity in the distribution of node degrees. For instance, in the World Wide Web, α values typically range from 2 to 3 [291]. As the table shows, the token networks exhibit significantly high α values.

Table 5.1: α values for the power-law degree distributions of seven tokens.

Dataset	$\boldsymbol{\alpha}$
Adex	4.4387
Bancor	3.8886
Aragon	3.3336
Dgd	3.3696
Coindash	4.8846
Iconomi	3.3369
Centra	3.7404

5.2.2 Trajectory in the Topological Space

Consider a temporal trajectory $\widehat{\mathbf{x}} = (\mathbf{x}_0, \ldots, \mathbf{x}_n)$ with $\mathbf{x}_i \in \mathcal{P}$, i.e., $\widehat{\mathbf{x}}$ is an n-step trajectory with the initial point $\mathbf{x}_0$ to the final point $\mathbf{x}_n$ passing through intermediate points $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{n-1}$. While we move on the phase space $\mathcal{P}$ through $\widehat{\mathbf{x}}$, the original graph family defines a corresponding trajectory $\widehat{\mathcal{G}} = (\mathcal{G}_0, \ldots, \mathcal{G}_n)$ in the space of graphs $\mathfrak{S}_{\mathcal{P}}$. Similarly, for the same trajectory $\widehat{\mathbf{x}} \subset \mathcal{P}$, we induce the corresponding TDA Mapper trajectory $\widehat{\Gamma}$ in the space of graphs where each Γ_i is the TDA Mapper network of $\mathcal{G}_i$ by using the induced features (Section 5.1). We give an illustration of our



(a) The graph $\mathcal{G}_1$ has four nodes and three edges. In $\mathcal{G}_2$, the graph gains two more nodes and two edges. However, only in $\mathcal{G}_3$ do two triangular edges emerge in the graph.

(b) Induced TDA Mapper networks Γ_1, Γ_2 and Γ_3 for $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$ by using degree as the node feature. Mapper node sizes (denoted by numbers) indicate cluster sizes.

Figure 5.3: **Evolution of Graph Complexity.** Mapper representations (b) of snapshot graphs (a) where the Mapper lens uses the number of neighbors.

TDA Mapper networks on a toy example using only one node feature (node degree) in Figure 5.3.

Mapper trajectories (theoretical view). Let G be a graph where a monotonically increasing or decreasing function f, such as the number of unique neighbors, is defined over the nodes, indicating changes by the addition of new nodes or edges. Assume that the addition of a new node or edge pair creates a graphlet whose isomorphic copies exist in G. Furthermore, assume that the function f remains unchanged over the nodes of G', a modified graph where node features are retained, and over N', the nodes of the new graphlet, which have features similar to those in the existing nodes N (or the changes are for few nodes only and minimal, as in a star shaped graph with new nodes added at the periphery). If Mapper is used to create a 2D representation that assigns data points to specific cubes, the following holds:

Proposition 1. The addition of N' does not change the position of N within the Mapper network. Specifically, the cluster that contains N can include N' without any modification.

Proof. By construction, Mapper identifies clusters based on similar function values. Since N' and N have similar function values, they fall into the same cluster within the Mapper network. The addition of N' does not affect the existing cluster structure since it shares the same function values as N.

Proposition 2. Consequently, the graphlet's addition to G does not change the similarity of G to G'.

Proof. Since N and N' fall into the same cluster within the Mapper network, this implies that

their topological features are similar in G and G'. Therefore, the addition of the graphlet, which did not change the features of N, does not alter the similarity between G and G'. In other words, $\text{sim}(G, G')$ remains unchanged.

Proposition 3. The graphlet added to G changes $\text{sim}(G, G')$ only if it brings additional information to the graph.

Proof. By definition, $\text{sim}(G, G')$ quantifies the amount of shared information between the two graphs. If the graphlet introduced into G does not bring any new information or topological changes, then $\text{sim}(G, G')$ remains the same. It changes only when the graphlet contributes distinct features or structure to G.

Kolmogorov complexity [295] supports this interpretation. Kolmogorov complexity measures the shortest possible program that can generate a given object. If the graphlet does not modify the structure or features of G, then the minimal description for G and G' remains identical, meaning no additional information has been introduced. Any increase in complexity reflects structural novelty within the evolving graph.

We hypothesize that trajectories can be captured and analyzed efficiently by using Mapper networks of the snapshot graphs. To test our hypothesis, we run extensive experiments in two well known phase space settings, i.e., Erdős Rényi and Barabási Albert. For a given trajectory $\widehat{\mathbf{x}} = (\mathbf{x}_0, \ldots, \mathbf{x}_n)$ in the phase space $\mathcal{P}$, we obtain the corresponding original graph trajectory $\{\mathcal{G}_i\}$ and the corresponding TDA Mapper trajectory $\{\Gamma_i\}$. We compare the structural changes in the original graph trajectory and changes in the TDA Mapper trajectory. In order to measure the structural similarity of graphs, we use a similarity measure induced from graph Laplacian eigenvalues [296]. Specifically, our results in Figure 5.7 show that Mapper based trajectory encodes neighborhood in the phase space more efficiently. Furthermore, as we show in Section 5.4, our experimental findings demonstrate a significant enhancement in the predictive capacity of our model within real world networks due to these topological summaries.

5.2.3 Empirical Evaluation for Mapper Trajectories

To demonstrate the efficiency of TDA Mapper in capturing trajectory (see Figure 5.2) information within the phase space, we implement the following experimental setup. Initially, we establish a reference graph and construct a grid of neighboring points positioned within the phase space around this reference graph (refer to Figure 5.5). Each data point within this grid represents a graph instance generated using the specific parameters corresponding to its location in the phase space.

Formally, we define n^{th}*-neighbourhood* (or n-shell) of a graph $\mathcal{G}_0$ in the phase space $\mathcal{P}$ as the set of all graphs in $\mathcal{P}$ whose distance to $\mathcal{G}_0$ is exactly equal to n with respect to supremum norm. For example, in Figure 5.4, the red dot represents the reference graph $\mathcal{G}_0$, while all light blue graphs represent 1^{st}-neighbourhood of $\mathcal{G}_0$.

For instance, the reference graph can be an Erdős Rényi graph characterized by $p = 0.5$ and $n = 50$. Its phase space neighbors are produced by increasing and decreasing the values of n and p, resulting in a graph with the chosen parameter pair. Consider Figure 5.4 as an example; an immediate neighbor, *neighbor 7*, is generated with $p = 0.5$ and $n = 51$.

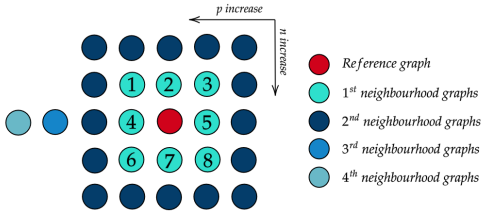


Figure 5.4: **Erdős-Rényi Graph neighbourhoods.** The red dot indicates the reference graph, with its neighbourhood graphs in the phase space. first neighbourhood indexed from 1 to 8.

Within this grid, we establish two key criteria. Firstly, reference graph should exhibit greater similarity to its 1 hop neighbors compared to those beyond 1 hop. Secondly, the 1 hop neighbors should possess similarity values that are nearly identical since they are equidistant from the reference graph.

To this end, we choose *Erdős Rényi graphs* [287] for their widespread usage and *Stochastic Block Models* [288] for their ability to capture community structures within networks. We experimented on a sample of 250 reference graphs generated by the Erdős Rényi graph generator where we varied the number of nodes n from 30 to 50 and p from 0.21 to 0.4 to make the phase space.

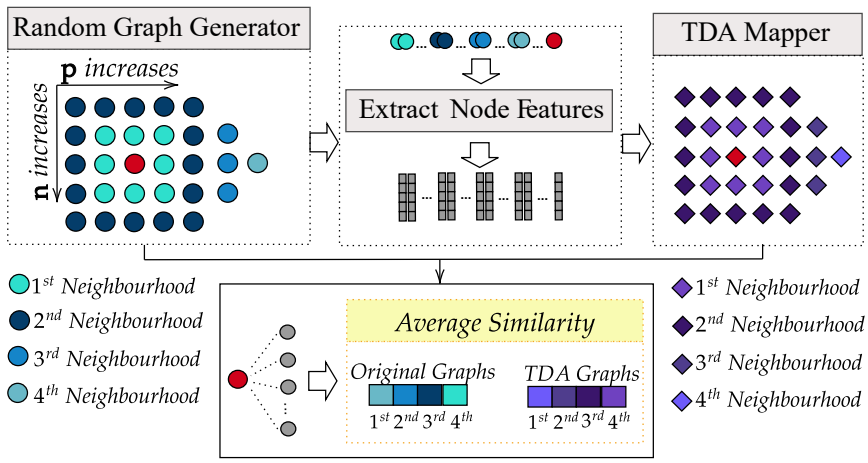


Figure 5.5: **Simulating Graph Similarity**. We generate k graphs using a graph generator model (e.g., Erdős Rényi) with parameters (p and n). Average similarity to neighbors in phase spaces is computed, revealing insights into graph continuity across parameter values.

For the Barabási Albert model, the phase base is defined with the number of nodes n from 120 to 500 and the number of edges to attach from a new node to existing nodes m from 20 to 100. We randomly choose the value for each parameter for each model 50 times and use that combination of parameters to generate reference graphs five times.

Graph Similarity Measure. There are numerous graph similarity measures [297]. In our experiments, we use *the eigenvalue method* for graph similarity measure [296], where graph similarity is defined based on the eigenvalues of the graph Laplacian. We chose this method because it allows us to compare networks of different sizes, which is ideal for temporal graphs where nodes and edges vary across time snapshots. The eigenvalue method involves utilizing the eigenvalues of the graph Laplacian, and it proves to be suitable for our specific scenario.

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the adjacency matrices of graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. Let $\mathcal{L}_1 = \mathcal{D}_1 - \mathcal{A}_1$ and $\mathcal{L}_2 = \mathcal{D}_2 - \mathcal{A}_2$ be their Laplacians, where $\mathcal{D}_1$ and $\mathcal{D}_2$ are the corresponding diagonal degree matrices. Considering that the eigenvalues of $\mathcal{L}_j$ are denoted as $\{\lambda_{j1}, \lambda_{j2}, \ldots\}$, we define the similarity between the graphs as follows:

$$\mathbf{sim}(\mathcal{G}_1, \mathcal{G}_2) = \sum_{i=1}^{k} (\lambda_{1i} - \lambda_{2i})^2$$

where k is the smallest value that satisfies the following condition:

$$\min\{\frac{\sum_{i=1}^{k}\lambda_{1i}}{\sum_{i=1}^{|\mathcal{V}_1|}\lambda_{1i}}, \frac{\sum_{i=1}^{k}\lambda_{2i}}{\sum_{i=1}^{|\mathcal{V}_2|}\lambda_{2i}}\} > 0.9$$

It is understood as the top k eigenvalues containing 90% of the energy [296]. Please note that this method yields unbounded similarity scores within the $[0, \infty]$ range. When the dissimilarity score approaches 0, it indicates a high degree of dissimilarity between the graphs, whereas higher values suggest greater dissimilarity [296].

Similarity Score Comparison. Having selected a similarity function, we now turn our attention to how we compare the Mapper trajectories with those of the snapshot graphs. Figure 5.6 illustrates the main idea in our experiments for comparing the similarity scores of original graphs and their topological counterparts.

First, for any graph $\mathcal{G}$, we use TDA Mapper to induce a summary graph Γ (which is the Mapper network of $\mathcal{G}$) by using its node features. We consider the following features to represent the nodes: PageRank, Degree Centrality, Closeness Centrality, Betweenness Centrality, clustering coefficient, and the number of neighbors. We consider these features because they can be computed from any graph without node or edge labels, which is preferable in our setting where labels might be difficult to collect. For each graph $\mathcal{G}$, the nodes $\mathcal{V}$ are represented as a point cloud $\mathcal{X}_\mathcal{G}$ in the feature space $\mathbb{R}^{N\times d}$. Then, by applying TDA Mapper on the point cloud $\mathcal{X}_\mathcal{G} \subset \mathbb{R}^{N\times d}$, we obtain its Mapper network $\Gamma_\mathcal{G}$.

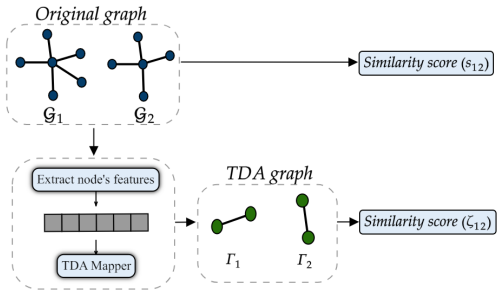


Figure 5.6: **Similarity score comparison.** Using original graphs and extracted TDA Mapper graphs based on original node features for similarity score comparison.

Now, consider two neighboring graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ in the phase space (e.g., Erdős Rényi, Barabási Albert), and let their similarity score be s_{12}, i.e. $s_{12} = \text{sim}(\mathcal{G}_1, \mathcal{G}_2)$. Now, let Γ_1 and Γ_2 be their induced Mapper networks. Let ζ_{12} be their similarity score, i.e., $\zeta_{12} = \text{sim}(\Gamma_1, \Gamma_2)$ By our definition, if ζ_{12} is smaller than s_{12}, then the induced Mapper networks are more similar to the original

Table 5.2: Dissimilarity scores with standard deviations: (a) Barabási–Albert, (b) Erdős–Rényi.

(a) Barabási–Albert

Neighborhood	TDA ($\log_{10}$)	Original ($\log_{10}$)	σ_{TDA}	σ_{original}
1	0.9392	3.6165	0.33	0.20
2	0.9516	4.0612	0.31	0.19
3	0.9756	4.3289	0.32	0.20
4	0.9797	4.5013	0.30	0.20

(b) Erdős–Rényi

Neighborhood	TDA ($\log_{10}$)	Original ($\log_{10}$)	σ_{TDA}	σ_{original}
1	0.9364	2.0726	0.15	0.25
2	0.9430	2.6166	0.11	0.23
3	0.9548	2.9734	0.09	0.22
4	0.9769	3.2091	0.08	0.22

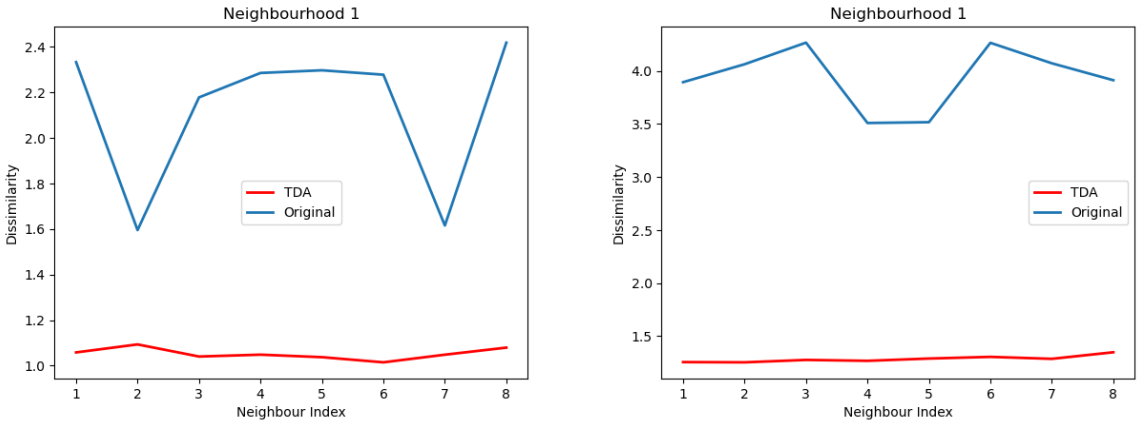
graphs. This interprets that Mapper networks inherit the similarity information better than the original graphs, hence they capture graph trajectory better.

Results. In Table 5.2, we report the median similarity scores for original graphs and TDA Mapper networks in $\log_{10}$ base for both Erdős Rényi and Barabási Albert setting. Due to the unstable frequency of outliers, we use the median to calculate the average similarity score of four neighbourhoods instead of the mean since the median is more robust to outliers than the mean. However, both Mapper networks and original graphs become increasingly dissimilar to the reference graph as the distance (modeled with the k hop neighbourhood) in the phase space increases. Considering this, we see a significant increase in the similarity score in the original graphs while the induced TDA Mapper networks remains highly steady.

Figure 5.7 shows that TDA Mapper yields nearly identical dissimilarity values for the neighbors of the reference graph. The result further offers evidence that TDA Mapper based trajectories would better capture the graph's moves in the phase space.

With the two criteria fulfilled by these empirical results, we conclude that the TDA Mapper can efficiently capture and model graph trajectories in the phase space.

Stability of Mapper networks. We also want to underline the stability of Mapper networks with respect to small changes, and in particular, their robustness against noise. As Figure 5.4 shows every reference graph (red dot) has 8 neighbors (light blue dots) in their first neighbourhood in a 2 parameter phase space (e.g., Erdős Rényi, Barabási Albert). In Figures 5.7a and 5.7b, we consider the first neighbourhoods of the reference graphs, and give the average similarity scores between the reference graphs and these 8 graphs in their first neighbourhood for both Erdős Rényi, Barabási Albert settings.



(a) Erdős Rényi setting.

(b) Barabási Albert setting.

Figure 5.7: **Initial Neighbourhood Comparison.** Comparison of similarity scores in the initial neighbourhood for Erdős Rényi and Barabási Albert configurations. Refer to Figure 5.4 for neighbor indices. Given the proximity of these neighbors to the reference graph in phase space, we expect their dissimilarity to exhibit both low variability and similarity among themselves. Notably, TDA Mapper scores demonstrate greater stability and lower values than the dissimilarity scores of the original graphs.

Changes in the Erdős Rényi and Barabási Albert parameters result in fluctuating the dissimilarity score between the original and reference graphs. Each time a new neighbor graph is created, the number of nodes n and probability p increases or decreases by 2 and 0.05, respectively. These small changes are considered as noise to the graph's trajectory. While there is a fluctuation in dissimilarity scores between the original graph, dissimilarity scores between TDA graphs are stable. Therefore, it is concluded that the TDA method is robust to the noise of graph trajectory.

Table 5.3: Similarity scores in the Erdős–Rényi setting with "rough" Mapper parameters, representing an *imposed failure* scenario.

Neighborhood	**TDA** ($\log_{10}$)	**Original** ($\log_{10}$)	σ_{TDA}	σ_{Original}
1	0.0	2.0743	0.0	0.28
2	0.0	2.5988	0.0	0.25
3	0.0	2.9810	0.0	0.24
4	0.0	3.2105	0.0	0.25

5.2.4 Features for TDA Mapper

To induce our TDA Mapper networks for a given graph $\mathcal{G}$, we use the following node features to induce a point cloud $\mathcal{X}_{\mathcal{G}}$ in the feature space $\mathbb{R}^N$ and follow the method described in Section 5.1.

The node features we use are as follows:

1. **Outgoing Edge Weight Sum:** For each node in the snapshot graph, this feature calculates the sum of the weights of all outgoing edges connected to that node. It provides information about the total influence or importance of the node in sending information to its neighbors.

2. **Incoming Edge Weight Sum:** Similar to the previous feature, this one calculates the sum of the weights of all incoming edges connected to each node in the snapshot graph. It represents the total influence or importance of the node in receiving information from its neighbors.

3. **Outgoing Edge Count:** This feature keeps track of how many edges leave each node in the snapshot graph. The number of other nodes the node is directly connected to as well as its level of connectedness are reflected by this.

4. **Incoming Edge Count:** The number of incoming edges to each node in the snapshot graph is counted by this feature. It gives details on how many nodes are connected to a single node directly.

5.2.5 Advantages of TDA Mapper networks

Our approach introduces a novel perspective to representing and visualizing the evolution of dynamic systems in parameter space. By doing so, we can gain several advantages over using traditional snapshot graphs for trajectory analysis, as follows.

Topological Features. Mapper networks focus on essential topological features, providing a nuanced view of graph evolution under changing parameters. Consider Figure 5.3a that shows three graphs of increasing node counts and connectivity. As we transition from $\mathcal{G}_1$ to $\mathcal{G}_2$, the sole modification involves introducing two nodes while retaining the pattern of satellite nodes linking to a crucial central node. Such a scenario is common on blockchain networks where addresses of blockchain exchanges distribute tokens in airdrops [298]. The Mapper encapsulates that $\mathcal{G}_1$ and $\mathcal{G}_2$ are similar by creating a network of size two with a connecting edge for both Γ_1 and

Γ_2. Retaining such essential aspects in networks becomes particularly significant when addressing complex interactions that might not be evident through simple node attributes (e.g., degree) alone.

Compressed Representation. Mapper networks compress complex data into compact yet meaningful representations, enhancing visualization and robustness against data fluctuations. In Figure 5.3a, we can add another triangle $\triangle$ to $\mathcal{G}_3$ centered on the existing center graph node, but Mapper would just include the two new graph nodes to the bottom Mapper node (i.e., cluster) of Γ_3 without adding a new mapper node.

Multi-Resolution. Mapper's multi-resolution capability enables a detailed exploration of trajectories, revealing fine details and broader trends. As the trajectory moves through parameter space, we can identify clusters (i.e., network nodes) and connections, unveiling significant regions and transitions often hidden in raw snapshot graphs. With clear node and edge features, the resulting visualizations offer valuable insights into dynamic system behavior across parameter trajectories.

We report a comparative analysis of snapshot graphs and their Mapper networks in the Section 5.2.3. In particular, we demonstrate that Mapper networks exhibit greater stability than the original snapshot graphs when subjected to minor perturbations within the phase space. This observation underscores the considerable utility of Mapper summaries in predicting trajectories. We further validate our hypothesis regarding Mapper efficiency through an empirical evaluation of a novel task of predicting graph properties as we discuss in Section 5.4.

5.3 GraphPulse

So far, we have explained how the Mapper-based topological representations can help us track graph trajectories in the phase space. In this section, we describe the overall flow of GraphPulse and detail the third step: sequential modeling.

Our methodology comprises a sequence of three distinct phases, illustrated in the visual representation provided in Figure 5.8: partitioning, topological learning, and sequential modeling. The partitioning step involves the conversion of input data into snapshot graphs denoted as $\widehat{\mathcal{G}} =$

$\{\mathcal{G}_{t_1}, \mathcal{G}_{t_2}, \ldots, \mathcal{G}_{t_n}\}$. In this work, we adopt 24-hour snapshots, although our model remains applicable to snapshots of varying durations. We construct TDA Mapper representations for graphs $\mathcal{CT}_1, \ldots, \mathcal{CT}_n$, each corresponding to a graph within $\{\widehat{\mathcal{G}}\}$. To achieve this, we first extract graph node features $\mathcal{X} \in \mathbb{R}^{N \times d}$ from the snapshot graph $\mathcal{G}_{t_i}$, which serve as inputs for the TDA Mapper (see Section 5.1). Example node features are the count of neighbors and the summation of incoming edge weights, details of which are given in Section 5.4. TDA Mapper creates n Mapper networks for n snapshots, which serve as the input for the sequential modeling step, as we explain next.

In the sequential modeling step, we aggregate pertinent features from both snapshot graphs and Mapper networks such as the number of nodes and edges (see Section 5.2.4 for our selected features). Denoting snapshot graph features as F_{snapshot} and Mapper network features as F_{Mapper}, where F_{snapshot} includes graph-level characteristic features of the snapshot graph such as the number of nodes and edges, and F_{Mapper} encompasses characteristics such as node count and average node size. The fusion of F_{snapshot} and F_{Mapper} provides a holistic perspective on the system's evolution. Equipped with these features, we compose a sequence spanning n days, denoted as S_n, serving as input for our specialized sequential model. This sequential model illuminates the dynamic interplay between the changing graph property and the structural insights extracted from Mapper networks in an organized manner.

The specific nature of the predicted graph property dictates the form of the sequential model. For instance, for real-valued properties, a regression model may be appropriate, while binary properties can be addressed using a classification model. This adaptable approach aligns the model's structure with the unique characteristics of the targeted graph property.

5.4 Experiments

Graph property. We use network growth in terms of edge count as the predicted graph property. Formally, let $\mathcal{G}$ be a graph, t be a specific time, δ_1 and δ_2 be time intervals, and $E(t_1, t_n)$ denote

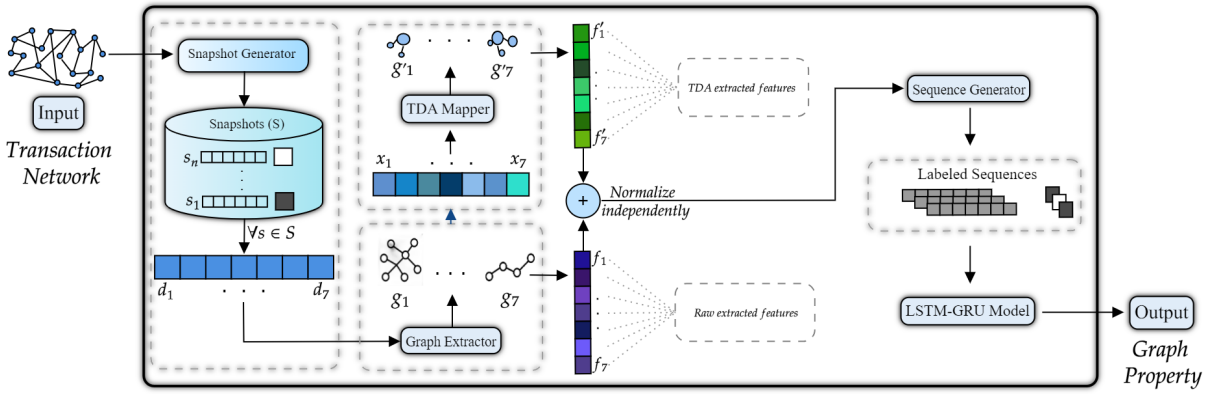


Figure 5.8: **GraphPulse Flowchart**. The GraphPulse system workflow starts with snapshot graph extraction, followed by the generation of Mapper networks. Next, a sequential LSTM+GRU model is developed, incorporating features from both the original snapshot graphs and the Mapper networks.

the multi-set of edges between times t_1 and t_n. We define the graph property P as:

$$P(\mathcal{G}, t_1, t_n, \delta_1, \delta_2) = \begin{cases} 1, & \text{if } |E(t_n + \delta_1, t_n + \delta_2)| > |E(t_1, t_n)| \\ 0, & \text{otherwise.} \end{cases}$$

Setting $n = 7$, $\delta_1 = 1$, and $\delta_2 = 7$, we establish a meaningful and practical graph property. This choice of parameters is valuable due to the application of 7-day predictions, which hold significance in both financial contexts, such as Ethereum asset networks [299], where they can guide financial decisions, and in the realm of social network infrastructure, like Reddit, where they aid in planning maintenance activities.

Datasets.

We perform experiments on MathOverflow [300] and Reddit-Body [301] datasets, and seven ERC20 token networks that we have extracted from the Ethereum blockchain. The ERC20 token networks, consisting of real transaction data from the beginning period of the Ethereum blockchain, provide valuable insights into the dynamics of digital assets. A summary of the statistics of these datasets is presented in Table 5.4.

Mapper aspects.

Section 5.1 constructed Mapper networks based on node features $\mathcal{X}$, which we construct by extracting outgoing edge weight sum, incoming edge weight sum, outgoing edge count, and incoming edge count (see Section 5.2.4 for feature definitions.) We used 2D-TSNE as our lens and KMeans for the clustering algorithm. We set the Mapper hyperparameters as *cls* $= 5$, $n\_cubes = 2$, and *perc_overlap* $= 0.4$ and study their $F_{\textbf{snapshot}}$ **and** $F_{\textbf{Mapper}}$. In the sequential modeling step, we collect specific features from both snapshot graphs and Mapper networks. From the snapshot graphs, we extract three key features: the number of nodes, the number of edges, and the average value of edge weights. Additionally, we leverage Mapper networks to extract five supplementary features: the number of nodes, the number of edges, the maximum cluster size, the average cluster size, and the average value of edge weights.

Table 5.4: Dataset statistics. $\overline{|\mathcal{E}_{\text{daily}}|}$ and $\overline{|\mathcal{V}_{\text{daily}}|}$ denote the average number of daily edges and nodes, respectively. $\overline{|\mathcal{G}|}$ indicates the total number of snapshots. *Reported as *(years,months,days)*.

Dataset	$\overline{\lvert\mathcal{E}_{\textbf{daily}}\rvert}$	$\overline{\lvert\mathcal{V}_{\textbf{daily}}\rvert}$	Duration*	$\lvert\mathcal{G}\rvert$
Adex	259.15	126.41	(0,10,6)	293
Bancor	320.84	154.97	(0,10,24)	311
Aragon	367.99	189.08	(0,11,19)	337
Dgd	90.16	29.32	(2,0,7)	720
Coindash	346.55	128.61	(0,9,11)	268
Iconomi	205.57	85.62	(1,6,12)	542
Centra	294.24	140.85	(0,9,5)	261
Reddit-Body	688.84	86	(1,1,20)	399
Mathoverflow	2532.75	124.09	(0,6,16)	183

Models. We compare GraphPulse against two baselines and three state-of-the-art TGNN models (Section 5.4.1 details node, edge features, and graph types in models):

- The two baseline approaches adopt the powerful GIN framework [107] within a binary graph classification context. We feed a static graph into GIN, encompassing edges from days t to $t+7$, denoted as $\widehat{\mathcal{G}}^i = (\mathcal{G}^i_{t_1} \cup \mathcal{G}^i_{t_2} \cup \cdots \cup \mathcal{G}^i_{t_7})$ to predict the graph property. In a second variant of GIN, we integrate cluster membership information from Mapper networks as additional node features in a model called TDA-GIN. This baseline is designed to gauge the contribution of the information within Mapper networks.

- The state-of-art models in TGNNs are EvolvedGCN [110], GRUGCN [302] and HTGN [112]. EvolvedGCN focuses on evolving graph structures, GRUGCN incorporates variational principles, and HTGN handles discrete time intervals in temporal graphs. The models are notable

Table 5.5: Summary of analyzed models.

Model	Type	**Data**	**Edge Feature**	**Node Feature**
GIN	GNN	$\{\mathcal{G}_{t_1} \cup \mathcal{G}_{t_2} \cup \cdots \cup \mathcal{G}_{t_n}\}$	edge weight	—
TDA-GIN	GNN	$\{\mathcal{G}_{t_1} \cup \mathcal{G}_{t_2} \cup \cdots \cup \mathcal{G}_{t_n}\}$	edge weight	mapper cluster membership
EvolvedGCN	TGNN	$\mathcal{G}_{t_1}, \mathcal{G}_{t_2} \ldots \mathcal{G}_{t_n}$	edge weight	—
GRUGCN	TGNN	$\mathcal{G}_{t_1}, \mathcal{G}_{t_2} \ldots \mathcal{G}_{t_n}$	edge weight	—
HTGN	TGNN	$\mathcal{G}_{t_1}, \mathcal{G}_{t_2} \ldots \mathcal{G}_{t_n}$	edge weight	—
F_{mapper}-RNN	TDA+RNN	$\mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_n$	edge weight	NA
F_{snapshot}-RNN	RNN	$\mathcal{G}_{t_1}, \mathcal{G}_{t_2} \ldots \mathcal{G}_{t_n}$	edge weight	NA
GraphPulse	TDA+RNN	$\mathcal{G}_{t_1}, \mathcal{G}_{t_2} \ldots \mathcal{G}_{t_n}$	edge weight	—

advancements in the field of temporal GNNs.

Implementation details. GIN and TDA-GIN models use a Graph Isomorphism Network with 64 hidden units followed by a target output dimension of two. Raw RNN and TDA RNN models utilize LSTM and GRU layers with an Adam optimizer and a learning rate of 1×10^{-4}. A hybrid LSTM-GRU model processes sequences in a (7,3) and (7,5) format for input, respectively. We evaluate the models using the AUC-ROC score, a suitable metric for prediction assessment. We ran all experiments on a *Dell PowerEdge R630*, featuring an *Intel Xeon E5-2650 v3 Processor (10-cores, 2.30 GHz, 20MB Cache)*, and 192GB of RAM *(DDR4-2133MHz)*.

5.4.1 Baseline Models

In our evaluation of different models, we included baseline models using graph neural networks and recurrent neural networks to tackle our graph-based property prediction task. Here, we will provide explanations for all the baseline models employed in our study. Table 5.5 presents an overview of the model summary information.

GNNs and Our Models

GNNs: For the GNN baselines, we employed the Graph Isomorphism Network (GIN) as a static model due to its remarkable capacity for capturing both local and global graph structures, making it adaptable for tasks such as graph classification, node classification, and link prediction. Also,

we used three state-of-the-art GNN models including EvolvedGCN, GRUGCN, and HTGN as baselines.

GIN. We extract each snapshot graph and augment it with four essential node features. The following features are included: Outgoing Edge Weight Sum, Incoming Edge Weight Sum, Outgoing Edge Count, and Incoming Edge Count.

After incorporating these four node features into the graph representation, Based on the chronological order, the graphs are divided into 80% training and 20% testing data, which are then fed into the model. We employ a GIN model for graph classification. The GIN model consists of four middle layers with 64 hidden units followed by a target output dimension of two, which serves as label prediction. We use Adam optimizer with a learning rate of 0.0001.

TDA-GIN. The TDA-GIN method is a two-step approach that builds upon the previous static GIN method. First, we extract the same graph with the four node features explained earlier for each node. Additionally, we leverage these node features as input for a TDA Mapper algorithm, which forms clusters by grouping similar nodes together. This process yields a new TDA graph, and for each node in this graph, we incorporate the cluster size as a node feature.

We perform a grid search on parameter combinations to determine the ideal combination of TDA Mapper parameters. We choose the most advantageous combination through this optimization process in order to produce the most instructive TDA graph for learning and classification. The TDA-GIN method enhances the representation and classification of temporal graphs by incorporating TDA techniques into the GIN framework, leading to better prediction performance.

While TDA-GIN incorporates additional structural information through the TDA Mapper representation, it has been observed that for temporal tasks, this method may not always yield improved results. The temporal nature of the data brings challenges related to dynamic changes, time dependencies, and evolving patterns, which may not be fully captured by the TDA-GIN approach. As a result, the benefits gained from TDA-based graph representations may not always translate into superior performance for temporal graph property prediction tasks.

RNNs: For our RNN baseline, we developed a hybrid LSTM-GRU model, combining the

strengths of both architectures to address the specific challenges posed by our dataset and task. A hybrid LSTM+GRU model often exhibits superior performance compared to standalone LSTM and GRU models due to its ability to effectively combine the unique advantages of both architectures. LSTM excels at capturing long-range dependencies in sequential data, making it suitable for tasks involving context over extended sequences. On the other hand, GRU is computationally more efficient and can capture short-term dependencies effectively [303]. By blending these two architectures into a hybrid model, we harness the capacity to capture both short and long-term dependencies simultaneously. This enables the model to better understand the complex temporal relationships present in the data, which might be challenging for standalone LSTM or GRU models to grasp individually. Moreover, A hybrid LSTM+GRU model leverages the diverse internal structures and regularization mechanisms of LSTM and GRU models to improve performance and generalization in sequential data tasks, making it a versatile choice. This combination creates an ensemble-like effect, enhancing the model's ability to capture different data features and reduce overfitting risk.

F_{snapshot}-RNN. The F_{snapshot}-RNN method entails building a temporal daily pipeline to capture the temporal component of the data. We extract daily graphs for each day from each snapshot, which contains seven days' worth of data. We extract three graph-level features from each graph, including the number of nodes, the number of edges, and the average value of edge weights. We give this feature a constant value if the graph is unweighted. After these features are extracted, a sequence of three features is created for seven consecutive days, producing a sequence with the shape (7,3) for each snapshot. To carry out the classification task, a hybrid LSTM-GRU model gets these sequences and their corresponding labels. Based on the chronological order, the sequences are divided into 80% training and 20% testing data, which are then fed into the model. Our RNN model consists of two LSTM layers, two GRU layers, and a dense layer for classification. Each snapshot has binary classification labels in the model's output. The AUC-ROC is the evaluation metric employed for this task.

F_{Mapper}-RNN. The F_{Mapper}-RNN method consists of two main steps. In the first step, similar to the

previous model, we construct daily graphs from the temporal data. For each daily graph, we utilize the four features previously discussed in the initial section to generate a TDA graph. This involves extracting the four features for each node in the daily graphs and subsequently constructing a TDA Mapper network for each individual day.

From the TDA daily graphs, we extract five additional features, namely the number of nodes, the number of edges, the maximum cluster size, the average cluster size, and the average value of edge weights with the shape (7,5) for each snapshot. These features are then used to create a sequence representing seven consecutive days. This sequence is fed into the same LSTM-GRU model described before for the classification task. The subsequent steps are identical to the previous model. The sequences are partitioned into 80% training and 20% testing data, based on the chronological order, and are employed as input for the hybrid LSTM-GRU model. The model outputs binary classification labels for each snapshot, and the performance is evaluated using the AUC-ROC metric.

Temporal Graph Representation Learning Methods

Our work builds upon established temporal graph representation learning methods that model evolving structures through time. These approaches serve as baselines in our experiments, and their detailed explanation is discussed in Chapter 2.2.6.

EvolveGCN [110] extends graph convolutional networks with a recurrent mechanism that updates model parameters over time, enabling adaptation to changing graph structures. **GCRN (GRUGCN)** [302] combines convolutional and recurrent architectures to capture spatial and temporal dependencies in structured sequence data. **HTGN** [112] employs hyperbolic geometry to represent hierarchical and evolving relationships within temporal graphs, providing a richer latent space for dynamic structures.

These models were originally developed for node-level tasks such as node classification. Since our focus is on graph-level property prediction, we extend them by adding a pooling layer on top of the encoder to generate holistic graph representations. The resulting pooled embeddings are then

Table 5.6: ROC-AUC results for the graph property prediction task. The **bold** results represent the best methods for each dataset, and the underlined results represent the second-best methods.

Dataset	GIN	TDA-GIN	EvolveGCN	GRUGCN	HTGN	GraphPulse
Adex	0.4484±0.0681	0.6089±0.0574	0.7167±0.1096	0.6843±0.1594	0.7330±0.0849	**0.8928**±0.0022
Bancor	0.5895±0.0514	0.5114±0.0496	0.7931±0.1773	0.8588±0.0190	0.7412±0.0629	**0.8722**±0.0013
Aragon	0.3915±0.0608	0.4648±0.0499	0.7939±0.0875	0.7854± 0.0556	0.7781±0.0508	**0.8926**±0.0035
Dgd	0.5748±0.0163	0.5789±0.0469	0.7460±0.0225	0.6704±0.0557	0.6861±0.0530	**0.7804**±0.0062
Coindash	0.5065±0.0408	0.5015±0.0278	0.7002 ±0.0561	0.7321±0.0399	0.7530±0.0348	**0.7904**±0.0037
Iconomi	0.6079±0.0651	0.5158±0.0651	0.8379±0.0327	0.8105 ±0.0230	0.8221±0.0139	**0.8518**±0.0044
Centra	0.4252±0.0916	0.4777±0.1042	0.8663±0.1302	0.9001±0.0040	**0.9044**±0.0082	0.8670±0.0077
Reddit-Body	0.3563±0.0608	0.5281±0.0777	0.7709±0.0667	0.6591±0.0765	0.6557±0.0401	**0.8692**±0.0070
Mathoverflow	0.3789±0.0867	0.5953±0.1045	0.6734±0.0306	0.7405±0.0938	0.7881±0.0509	**0.8008**±0.0050

passed to a classifier for the downstream task of graph property prediction.

Experimental Details. The experimental parameters are set according to the best practices proposed in baseline methods [112]. For all these methods, we used the in-degree, out-degree, weighted in-degree, and weighted out-degree of the nodes as their initial features. We set the final embedding dimension as 16, and used a *mean*-pooling layer for generating graph-level representations. All methods are composed of one layer of recurrent units and two-layer graph convolutions. For HTGN, the number of historical windows in the HTA module is set to 5. For all methods, we utilized a chronological $\%80$–$\%20$ split of the graph snapshot sequence as our *train-validation* and *test* data, respectively.

5.4.2 Evaluation Results

Table 5.6 shows the ROC-AUC results for the two baselines (GIN, TDA-GIN) and three TGNN models. GIN has > 0.5 AUC only in four of the nine datasets. All results are averages of five runs. Incorporating topological information into GIN as node features, TDA-GIN improves the AUC values in five datasets. The largest gain is noted in the MathOverflow dataset with $+0.216$. However, the increased AUC values of TDA-GIN are still low.

The temporal GNN models have consistently high AUC values with HTGN having the highest mean AUC of 0.770 across datasets. EvolvedGCN's mean AUC is 0.764, while GruGCN follows

closely with a mean AUC of 0.759. GraphPulse has a mean AUC value of 0.849 and has the highest AUC value for eight out of nine datasets.

GraphPulse employs features from both snapshot graphs and Mapper networks within a sequential model. This naturally raises the question: which specific features contribute to the predictive capability of GraphPulse? To address this question, we conduct an ablation study, where, we utilize graph features and Mapper network features in isolation within the same sequential model to predict the targeted graph property. Table 5.7 indicates that the median AUC value for F_{snapshot}-RNN is 0.7981, while for F_{Mapper}-RNN, the value is 0.8501. While Mapper-based features result in an overall higher AUC, it's noteworthy that GraphPulse achieves a considerably higher median AUC of 0.8670, surpassing the AUC of both feature sets in isolation. This observation provides compelling evidence that the topological insights acquired through Mapper offer complementary information to the snapshot graph-based features.

Table 5.7: Ablation study. showcasing ROC-AUC values by incorporating graph features F_{snapshot} and Mapper network features F_{Mapper} within a sequence-based model.

Data	F_{snapshot}-RNN	F_{Mapper}-RNN
Adex	0.8673± 0.0027	**0.8831**± 0.0050
Bancor	0.7981± 0.0078	**0.8501**± 0.0018
Aragon	0.6898± 0.0794	**0.8819**± 0.0014
Dgd	**0.7689**± 0.0090	0.7314± 0.0106
Coindash	0.7676± 0.0025	**0.7790**± 0.0018
Iconomi	0.8404± 0.0204	**0.8417**± 0.0048
Centra	0.8610± 0.0065	**0.8673**± 0.0068
Reddit-Body	**0.8690**± 0.0070	0.7735± 0.0125
Mathoverflow	**0.7798**± 0.0133	0.7522± 0.0057

To assess the importance of individual features within the F_{Mapper}-RNN, we conduct a secondary ablation study within the same sequence model for the Aragon network which has the largest number of nodes and edges per day among our datasets. This study aims to identify key features that significantly influence the F_{Mapper}-RNN model's performance. Table 5.8 indicates that removing the number of edges (shared nodes between Mapper clusters) causes the biggest drop in AUC values for this dataset. Additionally, we assessed the effectiveness of GraphPulse through two supplementary graph property prediction

Table 5.8: Ablation study. ROC-AUC values when removing a Mapper network feature from F_{Mapper} within a sequence-based model for the Aragon network.

Removed feature	ROC-AUC
Number of nodes	0.8716± 0.0087
Number of edges	0.7341± 0.0466
Max cluster size	0.8799± 0.0024
Average cluster size	0.8650± 0.0089
Average edge weight	0.8530± 0.0409

tests: density growth and node count growth. The detailed results for these tests are presented in Section 5.4.3

Scalability. Due to space limitations, we report the computational complexity and scalability results in Section 5.4.5. Here we note that on the largest token network, GraphPulse completes the training process 26% faster than the time required by the state-of-the-art HTGN method. Furthermore, TDA Mapper can process snapshot graphs of 20,000 nodes in under 4 minutes.

5.4.3 Additional Graph Properties

In addition to network growth in edges, we have carried out two experiments to test the predictive power of GraphPulse: predicting the network growth in node count and density.

Table 5.9 displays the results of the node count experiments. Among the nine datasets utilized in the experiment, GraphPulse achieves the highest AUC in five datasets and the second-highest AUC in the Iconomi dataset. EvolveGCN attains the highest AUC value in the four datasets; however, it produces poor results in Adex and Coindash. Among the methods, GraphPulse boasts the highest mean AUC (0.8044). HTGN consistently delivers high AUC results, consequently achieving the second-highest mean AUC at 0.7912.

Table 5.10 illustrates the results for the density property prediction experiments. In this task, GraphPulse outperforms other methods in four of nine datasets and is the second-best method in the Bancor and Aragon datasets. EvolveGCN attains the highest AUC value in three datasets; however, it produces poor results in Coindash and Bancor. HTGN and GRUGCN both achieve the best result in only one dataset. Among the methods, HTGN has the best mean AUC (0.7558). GraphPulse has the second-best mean AUC at (0.7403) however GraphPulse has the best median value (0.7832) and consistently delivers high AUC results for all of the datasets with AUC higher than (0.65)

Table 5.9: ROC-AUC results for the graph node count prediction task.

Dataset	GIN	TDA-GIN	EvolveGCN	GRUGCN	HTGN	GraphPulse
Adex	0.5899±0.0486	0.5002±0.0180	0.5699±0.3690	0.5566±0.3141	0.7720±0.1100	**0.8224**±0.0047
Bancor	0.4724±0.0377	0.5879±0.0627	0.8078±0.1688	0.8165±0.0266	0.6859±0.0781	**0.8182**±0.0115
Aragon	0.5121±0.0918	0.4880±0.0096	0.7020±0.0886	0.6637±0.0380	0.6335±0.0223	**0.7416**±0.0116
Dgd	0.5334±0.0519	0.5400±0.0625	**0.8291**±0.0609	0.7497±0.0629	0.8115±0.0263	0.7851±0.0046
Coindash	0.4582±0.0965	0.4637±0.0497	0.6055±0.1954	0.7900±0.019	0.7969±0.0194	**0.8078**±0.0067
Iconomi	0.6515±0.0200	0.5512±0.0335	**0.9086**±0.0240	0.8505±0.0200	0.8334±0.0261	0.8582±0.0069
Centra	0.5034±0.0994	0.5637±0.0248	0.8221±0.1334	0.8726±0.0035	0.8753±0.0040	**0.8790**±0.0046
Reddit-B	0.4995±0.0329	0.5365±0.0411	**0.9408**±0.0164	0.8147±0.0516	0.8289±0.0424	0.7283±0.0092
Mathoverflow	0.6314±0.0632	0.4559±0.0784	**0.9257**±0.0026	0.8352±0.0485	0.8933±0.0530	0.8404±0.0041

Table 5.10: ROC-AUC results for the graph density prediction task.

Dataset	GIN	TDA-GIN	EvolveGCN	GRUGCN	HTGN	GraphPulse
Adex	0.4763±0.0645	0.4929±0.0206	0.7634±0.3037	0.7234±0.3040	0.7356±0.2419	**0.7938**±0.0017
Bancor	0.5320±0.0367	0.5529±0.0209	0.5810±0.1683	**0.8120**±0.0154	0.7492±0.0753	0.7661±0.0077
Aragon	0.4571±0.1195	0.4896±0.0514	**0.7938**±0.0887	0.6419±0.0151	0.6680±0.0511	0.7832±0.0050
Dgd	0.5228±0.0755	0.5475±0.0581	**0.8711**±0.0292	0.7881±0.0274	0.8171±0.0108	0.7958±0.0088
Coindash	0.4326±0.0646	0.4694±0.0406	0.5021±0.1506	0.7744±0.0051	0.7679±0.0043	**0.7767**±0.0125
Iconomi	0.7209±0.0582	0.5086±0.0588	**0.9146**±0.0082	0.8927±0.0085	0.8784±0.0051	0.8444±0.0105
Centra	0.3991±0.0417	0.5622±0.0746	0.7196±0.1631	0.8434±0.0028	0.8491±0.0017	**0.8908**±0.0050
Reddit-Body	0.4014±0.0569	0.5329±0.0236	0.7072±0.0437	0.6055±0.0633	0.6093±0.0347	**0.7270**±0.0109
Mathoverflow	0.7421±0.0469	0.5133±0.0710	0.6720±0.1883	0.7335±0.0352	**0.7475**±0.0284	0.6846±0.0100

5.4.4 Dependency on Mapper Parameters

The configurability of TDA Mapper is enhanced by its set of hyperparameters. These parameters provide users with the ability to customize TDA Mapper's functionality, allowing for alignment with their specific data and analytical goals. This adaptability facilitates the comprehensive capture and visualization of topological features within diverse datasets. The hyperparameters of TDA Mapper include:

- **Number of Cubes** (n_cubes): Number of hypercubes along each dimension of the projected point cloud using the lens function.
- **Percentage of Overlaps** $(perc\_overlap)$: Percentage of overlap between adjacent cubes calculated along each dimension.

- **Number of Clusters** (cls): Number of clusters in the K-means algorithm that determines the number of inner clusters formed within each cube.

The effectiveness of the GraphPulse can be influenced by the choice of parameters in the Mapper algorithm.

In experiments conducted in 5.2.3, we set the hyperparameters as *cls* $= 5$, $n\_cubes = 2$, and *perc_overlap* $(gain) = 0.4$ to induce our Mapper networks, where *cls* represents how fine the clustering in the point cloud, n_cubes (interval size) represents how big the clusters are, and finally, *perc_overlap* represents how fine the connections between the clusters are. Our experimental results in Figure 5.9 show that Mapper network representations remain stable under small changes in the phase space.

It is noteworthy that the level of view granularity in Mapper is an important parameter. If we increase *perc_overlap* significantly, this will add an edge between most Mapper clusters, and the resulting Mapper network will be a very dense (sometimes complete) graph. We show this dependency with the following experiment in the following Mapper setting: *cls* $= 5$, $n\_cubes = 2$, and *perc_overlap* $= 0.7$. In Table 5.3, we see that the resulting TDA graphs are summarizing too much, and there is no change in the Mapper network even if the original graphs are changing.

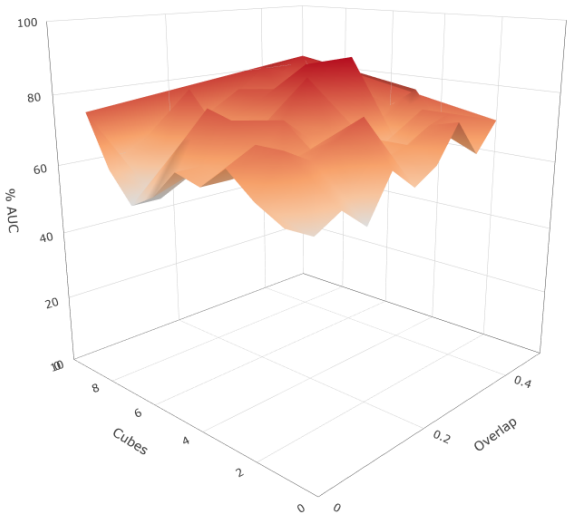


Figure 5.9: **Adex Network Mapper Analysis.** 3d AUC plot over Mapper parameter for the Adex network. Predictive performance (ROC-AUC) is consistently above 0.8 for the region $0.2 < overlap < 0.45$ and $4 < cubes < 8$.

5.4.5 Scalability Analysis

The computational cost of GraphPulse is dominated by the cost of reducing our 4D feature data $\mathcal{X}$ to a $2D$ form to be used by Mapper. We have used tSNE [232] for the reduction which has a

quadratic computational complexity in the number of data points. The cost can be significantly reduced by using tSNE approximations [304]. We leave this improvement to future work. Once a lens is selected, creating the TDA Mapper network involves sorting data points (graphs), which is an $O(n\ logn)$ operation where n is the number of graphs.

We demonstrate GraphPulse's scalability through two key aspects: end-to-end model training costs for the most resource-intensive dataset, and the computational overhead of Mapper analysis for daily snapshot graphs.

Figure 5.10 illustrates the computational time requirements of the considered models on the Dgd network, which boasts the largest number of snapshots (720). Notably, GraphPulse completes training in *1550* seconds, while HTGN demands over *2100* seconds for the same task.

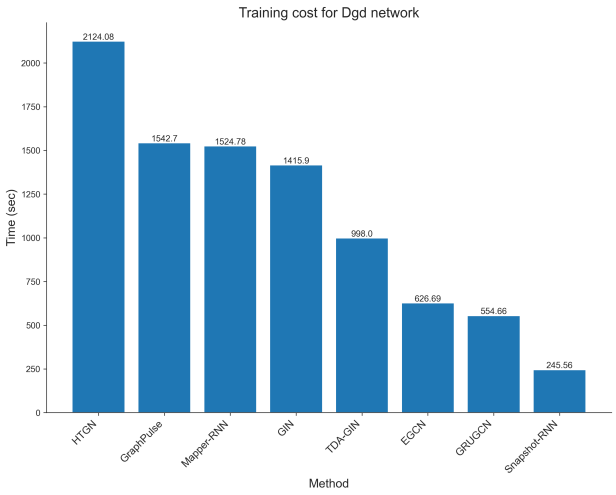
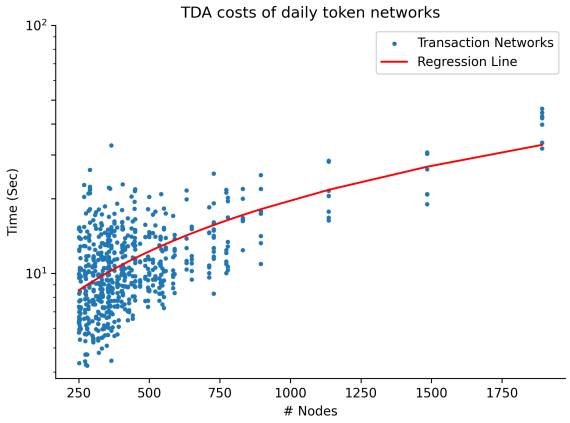


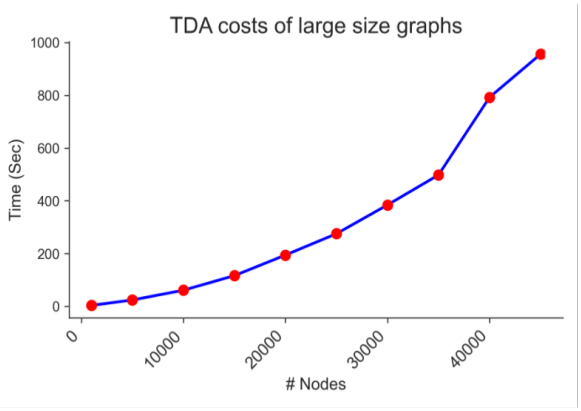
Figure 5.10: **Methods Training Time.** Comparison of Training Time for Methods on the Dgd Network. GraphPulse completes the training process 26% faster than the time required by the state-of-the-art HTGN method.

The processing time of GraphPulse encompasses three distinct stages: the extraction of TDA sequences, the extraction of daily sequences, and the training of the RNN model. Notably, the TDA sequence extraction phase is the most time-consuming component of this process. Consequently, the overall execution time for GraphPulse and F_{Mapper}-RNN is significantly higher when compared to the F_{Snapshot}-RNN model, primarily due to the additional computational requirements of TDA sequence extraction.

Given that the extraction of TDA sequences constitutes a significant portion of GraphPulse's processing time, we conducted a comprehensive analysis focused on the daily TDA cost. Figure 5.11 presents the outcomes of this analysis, shedding light on the question of how substantial the daily cost can be while still permitting efficient TDA processing. Remarkably, the results illustrate that TDA remains highly effective, even with graphs containing approximately 20,000 nodes,



(a) Daily cost of TDA Mapper creation for all token networks. Each data point is a snapshot graph from a token. As shown, most of the daily graphs have less than 750 nodes.



(b) The cost of extracting TDA features for graphs using the Erdős-Rényi graph generation model with $p = 0.3$.

Figure 5.11: **TDA Mapper Scalability.** The daily cost of TDA Mapper for token networks is illustrated in (a). The mapper cost has a direct relation with the size of the graph. (b) shows the increasing cost of TDA Mapper with the growing size of the graph.

processing them in under 4 minutes. Furthermore, the experiments reveal that a majority of the daily networks processed exhibit fewer than 1,000 nodes, ensuring swift processing. This evidence underscores the scalability of TDA for real-world daily graphs within the GraphPulse framework.

These results collectively underscore GraphPulse's outstanding scalability in temporal graph machine learning. By efficiently managing training costs and Mapper analysis, GraphPulse offers a high degree of scalability across various datasets, establishing its suitability for larger and more complex temporal graph scenarios.

5.5 Conclusion

We have introduced GraphPulse, a principled approach for predicting graph properties in temporal graphs. By leveraging a combination of snapshot graphs and Mapper networks, GraphPulse capitalizes on both structural and topological insights to enhance prediction accuracy. Through empirical evaluations, we have demonstrated the effectiveness of GraphPulse across diverse datasets, showcasing its superior performance compared to baseline methods. Notably, our approach demon-

strates scalability across both training and analysis phases, making it an adaptable solution for large-scale temporal graph scenarios. GraphPulse presents a promising advancement in the field of temporal graph property prediction, bridging the gap between structural and topological aspects for accurate predictions.

Chapter 6

MiNT: Multi-Network Transfer Benchmark for Temporal Graph Learning

Kiarash Shamsi, Tran Gia Bao Ngo, Razieh Shirzadkhani, Shenyang Huang, Farimah Poursafaei, Poupak Azad, Reihaneh Rabbany, Baris Coskunuzer, Guillaume Rabusseau, Cuneyt Gurcan Akcora

Published in the Thirty-Ninth Conference on Neural Information Processing Systems (NeurIPS 2025), Datasets and Benchmarks Track

Temporal graph learning has emerged as a vital area of research for modeling dynamic systems where relationships among entities evolve over time. Many real-world phenomena naturally form temporal graphs, including social interactions [305], blockchain transactions [7], biological networks [306], and communication systems [307]. Unlike static graphs, temporal graphs capture time-varying patterns, enabling more accurate forecasting, anomaly detection, and representation learning [5].

The recent success of large pre-trained models in natural language processing [308, 309, 310] and computer vision [168, 311] has spurred interest in developing Graph Foundation Models [312]. These models use a *pre-train and transfer* strategy, where neural networks trained on large datasets can generalize to new tasks with minimal supervision [313, 314]. While this paradigm is well-established in NLP and CV, its application to graph data, especially temporal graphs, is still nascent.

Existing TGL literature typically focuses on training and evaluating models on a single temporal network [14, 315, 128]. These methods learn temporal patterns from a single graph's evolution, implicitly assuming that the temporal dynamics are unique and not generalizable. This practice limits the potential to learn shared temporal structures across different networks and hinders transfer learning, especially in zero-shot scenarios where labeled data is unavailable for new graphs.

In this work, we challenge this limitation and ask two fundamental questions: (1) Can temporal graph models benefit from learning across multiple networks within a single domain? (2) Can these models generalize to previously unseen networks, including those from different domains?

To address these questions, we construct a benchmark of 84 temporal transaction networks derived from the Ethereum blockchain. These networks collectively contain over 3 million nodes and 19 million edges, reflecting real-world financial dynamics over multi-year periods. To study cross-domain generalization, we also incorporate eight temporal social interaction networks from online communities. This benchmark allows for systematic evaluation of scaling behavior, transfer learning, and zero-shot generalization in TGL.

We introduce **Temporal Multi-network Transfer (MiNT)**, the first algorithm to pre-train tem-

poral graph neural networks across multiple dynamic graphs. MiNT alternates between networks during training, resets historical embeddings to ensure independence, and uses network-agnostic validation for model selection. We train MiNT models on up to 64 transaction networks and evaluate them on held-out networks. Our results show that MiNT models outperform single-network models, with performance improving as the number of pre-training networks increases (see Figure 6.1). These findings reveal a neural scaling trend in TGL and highlight the potential for building generalizable temporal graph models.

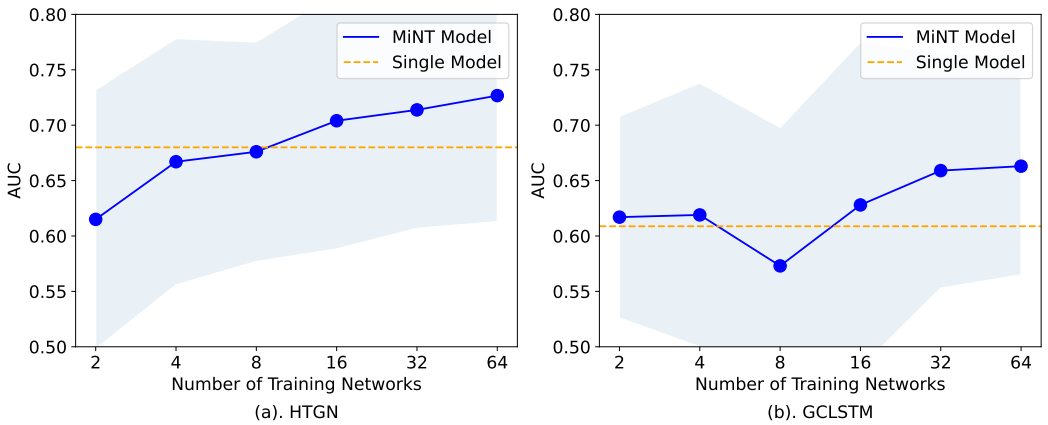


Figure 6.1: **Scaling behavior of MiNT on unseen networks.** Transferred inference performance of MiNT (multi-network model) on unseen networks, compared with standard training of individual networks (single model). The base TGNN models are (a) HTGN and (b) GC-LSTM. The metric is the average ROC AUC over 20 test networks.

Our contributions are:

- **Extensive Temporal Benchmark.** We release the first large-scale transfer learning benchmark of 84 Ethereum-based token networks and 8 existing social networks, enabling research on multi-network pre-training and generalization in TGL.
- **Multi-network Training Algorithm.** We introduce *MiNT-train*, the first algorithm to train TGNNs across multiple temporal graphs, leveraging order shuffling and context switching to ensure robustness and network independence.
- **Neural Scaling in TGL.** We empirically demonstrate that model performance improves with both the number of pre-training networks and the duration of training (in days), revealing a scaling trend analogous to those seen in NLP and CV foundation models for the first time.

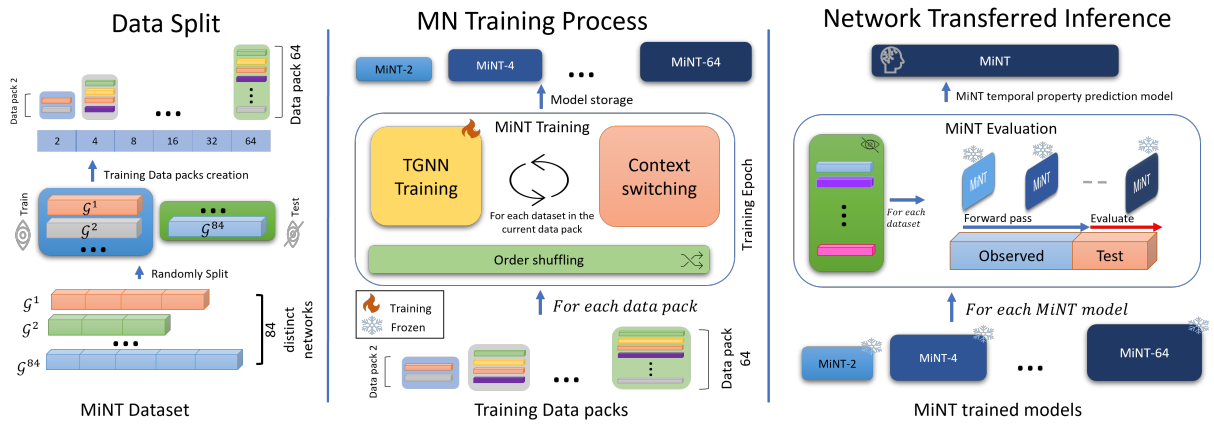


Figure 6.2: **MiNT framework.** Temporal graphs are preprocessed to generate discrete-time snapshots. The multi-network training pipeline leverages these snapshots to train TGNNs across multiple networks for network-transferred inference on unseen temporal networks. MiNT models of varying sizes (for example, MiNT-2, MiNT-4, MiNT-8, up to MiNT-64) each corresponding to a different number of training networks. Each model is trained on a distinct number of networks (e.g., 2, 4, 8, 16, 32, 64) and evaluated in a zero-shot setting on the same set of held-out test networks.

- **Superior Zero-shot Transferability.** MiNT achieves competitive zero-shot performance on 20 unseen token networks, with the best average rank over all the baselines.

6.1 MiNT: Temporal Multi-network Training

Algorithm 2 Multi-Network Transfer for Temporal Graphs

Require: Temporal graph dataset $D = \{\mathcal{G}^1, \ldots, \mathcal{G}^m\}$ where $\mathcal{G}^i = \{\mathcal{G}^i_1, \ldots, \mathcal{G}^i_{n_i}\}$; a **TGNN** model; a graph property **Decoder**
Ensure: Trained TGNN with transferable representations

for each epoch **do**
  Shuffle(D) ▷ Order shuffling across networks
  for each $\mathcal{G}^i \in D$ **do**
    Initialize historical embeddings $\mathcal{H}_0$ ▷ Context switching
    for $t \leftarrow 1$ **to** n_i **do**
      $\mathcal{H}_t \leftarrow \mathbf{TGNN}(\mathcal{G}^i_t, \mathcal{H}_{t-1})$
      $\hat{y}_t \leftarrow \mathbf{Decoder}(\mathcal{H}_t)$
      $\mathcal{L} \leftarrow \mathbf{Loss}(y_t, \hat{y}_t)$
      Backpropagate
    end for
    Evaluate on validation snapshots of $\mathcal{G}^i$
  end for
  Compute average validation performance across datasets
  Save best-performing model
end for

In this section, we introduce our Temporal Multi-network Training (MiNT) algorithm, an innovative multi-network pre-training framework designed to be applied to any backbone TGNN

architecture for DTDG. By leveraging MiNT pre-training, the base TGNN model can transfer to unseen networks for inference. Figure 6.2 provides an overview of our MiNT framework, illustrating the process from dataset curation to the model pre-training stage, and finally *network transferred inference* on test networks. In this work, we evaluate the model on multiple temporal property prediction tasks, including network growth and shrinkage, as well as variations in the size of the largest connected component. These properties, formally defined in section 6.2.1, collectively measure the expansion, contraction, and structural robustness of temporal networks.

6.1.1 Multi-network Training

Existing temporal graph learning models typically train on a single temporal graph, limiting their ability to capture similar behaviors and generalize across different networks [315, 126]. In contrast, we consider a classical learning scenario where a training dataset of m temporal graphs $D = \{\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^m\}$ is drawn identically and independently (IID) from an unknown distribution, and the learned model is evaluated on a test set of unseen temporal networks.

Our MiNT algorithm trains across multiple temporal graphs by modifying a single network training model with two crucial steps: *shuffling* and *context switching*. As explained below, these steps render the algorithm network-agnostic, capable of learning from various temporal graphs to generalize effectively to unseen networks. Algorithm 2 shows the MiNT-train approach in detail. As the first step, we load the list of temporal graphs D, where each temporal graph $\mathcal{G}^i$ is represented as a sequence of snapshot $\{\mathcal{G}^i_1, \mathcal{G}^i_2, \ldots, \mathcal{G}^i_{n_i}\}$. For each epoch, we shuffle the orders of the list of datasets D to preserve the IID assumption of neural network training.

Order Shuffling. Previous works focus on training models on a single network for temporal tasks; instead, we incorporate a shuffling step at each epoch to facilitate training on multiple networks and enable inference on unseen networks. The randomized ordering of networks during training at each epoch is important because it helps prevent the model from learning spurious correlations that could arise if the data were presented in a fixed order. Shuffling the datasets promotes randomness in the training process, contributing to more robust and generalizable model perfor-

mance. Sequentially, we first initialize the historical embeddings, then train the model end-to-end on each dataset $\mathcal{G}^i$ in a similar manner to training a single model, and evaluate the performance on the corresponding validation set of dataset $\mathcal{G}^i$. After training on m datasets from D, we compute the average validation results across these datasets. This average is used to select the best model, which is then used for inference. Early stopping is applied if needed. We verify the importance of order shuffling in the ablation study of Table 6.5.

Context Switching. Many TGNNs store and utilize node embeddings or latent states from previous timestamps at later timestamps; we refer to those embeddings as *historical embeddings* [126, 127, 128]. In Algorithm 2, this is represented in line 6 as

$$\mathcal{H}_t = \mathbf{TGNN}(\mathcal{G}_t^i, \mathcal{H}_{t-1}),$$

indicating that at time steps t, the temporal graph model takes as input both the current snapshot $\mathcal{G}_t^i$ and the *latent state* $\mathcal{H}_{t-1}$ from the previous time step (similar to RNNs). Resetting historical embeddings at the beginning of each epoch (line 4 of Algorithm 2) is a key step in training a temporal model across multiple networks for several reasons. First, it helps prevent the model from carrying over biases or assumptions from one network to another, ensuring that it can adapt effectively to the unique characteristics of each network. Second, it enables the model to learn the most relevant and up-to-date information from the current network, improving performance and generalization across different networks. This is equivalent to resetting the initial vector of recurrent neural networks at the beginning of each sequence.

NTI: Network-Transferred Inference. To evaluate the transferability of each multi-network model, we test the model on test sets that are unseen by the models during the training phase. We first divide our networks into two disjoint sets, where one set is used for training, obtained by randomly selecting 64 token networks, and the remaining 20 token networks are used to evaluate the performance. In the inference phase, we begin by loading all the weights of multi-network models, including the pre-trained encoder and decoder parameters, while initializing fresh historical em-

beddings. Then, we perform a single forward pass over the train and validation split to adapt the historical embeddings specific to the testing dataset.

6.1.2 MiNT Datasets

Numerous graph benchmark datasets have been introduced to advance research within the temporal graph learning community. [205] introduced six dynamic graph datasets while proposing visualization techniques and novel negative edge sampling strategies to facilitate link prediction tasks of dynamic graphs. Following the good practice from OGB [193], [14] introduced TGB, which provides automated and reproducible results with a novel standardized evaluation pipeline for both link and node property prediction tasks. However, these datasets belong to different domains, making them unsuitable for studying the scaling laws of neural network models trained with a large number of datasets from the same domain. [316] provides a temporal benchmark for evaluating graph neural networks in link prediction tasks, though their focus does not extend to training on multiple networks. Conversely, the Live Graph Lab dataset by [317] offers a temporal dataset and benchmark, employed for tasks like temporal node classification using TGNNs. This work aims to explore multi-network training and understand the transferability across temporal graphs. Therefore, we curate a collection of temporal graphs rather than focusing on individual ones as in prior work.

6.1.3 Datasets Extraction

We utilize a dataset of temporal graphs sourced from the Ethereum blockchain [282]. In this section, we will describe Ethereum, explain our data pipeline, and conclude by defining the characteristics of the resulting dataset.

Ethereum and ERC20 Token Networks. We create our transaction network data by first installing an Ethereum node and accessing the P2P network by using the Ethereum client Geth (`https://github.com/ethereum/go-ethereum`). Then, we use Ethereum-ETL (`https://github.com/blockchain-etl/ethereum-etl`) to parse all ERC20 tokens and ex-

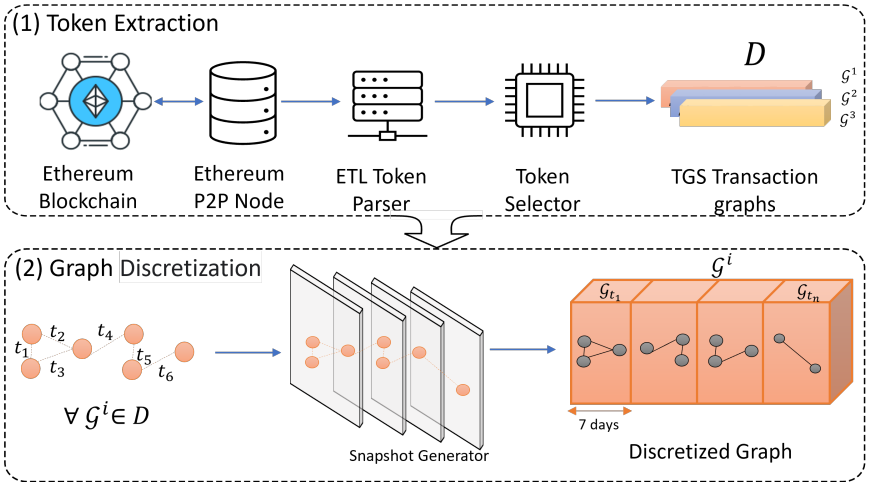


Figure 6.3: **MiNT data processing overview.** (1) *Token extraction*: extracting the token transaction network from the Ethereum node. (2) *Discretization*: creating weekly snapshots to form discrete time dynamic graphs.

tract asset transactions. We extracted more than sixty thousand ERC20 tokens from the entire history of the Ethereum blockchain. However, during the lifespans of most token networks, there are interim periods without any transactions. Additionally, a significant number of tokens live for only a short time span. To avoid training data quality challenges, we use 84 token networks with at least one transaction every day during their lifespan and are large enough to be used as a benchmark dataset for multi-network model training.

Temporal Networks. Each token network represents a distinct temporal graph, reflecting the time-stamped nature of its transactions. In these networks, nodes (addresses), edges (transactions), and edge weights (transaction values) evolve over time, capturing the dynamic behavior of the network. Additionally, these networks differ in their start dates and durations, introducing further variation in their evolution. While each token network operates independently with its own set of investors, they exhibit common patterns and behaviors characteristic of transaction networks. These similarities allow the model to learn and generalize from these patterns across different networks. Collecting temporal graphs from various ERC20 token networks allows for comparative analysis, uncovering common patterns and unique behaviors. This strengthens the model's ability to generalize and improves its robustness.

Figure 6.3 illustrates the MiNT overview from dataset extraction and discretizing graph net-

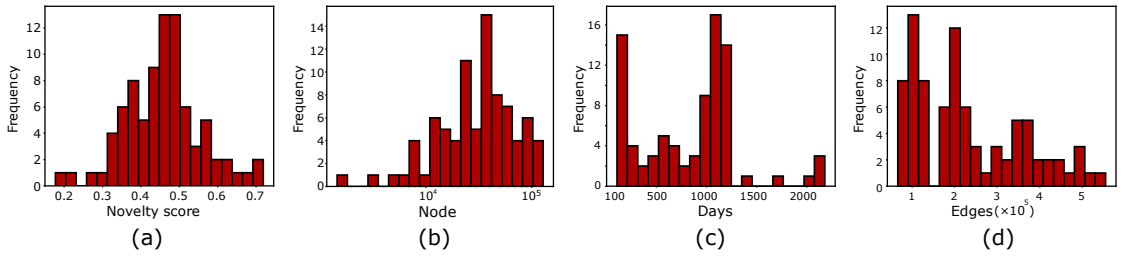


Figure 6.4: **Network statistics of MiNT networks.** (a) Novelty score, (b) number of nodes, (c) number of days, and (d) number of edges.

works for the model training step.

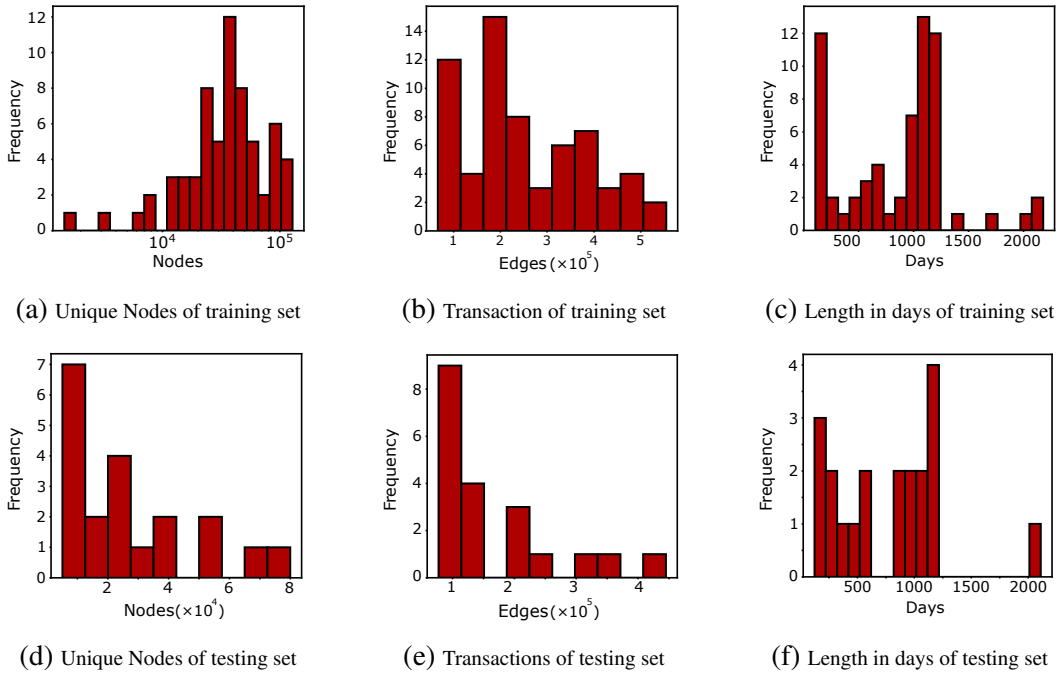
Table 6.1: All token networks' statistics.

Token	#Node	#Transaction	Duration (days)	Growth rate	Novelty	Surprise	Token	#Node	#Transaction	Duration (days)	Growth rate	Novelty	Surprise
ARC	11325	70968	606	0.43	0.32	0.88	Metis	52586	343141	907	0.44	0.48	0.89
CELR	65350	235807	1691	0.49	0.56	0.96	cDAI	52753	358050	1437	0.45	0.46	0.9
CMT	86895	205961	309	0.45	0.72	0.92	BITCOIN	34051	347054	178	0.48	0.39	0.63
DRGN	113453	341849	2164	0.44	0.57	0.97	INJ	60472	312822	1113	0.46	0.52	0.98
GHST	35156	180955	1146	0.43	0.51	0.93	MIM	23038	269366	885	0.44	0.4	0.89
INU	8556	66315	154	0.27	0.41	0.59	GLM	53385	234912	1080	0.5	0.53	0.96
IOTX	63079	288469	1993	0.45	0.56	0.99	Mog	14590	240680	107	0.37	0.38	0.55
QSP	117977	299671	2178	0.45	0.67	0.99	DPI	40627	234246	1150	0.49	0.5	0.86
REP	83282	224843	346	0.46	0.69	0.96	LINA	45342	227147	1144	0.45	0.46	0.95
RFD	23208	173695	169	0.3	0.39	0.6	Yf-DAI	22466	226875	1158	0.42	0.31	0.87
TNT	88247	316352	1216	0.43	0.55	0.93	BOB	42806	212099	199	0.35	0.48	0.73
TRAC	71667	299181	2110	0.46	0.54	0.97	RGT	35277	211932	1110	0.44	0.46	0.98
RLB	28033	240291	129	0.43	0.49	0.76	TVK	42539	208082	1062	0.41	0.48	0.93
steCRV	19079	211538	1033	0.45	0.53	0.9	RSR	50645	205906	659	0.47	0.62	0.91
ALBT	63042	434881	1152	0.43	0.44	0.89	WOJAK	34341	198653	201	0.37	0.48	0.73
POLS	128159	554705	1132	0.45	0.61	0.94	ANT	36517	200262	1107	0.47	0.46	0.93
SWAP	69230	509769	1213	0.46	0.45	0.79	LADYS	37486	192176	181	0.37	0.52	0.79
SUPER	83299	502030	986	0.47	0.46	0.85	ETH2x-FLI	11008	199088	965	0.47	0.28	0.84
RARI	87186	502960	1207	0.43	0.47	0.91	TURBO	38638	189048	189	0.33	0.48	0.72
KP3R	39323	493258	1102	0.43	0.33	0.88	REPv2	39061	191367	1194	0.48	0.5	0.97
MIR	79984	444998	1066	0.45	0.43	0.92	NOIA	29798	185528	1133	0.46	0.37	0.7
aUSDC	23742	475680	1067	0.46	0.4	0.73	0x0	21531	182430	283	0.51	0.46	0.81
LUSD	25852	430473	943	0.48	0.36	0.87	PSYOP	25450	168896	169	0.32	0.39	0.59
PICKLE	28498	430262	1149	0.48	0.34	0.69	ShibDoge	40023	134697	680	0.43	0.53	0.8
DODO	47046	390443	1131	0.47	0.45	0.91	ADX	14567	123755	1188	0.44	0.4	0.91
YFII	43964	391984	1196	0.44	0.44	0.96	BAG	11860	122634	298	0.31	0.44	0.87
STARL	71590	369913	856	0.46	0.48	0.86	QOM	21757	118292	598	0.46	0.41	0.81
LQTY	34687	374230	943	0.45	0.34	0.91	BEPRO	26521	120261	1132	0.46	0.48	0.87
FEG	118294	367584	1007	0.4	0.62	0.92	AIOZ	29231	119926	947	0.43	0.49	0.89
AUDIO	91218	362685	1108	0.45	0.58	0.95	PRE	40476	118625	1113	0.5	0.55	0.86
OHM	45728	377068	690	0.43	0.46	0.88	CRU	19990	117712	1144	0.5	0.43	0.95
WOOL	16874	351178	716	0.41	0.18	0.41	POOH	27245	111641	193	0.26	0.49	0.69
DERC	24277	111205	824	0.45	0.49	0.83	aDAI	13648	187050	1068	0.45	0.46	0.82
stkAAVE	37355	110924	1128	0.42	0.57	0.71	ORN	44010	239451	1134	0.46	0.47	0.87
BTRFLY	8450	108371	453	0.48	0.34	0.44	DOGE2.0	7664	79047	123	0.45	0.38	0.66
SDEX	9127	104869	240	0.41	0.44	0.75	HOICHI	5075	77361	436	0.36	0.32	0.71
XCN	20085	104185	607	0.46	0.42	0.84	EVERMOON	7552	79868	163	0.24	0.35	0.52
HOP	37004	102650	514	0.41	0.6	0.88	MUTE	12426	82345	977	0.43	0.46	0.95
MAHA	18401	96180	749	0.43	0.47	0.91	crvUSD	2950	88647	174	0.61	0.37	0.73
DINO	15837	94140	358	0.44	0.44	0.74	SLP	6675	95368	1151	0.43	0.36	0.91
bendWETH	1454	96898	593	0.51	0.21	0.51	sILV2	12838	92905	611	0.4	0.34	0.48
PUSH	14501	93103	936	0.46	0.38	0.83	SPONGE	25852	90468	184	0.31	0.66	0.81

6.1.4 Dataset Statistics

Our MiNT dataset is a collection of 84 ERC20 token networks derived from Ethereum from 2017 to 2023. Each token network is represented as a dynamic graph, in which each address and transaction

between addresses are a node and a directed edge, respectively. The biggest MiNT token network contains $128,159$ unique addresses and $554,705$ transactions, while the smallest token network has $1,454$ nodes.



(a) Unique Nodes of training set (b) Transaction of training set (c) Length in days of training set

(d) Unique Nodes of testing set (e) Transactions of testing set (f) Length in days of testing set

Figure 6.5: Distribution of the characteristics of the datasets over training and testing sets.

Figure 6.4 shows that most networks have more than 10k nodes and over 100k edges. The lifespan of MiNT networks varies from 107 days to 6 years, and there exists at least one transaction each day. Figure 6.4.a shows the novelty scores, i.e., the average ratio of unseen edges in each timestamp, introduced by [205]. Figure 6.4 shows that most of the 84 networks have novelty scores greater than 0.3, indicating that each day sees a considerable proportion of new edges in these token networks. We adopt a $70\% - 15\% - 15\%$ split of train-test-validation for each token network and calculate the surprise score [205], which indicates the number of edges that appear only in the test data. As Table 6.1 shows, the token networks have quite high surprise values with an average of 0.82. We also provide the node, edge and length distribution for train and test sets separately in Figure 6.5. Overall, train set datasets mostly have more nodes than those in the test set, while the number of edges and days are in the same range for both.

We summarize detailed statistics of each token network in MiNT datasets in Table 6.1. In the

table, the growth rate is the ratio of label 1, indicating the increase in the number of edge counts concerning the problem definition defined in section 6.2.1. In addition, we use the novelty and surprise scores introduced by [205]. The novelty score is defined as the average ratio of new edges in each timestamp. The surprise score is defined as the ratio of edges that only appear in the test set. Formally,

$$novelty = \frac{1}{T}\sum_{t=1}^{T}\frac{|E^t \setminus E^t_{seen}|}{|E^t|}, \tag{6.1a}$$

$$surprise = \frac{|E_{test} \setminus E_{train}|}{|E_{test}|}, \tag{6.1g}$$

where E^t and E^t_{seen} denotes the set of edges present only in timestamp t and seen in previous timestamps, respectively. E_{test} represents edges that appear in the test set, and edges appearing in the train set are represented as E_{train}.

Comparison Between Training And Testing Set. Nodes, transactions, and length (in days) distribution over the training and testing sets are shown in Figure 6.5. Training sets well-support the multi-network model to generalize characteristics of the entire MiNT dataset due to the similarity between nodes, edge and length in days distributions shown in Figures 6.5a, 6.5b, 6.5c and those distributions across 84 token networks of MiNT datasets. In addition, the variance of datasets' characteristics of the testing set is shown in Figures 6.5d, 6.5e and 6.5f.

Table 6.2: Overlapping Nodes Statistics

Dataset	Avg. Common Nodes vs Train Set (MiNT-64)	Train vs Test Node Overlap
MIR	0.021 ± 0.019	0.007
DOGE2.0	0.026 ± 0.033	0.015
MUTE	0.033 ± 0.020	0.045
EVERMOON	0.023 ± 0.033	0.043
DERC	0.020 ± 0.020	0.031
ADX	0.024 ± 0.020	0.018
HOICHI	0.023 ± 0.013	0.053
SDEX	0.024 ± 0.019	0.141
BAG	0.019 ± 0.017	0.107
XCN	0.016 ± 0.010	0.034
ETH2x-FLI	0.038 ± 0.041	0.028
stkAAVE	0.026 ± 0.027	0.057
GLM	0.014 ± 0.015	0.047
QOM	0.018 ± 0.014	0.044
WOJAK	0.025 ± 0.032	0.018
DINO	0.018 ± 0.014	0.049
Metis	0.020 ± 0.013	0.041
REPv2	0.016 ± 0.017	0.013
TRAC	0.015 ± 0.016	0.031
BEPRO	0.023 ± 0.022	0.021

Node Overlap Analysis. We analyze the overlap of nodes between different datasets and within each dataset, which helps demonstrate the highly dynamic nature of our datasets. Specifically, we compared the nodes in each test network with

those in the training networks and calculated the average overlap. As shown in Table 6.2, on average, only 2% of the nodes are common between the training and test datasets, highlighting the rapidly changing structure of these networks.

Furthermore, we analyzed the node overlap within each test dataset by splitting it into the standard train-validation-test setup. We compared the nodes in the 70% training snapshots with the nodes in the final 15% test snapshots, and on average, only 4% of the nodes overlapped. This indicates the highly inductive nature of our model and emphasizes the zero-shot challenge it addresses in this domain. These findings underscore the importance of tackling such dynamic and evolving challenges in temporal graph learning.

6.2 Experiments

We evaluate the transferability of our proposed MiNT approach on unseen test networks in both single-domain (Section 6.2.4) and cross-domain (Section 6.2.5) settings. Our code is available at `https://github.com/benjaminnNgo/ScalingTGNs`, and the datasets are hosted at `https://zenodo.org/records/15364297`. We begin by defining the baseline models and backbone architectures used in our experiments.

We focus on network growth or shrink [17] as the main prediction task. Additionally, we include the growth of the largest connected component to demonstrate the model's capability on a different property prediction task in section 6.2.6. Detailed task definitions are provided in Section 6.2.1. As weekly forecasts are common in the financial context for facilitating financial decisions [299], we set $\delta_1 = 3$ and $\delta_2 = 10$ days for the tasks, thus predicting the temporal graph property over weekly snapshots.

In addition, to show that MiNT is agnostic to the underlying TGNN architecture, we select two widely-used TGNN models as our backbone: HTGN [126] and GCLSTM [127]. We formulate our datasets as discrete-time dynamic graphs, as our prediction tasks are defined over weekly graph snapshots that naturally capture financial and behavioral cycles observed in blockchain net-

works [266]. This formulation preserves the temporal evolution of transaction structures while maintaining consistency with the aggregated nature of the data. While continuous-time dynamic graph models can also be applied to this domain, we limit the scope of this study to the discrete-time formulation, which is highly suitable for our prediction setting and data granularity. Based on this setting, we select HTGN, which has shown state-of-the-art performance on similar task [17], and GC-LSTM, a robust baseline that effectively models temporal dependencies in discrete sequences. These two architectures provide strong and complementary backbones for evaluating the transferability and stability of MiNT. Experimental results suggest that HTGN has superior performance overall; thus, we adopt this model for in-depth analysis and ablation studies. [1]

6.2.1 Task Formulation

We define graph property prediction as the task of forecasting a specific structural property of a temporal graph over a future time interval. In this work, we focus on two binary classification tasks: predicting the growth or shrinkage of (i) transaction volume (i.e., edge count), and (ii) the size of the largest connected component (LCC).

In the network growth prediction task, the goal is to determine whether the number of transactions will increase in the upcoming time window relative to a preceding interval. Given the current weekly snapshot of a network, the model predicts whether transaction activity will rise or decline in the following week. This task is particularly relevant in financial domains, where fluctuations in transaction volume can reflect shifts in user engagement, liquidity, or investor interest. We adopt the same evaluation setup used in GraphPulse [17], and define the property formally as follows:

Definition (Network Growth). Let t_1 and t_n denote the start and end of the observation window, and δ_1, δ_2 define the prediction interval. Let $E(t_{n+\delta_1}, t_{n+\delta_2})$ be the multi-set of edges between times $t_{n+\delta_1}$ and $t_{n+\delta_2}$. The binary property P is defined as:

[1] In what follows, the default MiNT models refer to HTGN with our MiNT pre-training, for example in Figure 6.6.

$$P(\mathcal{G}, t_1, t_n, \delta_1, \delta_2) = \begin{cases} 1, & \text{if } |E(t_{n+\delta_1}, t_{n+\delta_2})| > |E(t_1, t_n)|, \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

Why is this task useful? The network growth/shrink property prediction in financial networks forecasts changes in transaction numbers (edge count), revealing trends in investment activity. A growth in edge count indicates increased investor engagement, while a shrinkage suggests reduced activity or market hesitation. Such investor behavior impacts token prices, and analyzing the behavior helps guide investment strategies, resource allocation, and risk management by providing insights into the evolving dynamics of token networks. For social networks, network growth in time requires resource (e.g., server) allocation and maintaining dynamic load balancing. As a result, forecasting the growth allows for efficient planning.

Definition (LCC Growth). The second prediction task focuses on structural connectivity. Let C_t denote the size of the largest connected component (LCC) at time t. The model predicts whether the LCC will grow over the upcoming interval. Formally:

$$P(\mathcal{G}, t_1, t_n, \delta_1) = \begin{cases} 1, & \text{if } |C(t_{n+\delta_1}, t_{n+\delta_2})| > |C(t_1, t_n)|, \\ 0, & \text{otherwise.} \end{cases} \tag{6.3}$$

Why is this task useful? In Ethereum token networks, the growth of the largest connected component reflects increasing structural integration, where more addresses become part of a unified transaction graph. This is important because token networks typically evolve through isolated pairwise trades, leading to many disconnected components or "islands" of investors. Such fragmentation limits information flow and liquidity, which can undermine price stability and market efficiency. A growing LCC, by contrast, indicates expanding interaction and network cohesion, which are often associated with higher liquidity, stronger network effects, and sustained adoption. Predicting LCC growth helps identify tokens that are moving toward broader market integration.

Setting $n = 7$, $\delta_1 = 3$, and $\delta_2 = 10$ days, we establish a practical graph property with a 7-day prediction window. This choice is particularly relevant in financial contexts, such as Ethereum

asset networks, where it can guide investment decisions, and in social network infrastructure, like Reddit, where it supports maintenance planning.

While this work focuses on specific properties, numerous other characteristics, such as temporal triangle counting that can identify wash trades [318], can also be defined in this domain to highlight the significance of temporal graph property predictions.

6.2.2 Hyperparameters

Single Models. We adopt a $70\% - 15\% - 15\%$ split ratio for the train, validation, and test, respectively, for each token network, and during each epoch, the training model processes all snapshots in chronological order. We train every single model for a minimum of 100 and a maximum of 250 epochs with a learning rate set to 15×10^{-4}. We apply early stopping based on the AUC results on the validation set, with patience and tolerance set to 20 and 5×10^{-2}, respectively. Specifically, in HTGN training, the node embeddings are reset at the end of every epoch. We use Binary Cross-Entropy Loss for performance measurement and Adam [319] as the optimization algorithm. It is important to note that the graph pooling layer, performance measurement, and optimization algorithm are also shared by the multi-network model training setup.

Multi-network Models. While following a similar training approach as in the single model training, we make specific adjustments for the multi-network model training. We set the number of epochs to 300 with a learning rate of 10^{-4} and a train-validation-test chronological split ratio same as single models. Early stopping is applied based on the validation loss with a tolerance of 5×10^{-2} and the patience is set to 30. The best model is selected based on the validation AUC and used to predict the unseen test dataset.

6.2.3 Contenders And Baselines

For comparison, we include heuristics, single-network models, and our pre-trained MiNT models. We refer to models trained on a single network (as in the existing literature) as *single models*. For each temporal graph, we use a 70%-15%-15% split for training, validation, and test sets. During

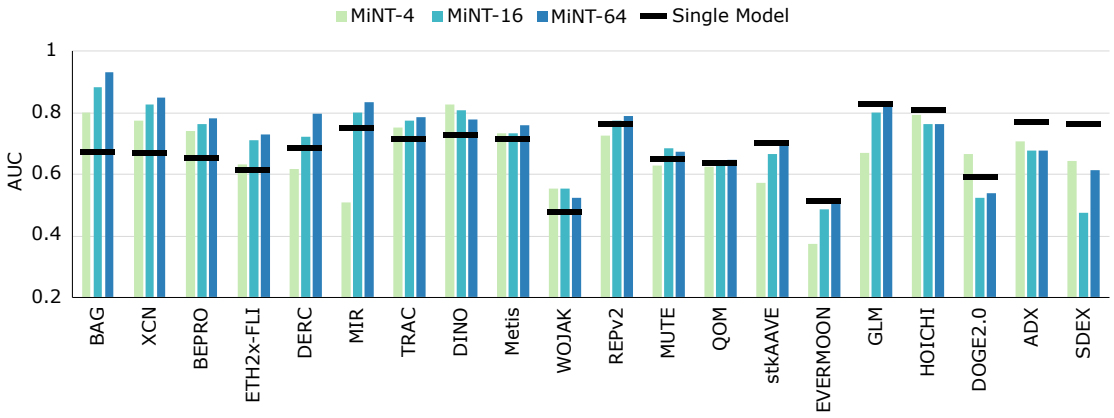


Figure 6.6: **MiNT Performance with Varying Training Scales.** Test AUC of MiNT models trained on 4, 16 and 64 networks and evaluated on unseen test datasets. We compare the performance with HTGN single models trained and tested on each dataset.

each epoch, the model processes snapshots sequentially in chronological order. Train and test networks share approximately 2% of their nodes (see Table 6.2). The results are reported with an average and standard deviation of three runs with different random seeds. We also provide our train/validation/test splits.

We train each model for 100 to 250 epochs with a learning rate of 1.5×10^{-3}, applying early stopping based on validation AUC with patience 20 and tolerance 5×10^{-2}. Binary Cross-Entropy Loss is used for evaluation, and Adam [319] for optimization. The graph pooling layer, loss function, and optimizer are shared in the multi-network training setup. The final pooling layer, implemented as a Multi-Layer Perceptron (MLP), computes the mean of node embeddings, concatenates it with four graph-level snapshot features (mean of in-degree, weighted in-degree, out-degree, and weighted out-degree), and outputs a binary prediction. The choice of mean pooling is further discussed in Section 6.2.10.

Single Models. We use six models from the literature, HTGN [126], GCLSTM [127], EvolveGCN [128], ROLAND [108], GraphPulse [17], WinGNN [129] and a naive heuristic baseline, Persistent Forecasting (P.F.), as our baseline models trained from scratch on individual networks. Persistence forecast uses data from the previous and current weeks to predict the next week's property. If we observe an increase in transactions this week compared to the previous week, we predict a similar increase next week. This simple model assumes that trends in transac-

tion networks persist over time. Our baseline method has three key aspects. First, we do not use any future information to generate the labels. Second, we compare the current week's transaction count to that of the previous week to determine the trend. Finally, if the current week shows an increase, we predict the same trend for the next week. This straightforward approach provides a basic baseline for comparison against more sophisticated predictive models. We provide detailed explanations for each model in section 2.2.6.

MiNT Models. We train six multi-network transfer models, each with a different number of networks corresponding to 2^k datasets, where $k \in [1, 6]$. We name each multi-network model based on the number of datasets used in training; for example, MiNT-16 is trained with 16 datasets. The default TGNN architecture is HTGN, which shows superior performance, while the GCLSTM architecture is also trained and discussed in Table 6.3.

Computational Resource. We ran all experiments on NVIDIA Quadro RTX 8000 (48G memory) with 4 standard CPU nodes (either Milan Zen 3 2.8 GHz and 768GB of memory each or Rome Zen 2, 2.5GHz and 256GB of memory each).

6.2.4 Single Domain Transfer Results

This section presents the network growth task performance of our multi-network models trained on the 64-token datasets and evaluated in zero-shot inference on 20 unseen token test datasets. Similar trends for the second task are reported in section 6.2.6. To evaluate that MiNT's transferability is not limited to the transaction domain, we also trained MiNT on social network datasets, where it demonstrates positive transfer. The detailed results are presented in section 6.2.7. We additionally investigate the (negligible) effect of data selection on the performance of MiNT, with detailed results provided in section 6.2.8.

Table 6.3 compares our results with the five baseline single models that are trained and tested on the same 20 datasets. We also report the top rank, average rank, and win ratio for each model. The top rank indicates the number of datasets where a method ranks first. To calculate the average rank, we assign an AUC-based rank (ranging from 1 to 9) to every model across the 20 test datasets

Table 6.3: **ROC AUC** scores of MiNT transfer models (with HTGN and GC-LSTM backbones) and single models (trained on test networks) on unseen test sets. Best AUC in **bold**, second best underlined.

	Single Model on Individual Networks							Transfer Model - 64 Networks	
Network	HTGN	GC-LSTM	EvolveGCN	GraphPulse	ROLAND	WinGNN	P.F.	GC-LSTM	HTGN
WOJAK	0.479±0.005	0.484±0.000	0.505±0.023	0.467±0.030	0.529 ± 0.005	0.511±0.026	0.378	**0.534±0.020**	0.524±0.027
DOGE2.0	**0.590±0.059**	0.538±0.000	0.551±0.022	0.384±0.180	0.513 ± 0.022	0.577±0.038	0.250	0.551±0.022	0.538±0.038
EVERMOON	0.512±0.023	**0.562±0.179**	0.451±0.046	0.519±0.130	0.349 ± 0.119	0.525±0.114	0.241	0.494±0.047	0.517±0.039
QOM	0.633±0.017	0.612±0.001	0.618±0.002	**0.775±0.011**	0.641 ± 0.003	0.645±0.099	0.334	0.618±0.004	0.647±0.019
SDEX	**0.762±0.034**	0.720±0.002	0.733±0.028	0.436±0.030	0.483 ± 0.254	0.726±0.000	0.423	0.723±0.002	0.614±0.020
ETH2x-FLI	0.610±0.059	0.670±0.009	0.688±0.010	0.666±0.047	0.621 ± 0.023	0.617±0.056	0.355	0.697±0.009	**0.729±0.015**
BEPRO	0.655±0.038	0.632±0.019	0.610±0.012	**0.783±0.003**	0.439 ± 0.125	0.736±0.018	0.393	0.746±0.015	0.782±0.003
XCN	0.668±0.099	0.306±0.092	0.512±0.067	0.821±0.004	0.765 ± 0.015	0.586±0.029	0.592	0.733±0.003	**0.851±0.043**
BAG	0.673±0.227	0.196±0.179	0.329±0.040	**0.934±0.020**	0.418 ± 0.016	0.485±0.105	0.792	0.529±0.023	0.931±0.028
TRAC	0.712±0.071	0.748±0.000	0.748±0.000	0.767±0.001	0.495 ± 0.223	0.752±0.007	0.400	0.742±0.004	**0.785±0.008**
DERC	0.683±0.013	0.703±0.022	0.669±0.009	0.769±0.040	0.405 ± 0.357	0.674±0.044	0.353	0.696±0.011	**0.798±0.027**
Metis	0.715±0.122	0.646±0.023	0.688±0.027	**0.812±0.011**	0.696 ± 0.108	0.690±0.039	0.423	0.697±0.013	0.760±0.025
REPv2	0.760±0.012	0.725±0.014	0.709±0.002	**0.830±0.001**	0.751 ± 0.003	0.744±0.026	0.321	0.733±0.019	0.789±0.020
DINO	0.730±0.195	**0.874±0.028**	0.868±0.029	0.801±0.020	0.497 ± 0.092	0.628±0.251	0.431	0.659±0.039	0.779±0.113
HOICHI	0.807±0.047	**0.857±0.000**	0.856±0.001	0.714±0.010	0.815 ± 0.036	0.769±0.101	0.374	0.847±0.005	0.765±0.018
MUTE	0.649±0.015	0.593±0.030	0.617±0.010	**0.779±0.004**	0.289 ± 0.042	0.593±0.054	0.536	0.636±0.003	0.673±0.013
GLM	0.830±0.029	0.451±0.003	0.501±0.033	0.769±0.018	0.559 ± 0.357	0.530±0.004	0.427	0.501±0.027	**0.831±0.024**
MIR	0.750±0.005	0.768±0.026	0.745±0.015	0.689±0.097	0.228 ± 0.060	0.742±0.015	0.327	0.788±0.022	**0.836±0.016**
stkAAVE	0.702±0.042	0.368±0.011	0.397±0.022	**0.743±0.006**	0.591 ± 0.122	0.572±0.018	0.426	0.650±0.028	0.709±0.022
ADX	0.769±0.018	0.723±0.002	0.718±0.004	**0.784±0.002**	0.761 ± 0.011	0.733±0.023	0.362	0.673±0.022	0.679±0.024
Top rank ↑	2	3	0	**8**	0	0	0	1	6
Avg. rank ↓	4.10	5.35	5.5	3.30	5.85	4.95	8.35	4.35	**2.80**

and compute the average. The win ratio represents the proportion of datasets where a model outperforms a single model. Overall, MiNT-64 exhibits the best overall performance, achieving the state-of-the-art AUC performance in 6 networks and top two performance in 7 out of 20 total test networks with network-transferred inference.

Although the single GraphPulse model achieves the top rank of 8, it is a topology-based method without a GNN and requires supervised training on each dataset. In contrast, our GNN-based MiNT-64 model performs transferred inference efficiently across all datasets in just a few minutes. Despite some trained models like HTGN or GCLSTM excelling on specific datasets (e.g., SDEX and DINO), MiNT-64 consistently ranks competitively across the full benchmark, demonstrating strong generalization without per-dataset training.

For visual clarity, Figure 6.6 shows the AUC on test data results for MiNT-4, MiNT-16 and MiNT-64, as well as the single HTGN model. We show the performance of all six multi-network models in Figure 6.7. Overall, an upward trend is observed in most datasets from multi-network models 2 to 64, such as in *BAG*, *MIR*, and *BEPRO* datasets, highlighting the power of larger multi-

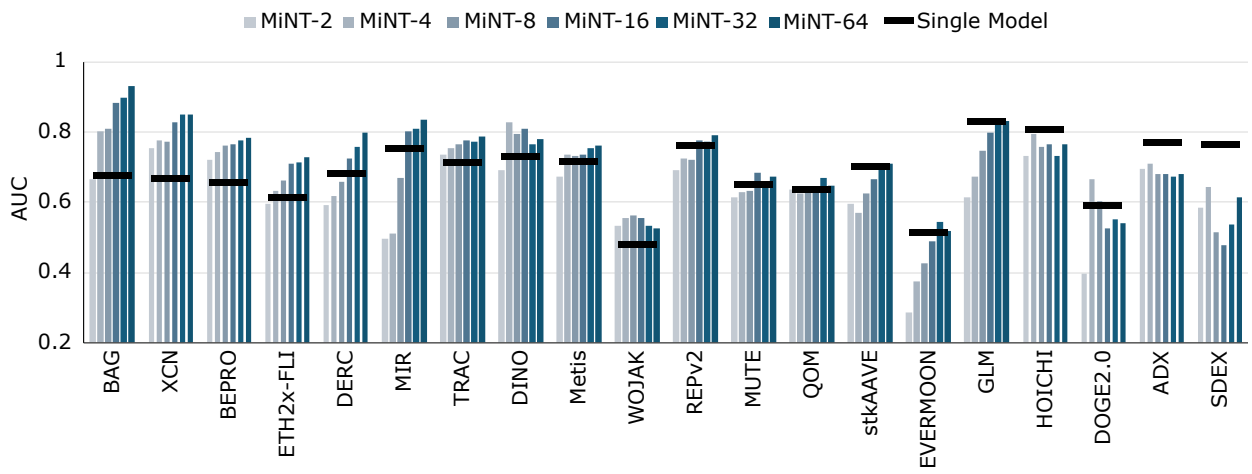


Figure 6.7: Test AUC of multi-network models trained on 2^n datasets where $n \in [1, 6]$ and evaluated on unseen test datasets for network growth or shrink task. Comparing the performance of single models trained and tested on each dataset.

network models in temporal graph learning.

Effect Of Scaling. In Table 6.4, we further compare the models by reporting the top rank, average rank, and win ratio for different configurations of the multi-network models. We observe a notable improvement in performance as the number of training networks increases. For instance, the average rank (lower is better) improves from 6.15 for MiNT-2 to 2.15 for MiNT-64, which signifies a roughly 50% performance enhancement when scaling from two networks to sixty-four. The improvement in the win ratio is also substantial, with MiNT-64 achieving the highest win ratio of 0.80, outperforming the other models in most datasets. This indicates that increasing the number of networks in a multi-network model significantly enhances its robustness and predictive power, particularly when compared to single models and smaller multi-network configurations. Overall, the multi-network-based models have shown superior zero-shot performance and transferability ability. We also conducted a study on the effect of the number of snapshots used for training, with the results presented in section 6.2.9.

Table 6.4: MiNT vs single model performance ranking.

Model	**Top Rank** ↑	**Avg. Rank** ↓	**Win Ratio** ↑
Single Model	3	4.35	–
MiNT-2	0	6.15	0.25
MiNT-4	2	4.35	0.45
MiNT-8	1	4.45	0.45
MiNT-16	1	3.45	0.65
MiNT-32	2	3.20	0.70
MiNT-64	**11**	**2.15**	**0.80**

Ablation Study.

Table 6.5: Ablation study results in AUC.

Model	MiNT-32 ↑	MiNT-64 ↑
Full Model	**0.714± 0.107**	**0.727± 0.114**
w/o Order shuffling	0.708± 0.099	0.694± 0.109
w/o Context Switching	0.677± 0.098	0.688± 0.095

We conducted an ablation study for the MiNT-train algorithm to assess the effects of *context switching* and *order shuffling*. Models are trained in the same way as a multi-network model training setup and tested on the 20 unseen test datasets. The average results are presented in Table 6.5. Training different multi-network models without resetting memory revealed that persistent memory across epochs negatively impacts generalization, emphasizing the importance of reset mechanisms to reduce overfitting. The gain from context switching is considerable when compared to the full model, as it enables stable zero-shot transfer across heterogeneous graphs. This mechanism prevents interference between structurally distinct networks during training and preserves embedding consistency across domains, which is essential for maintaining generalization under large-scale multi-network settings. Additionally, in Section 6.2.8, we explored the necessity of shuffling data by fixing the order of training networks. The observed performance decline indicates that incorporating randomness to MiNT is vital for improving the model's robustness and generalizability.

6.2.5 Cross-domain Transfer Results

To further explore the potential of domain mixing, we train a combined model (*MiNT-12 Mix*) using six token networks and six social networks for the network growth task. The numbers are chosen to be equal to have a balanced mix. This transfer model is evaluated in a fully zero-shot setting across unseen networks from both domains. Results in Table 6.6 show that *MiNT-12 Mix* achieves the best average rank (1.91) and consistently places in the top two across 16 out of 20 datasets and outperforms *MiNT-12* which is trained on 12 transaction networks. On social datasets, such as *RedditB* and *MathOverflow*, *MiNT-12 Mix* performs on par with, or better than, the standard HTGN, despite not being trained on any data from the evaluation set. On transaction networks such as *XCN* and *MIR*, it also outperforms both HTGN and MiNT models trained only on financial

graphs. Table 6.6 shows the AUC results of two Mix models compared to a single model on both social and transaction test networks.

These findings highlight the value of cross-domain pre-training: exposure to diverse structural and temporal patterns enables MiNT to develop representations that generalize effectively across domains. The results support the broader hypothesis that multi-network pre-training can help build more robust temporal graph models.

Table 6.6: **AUC** scores of MiNT, HTGN and MiNT Mix. Scores in **first** and second. † indicates social networks.

Network	Standard Training	Transfer Model	
	HTGN	MiNT-12	MiNT-12 Mix
MIR	0.750±0.005	0.771±0.038	**0.779±0.011**
DOGE2.0	**0.590±0.059**	0.538±0.000	0.538±0.000
MUTE	0.649±0.015	**0.698±0.033**	0.660±0.015
EVERMOON	**0.512±0.023**	0.503±0.037	0.438±0.011
DERC	0.683±0.013	**0.722±0.034**	0.661±0.006
ADX	**0.769±0.018**	0.677±0.014	0.712±0.004
HOICHI	0.807±0.047	0.795±0.041	**0.815±0.012**
SDEX	**0.762±0.034**	0.425±0.173	0.676±0.050
BAG	0.673±0.227	**0.872±0.029**	0.838±0.028
XCN	0.668±0.099	0.761±0.153	**0.837±0.014**
ETH2x-FLI	0.610±0.059	**0.714±0.014**	0.670±0.002
stkAAVE	**0.702±0.042**	0.656±0.010	0.615±0.019
GLM	**0.830±0.029**	0.811±0.011	0.718±0.045
QOM	0.633±0.017	**0.640±0.011**	0.631±0.010
WOJAK	0.479±0.005	0.521±0.024	**0.570±0.033**
DINO	0.730±0.195	**0.856±0.035**	0.846±0.036
Metis	0.715±0.122	0.697±0.036	**0.735±0.024**
REPv2	**0.760±0.012**	0.749±0.017	0.756±0.011
TRAC	0.712±0.071	0.761±0.034	**0.768±0.009**
BEPRO	0.655±0.038	**0.765±0.010**	0.736±0.041
mathoverflow †	**0.788±0.051**	0.575±0.195	0.782±0.004
RedditB †	0.656±0.040	0.653±0.011	**0.695±0.004**
Top One ↑	8	7	7
Top Two ↑	5	8	9
Avg. Rank ↓	2.05	2.00	1.91

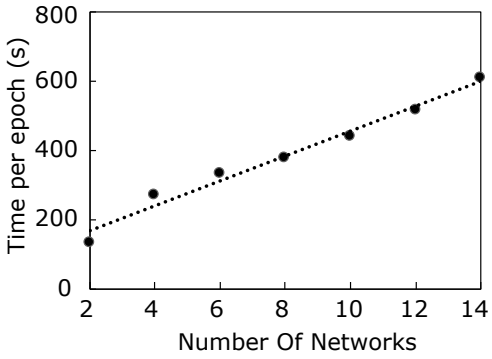


Figure 6.8: Time per epoch for training multi-network models.

Time Complexity Analysis. The MiNT-train algorithm has the same complexity as training the single model, but across all the training networks. Specifically, for the best performing HTGN-based model, the time complexity using MiNT-train is $O(m \cdot (N_{max}dd' + d'|\mathcal{E}_{max}|))$ where m is the number of training networks, N_{max} is set to the maximum number of nodes of networks in the training set, d and d' are the dimensions of the input and output features while $|\mathcal{E}_{max}|$ is the maximum number of edges in a snapshot. Empirically, we observe that the MiNT-training time scales linearly to the number of networks as seen in Figure 6.8, where we report the time per epoch for each multi-network model.

A key strength of MiNT is its ability to perform zero-shot inference: once trained on multiple networks, it can generalize to unseen networks via a single forward pass without retraining.

This makes MiNT highly efficient in real-world scenarios where new temporal graphs frequently emerge. To quantify this efficiency, we compare the time to train a single HTGN model on each test network with the inference time of a pretrained MiNT model, defining an *Efficiency Ratio* as the ratio between the two. As shown in Table 6.10, MiNT achieves over $180\times$ faster inference on average while maintaining competitive or superior performance. These results underscore the scalability and transferability of MiNT.

6.2.6 MiNT on Additional Property Prediction Task

Table 6.7: **AUC** scores of multi-network models and single models on test sets across three seeds on the largest connected component growth task. Best results in **bold**, second best underlined.

Network	Standard Training	Transfer Model				
	HTGN	MiNT-4	MiNT-8	MiNT-16	MiNT-32	MiNT-64
MIR	0.745 ± 0.023	0.570 ± 0.117	0.655 ± 0.012	0.783 ± 0.041	0.766 ± 0.053	$\mathbf{0.845 \pm 0.035}$
DOGE2.0	0.446 ± 0.164	0.530 ± 0.113	0.548 ± 0.063	0.631 ± 0.027	0.571 ± 0.139	$\mathbf{0.661 \pm 0.047}$
MUTE	0.574 ± 0.022	0.471 ± 0.014	0.468 ± 0.021	0.509 ± 0.037	$\mathbf{0.592 \pm 0.038}$	0.582 ± 0.078
EVERMOON	0.494 ± 0.127	0.424 ± 0.059	0.421 ± 0.029	0.376 ± 0.014	$\mathbf{0.542 \pm 0.077}$	0.527 ± 0.118
DERC	0.717 ± 0.035	0.552 ± 0.015	0.584 ± 0.040	0.554 ± 0.011	$\mathbf{0.733 \pm 0.067}$	0.689 ± 0.096
ADX	$\mathbf{0.753 \pm 0.013}$	0.610 ± 0.019	0.635 ± 0.033	0.603 ± 0.019	0.619 ± 0.012	0.587 ± 0.014
HOICHI	$\mathbf{0.746 \pm 0.010}$	0.738 ± 0.009	0.696 ± 0.072	0.715 ± 0.027	0.592 ± 0.147	0.722 ± 0.034
SDEX	$\mathbf{0.911 \pm 0.104}$	0.330 ± 0.117	0.425 ± 0.199	0.361 ± 0.113	0.437 ± 0.316	0.382 ± 0.280
BAG	0.493 ± 0.043	0.772 ± 0.213	0.685 ± 0.163	0.892 ± 0.036	$\mathbf{0.952 \pm 0.019}$	0.893 ± 0.074
XCN	0.566 ± 0.199	0.742 ± 0.039	0.688 ± 0.041	0.802 ± 0.037	0.774 ± 0.144	$\mathbf{0.827 \pm 0.025}$
ETH2x-FLI	0.561 ± 0.037	0.610 ± 0.015	0.625 ± 0.020	$\mathbf{0.658 \pm 0.018}$	0.636 ± 0.076	0.618 ± 0.025
stkAAVE	0.623 ± 0.077	0.613 ± 0.041	0.567 ± 0.038	0.668 ± 0.061	0.687 ± 0.045	$\mathbf{0.688 \pm 0.019}$
GLM	0.761 ± 0.031	0.585 ± 0.144	0.679 ± 0.026	0.698 ± 0.054	0.783 ± 0.031	$\mathbf{0.818 \pm 0.074}$
QOM	0.658 ± 0.150	0.535 ± 0.036	0.513 ± 0.003	0.566 ± 0.036	$\mathbf{0.696 \pm 0.092}$	0.645 ± 0.109
WOJAK	0.378 ± 0.028	0.407 ± 0.012	0.362 ± 0.053	0.384 ± 0.024	0.421 ± 0.029	$\mathbf{0.492 \pm 0.107}$
DINO	0.706 ± 0.120	0.794 ± 0.090	$\mathbf{0.827 \pm 0.039}$	0.815 ± 0.043	0.753 ± 0.165	0.561 ± 0.006
Metis	0.679 ± 0.039	0.697 ± 0.031	0.671 ± 0.047	0.711 ± 0.028	0.705 ± 0.047	$\mathbf{0.780 \pm 0.041}$
REPv2	0.730 ± 0.007	0.653 ± 0.014	0.642 ± 0.061	0.694 ± 0.002	$\mathbf{0.765 \pm 0.030}$	0.742 ± 0.041
TRAC	0.733 ± 0.009	0.658 ± 0.040	0.643 ± 0.052	0.720 ± 0.048	$\mathbf{0.767 \pm 0.012}$	0.762 ± 0.028
BEPRO	$\mathbf{0.694 \pm 0.009}$	0.587 ± 0.002	0.604 ± 0.006	0.589 ± 0.018	0.601 ± 0.129	0.628 ± 0.017
Top Rank ↑	4	0	1	1	6	**7**
Avg. Rank ↓	2.80	4.40	4.65	3.45	2.50	**2.20**

To further demonstrate that the scalability of our approach is not restricted to a specific property, we extended our experiments to evaluate the performance of MiNT models on a new task. This task involves predicting the growth or shrinkage of the largest connected component, which is particularly meaningful in the context of transaction networks.

Experimental Results. Table 6.7 presents the performance of MiNT models and the baseline HTGN across twenty networks. MiNT models, especially MiNT-32 and MiNT-64, outperform the

baseline in the majority of cases. MiNT 64 achieves the highest AUC in seven networks and ranks second in three others. It also records the best average rank overall, indicating strong generalization to this new property prediction task.

These results show that MiNT models are not limited to a particular type of graph signal. Instead, they are capable of adapting to a broad range of temporal properties.

6.2.7 Social Domain Results

To assess the generalizability of our proposed MiNT models beyond transaction-based networks, we conducted experiments on a diverse set of eight real-world social interaction networks. This evaluation aims to demonstrate that MiNT is not constrained to transactional graph domains and can effectively transfer learned representations to structurally and semantically distinct networks.

Table 6.8: **AUC** scores of social multi-network models and single models on test sets across three seeds for network growth or shrink task. Best scores per dataset are shown in **bold**.

Network	**Standard Training**	**Transfer Model**		
	HTGN	MiNT-2 Social	MiNT-4 Social	MiNT-6 Social
mathoverflow	**0.788** $\pm$ **0.050**	0.750 $\pm$ 0.000	0.751 $\pm$ 0.000	0.758 $\pm$ 0.015
RedditB	0.655 $\pm$ 0.040	0.650 $\pm$ 0.002	0.651 $\pm$ 0.004	**0.663** $\pm$ **0.008**

The selected social datasets include *LastFM*[320], *MathOverflow*[300], *SuperUser*[300], *Email-Eu*[300], *AskUbuntu*[300], *CollegeMsg*[321], *StackOverflow*[300], and *RedditB*[301]. These datasets span a wide range of social communication settings, from question-answering platforms to messaging and collaboration networks, providing a rigorous testbed for cross-domain transfer.

We trained three variants of the MiNT model, MiNT-2, MiNT-4 and MiNT-6 on six social networks and evaluated them on two held-out unseen networks: *MathOverflow* and *RedditB*. As shown in Table 6.8, the transferable MiNT models perform competitively with the standard HTGN model that is trained directly on the test network. Notably, MiNT-6 achieves the best performance on *RedditB* (0.663 AUC), surpassing the standard HTGN model, and demonstrates strong results on *MathOverflow* (0.758 AUC), further closing the gap with the single model baseline. We observe a consistent scaling behavior with increasing model capacity (i.e., number of source networks), similar to what was reported in transaction network experiments. This trend indicates that as the number of training networks increases, the MiNT models are better equipped to capture structural

and temporal patterns in unseen networks. This reinforces the model's ability to extract transferable knowledge and leverage broader training contexts effectively.

6.2.8 Effect of Data Selection

We investigate the effect of data selection on the performance of MiNT models trained with different training data packs. As the first work on multi-network training for temporal graphs, we explore the importance of our dataset selection process. To avoid any bias, we randomly sampled the training datasets from the 64 available networks. We conducted an empirical experiment to examine the impact of dataset selection on training MiNT models. In this experiment, we choose three disjoint sets of datasets (data pack A, B, and C) for training MiNT-2, MiNT-4, MiNT-8, and MiNT-16 and two disjoint sets of datasets (data pack A, B) for training MiNT-32. Using disjoint data packs ensures that each model is trained on unique data, eliminating any overlap that could obscure the results. We then test our models on 20 unseen test datasets. Note that MiNT-32 has only two packs, whereas other MiNT models have three packs due to the limited number of available training networks. Specifically, since MiNT-32 requires 32 distinct networks per training run and only 64 total networks are available, we can only create two non-overlapping training sets of size 32. In contrast, smaller models such as MiNT-2 through MiNT-16 allow for more disjoint groupings. As shown in Figure 6.9, as the number of training networks increases, the model performance increases while the variance between different choices of training networks decreases. However, the difference between models that use the same number of datasets diminishes as we move from models of 2 to 32 datasets. We observe that smaller models have a higher variance when compared to larger models; in addition, the model performance also increases from small to large models.

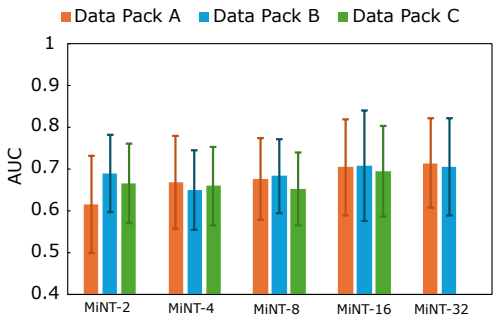


Figure 6.9: Effect of data selection on model performance for network growth or shrink task. When models reach larger sizes (i.e., Mint-32), the effect of data packs is negligibly similar.

6.2.9 Effect of Snapshot Scaling on Model Performance

We conducted an additional experiment to analyze how the number of training snapshots affects model performance over time. Specifically, we evaluated the scaling behavior of the MiNT-64 model by training it on different amounts of historical data. For this study, we selected five snapshot counts: 50, 100, 200, 500, and full snapshot history. These snapshots were drawn sequentially from the end of each dataset, just before the validation period, to simulate a realistic training scenario.

For each configuration, MiNT-64 was trained using three random seeds, and the average AUC results are presented in Figure 6.10. The trend illustrates the scaling behavior of the model as more snapshots are provided. As the training window expands, the model gains access to richer temporal information, which contributes to improved generalization and performance. The trend suggests that access to a larger number of historical snapshots enables temporal models to better capture evolving patterns and improve predictive performance.

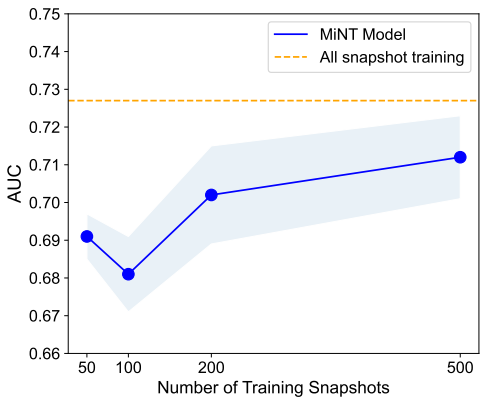


Figure 6.10: Scaling effect of number of snapshots used in MiNT-64 training.

Table 6.9: Test AUC on unseen networks between MiNT-4 with max pooling and MiNT-4 with mean pooling.

Dataset	MiNT-4 Test AUC Max Pooling	MiNT-4 Test AUC Mean Pooling
MIR	**0.588**	0.510
DOGE2.0	0.500	**0.667**
MUTE	**0.685**	0.627
EVERMOON	**0.622**	0.373
DERC	**0.761**	0.617
ADX	0.692	**0.708**
HOICHI	0.583	**0.795**
SDEX	0.401	**0.643**
BAG	0.610	**0.802**
XCN	0.557	**0.774**
ETH2x-FLI	**0.704**	0.632
stkAAVE	0.525	**0.571**
GLM	0.598	**0.671**
QOM	0.611	**0.624**
WOJAK	**0.611**	0.556
DINO	0.480	**0.827**
Metis	0.558	**0.734**
REPv2	**0.746**	0.725
TRAC	0.572	**0.752**
BEPRO	**0.788**	0.742
Average	0.610	**0.668**

6.2.10 Choice of Graph Pooling

Pooling plays an important role in temporal graph property prediction. In our study, we employ mean pooling due to its stability across diverse datasets. To assess alternative choices, we compared mean pooling with max pooling and found that the latter leads to an average 6% drop in AUC, Table 6.9, highlighting the effectiveness of mean pooling in our zero-shot temporal graph setting. Adaptive or hierarchical pooling mechanisms may better capture structural dependencies, and we consider this an interesting direction for future work.

6.2.11 Zero-Shot Inference Efficiency of MiNT

As shown in Table 6.10, MiNT demonstrates remarkable computational efficiency across all unseen datasets compared to training a single HTGN model on each individual unseen network. On average, HTGN requires 2141.66 seconds to train a model per dataset, whereas MiNT completes inference in only 11.52 seconds, yielding an impressive average efficiency ratio of $180.86\times$. This result highlights the clear advantage of MiNT 's zero-shot inference capability. Once pretrained on multiple networks, it can generalize to unseen temporal graphs without the need for retraining, thus saving substantial computational resources.

Table 6.10: Comparison of time efficiency on unseen datasets (in seconds): HTGN vs. MiNT.

Dataset	HTGN Single Model Train Time	MiNT Inference Time	Efficiency Ratio
EVERMOON	196.18	2.03	96.64
DOGE2.0	392.16	1.47	266.78
SDEX	516.29	3.21	160.84
BAG	504.99	4.06	124.38
DINO	451.06	4.91	91.87
WOJAK	665.12	2.67	249.11
XCN	755.11	8.33	90.65
HOICHI	1262.99	5.72	220.80
Metis	1359.59	12.93	105.15
QOM	1681.95	8.34	201.67
MUTE	1679.20	13.74	122.21
GLM	1896.96	15.35	123.58
ETH2x-FLI	2931.61	13.47	217.64
REPv2	3275.41	16.70	196.13
DERC	3486.62	11.49	303.45
BEPRO	3789.14	15.74	240.73
stkAAVE	3194.12	15.81	202.03
ADX	3673.39	16.57	221.69
MIR	4525.21	28.04	161.38
TRAC	6596.15	29.93	220.39
Average	**2141.66**	**11.52**	**180.86**

A closer look at the dataset-level results reveals consistent and significant efficiency gains across all cases. Particularly, datasets such as DERC (303.45×), DOGE2.0 (266.78×), WOJAK (249.11×), BEPRO (240.73×), HOICHI (220.80×), ADX (221.69×), and TRAC (220.39×) exhibit

extremely large efficiency ratios, with MiNT inference being more than two hundred times faster than training a new HTGN model. These datasets tend to have relatively complex temporal dynamics or larger network sizes, implying that MiNT's pretraining enables it to generalize efficiently without the expensive retraining process required by HTGN.

Even for datasets where the efficiency ratio is relatively smaller, such as DINO (91.87×), XCN (90.65×), and GLM (123.58×), the improvement still amounts to nearly two orders of magnitude, representing a dramatic reduction in computational cost. This consistency across diverse datasets underscores MiNT 's scalability and robustness.

Overall, these findings emphasize that MiNT not only provides dramatic time savings but also scales effectively across both small and large networks, maintaining reliable inference speed without sacrificing model performance. The ability to perform inference hundreds of times faster makes MiNT particularly advantageous in dynamic, real-world scenarios, such as financial transaction networks, communication systems, and social platforms, where new temporal graphs continuously emerge and require immediate adaptation. Consequently, this efficiency establishes MiNT as a highly practical and deployable framework for advancing the development of temporal graph foundation models.

6.3 Conclusion

This work addresses a central question in temporal graph learning: can models trained on collections of temporal networks generalize to predict the evolution of previously unseen networks, both within and across domains? Our findings show that such generalization is not only feasible but effective.

We have introduced a benchmark of 84 Ethereum-based temporal graphs designed to support research on graph-level forecasting, neural scaling, and the development of foundation models for temporal graphs. To enable learning across diverse networks, we have proposed MiNT-train, the first framework for training temporal graph neural networks on multiple independent dynamic

graphs.

Empirically, we have observed clear neural scaling behavior: model performance improves as the number of training networks and the number of snapshots increase. Additionally, MiNT-trained models achieve the highest average rank over single models on both within-domain and cross-domain test graphs. These results highlight the promise of multi-network training as a foundation for building generalizable temporal graph models and advancing the field toward temporal graph foundation models.

Chapter 7

Hydra: Towards Transferable Multi-Task Learning on Temporal Graphs

Kiarash Shamsi Farimah Poursafaei, Tran Gia Bao Ngo, Reihaneh Rabbany, Baris Coskunuzer, Guillaume Rabusseau, Shenyang Huang, Cuneyt Gurcan Akcora

Submitted to the International Conference on Learning Representations (ICLR 2026)

Temporal graph prediction addresses the challenge of forecasting the future state and dynamics of networks that evolve over time and possess an underlying relational structure. Unlike traditional graph analysis that assumes a static topology, temporal graphs are characterized by continuously changing nodes, edges, and features, representing dynamic real-world phenomena such as social network interactions [322], traffic flow [323], and biological processes [324]. The core difficulty lies in capturing both the complex spatial dependencies encoded in the graph structure and the temporal dynamics that governs its evolution [325]. Effective solutions must therefore move beyond static graph representation learning to models that can jointly capture spatio-temporal patterns, enabling accurate prediction of future interactions, node properties, and system-wide events. This capability underpins applications such as disease outbreak detection [326], fraud prevention [327], recommendation systems [328], and transportation networks [329]. Despite progress in temporal graph learning, two fundamental challenges remain open.

Multi-task Temporal Graph Prediction. The first challenge is multi-task temporal graph prediction. Most existing methods are designed for a single task, such as link prediction or node classification, whereas many real applications require forecasting multiple interdependent graph-level properties at once. For example, a temporal model may need to predict both community evolution and connectivity growth simultaneously. By learning such tasks jointly, models can exploit shared spatio-temporal patterns that underlie these phenomena, achieving stronger generalization and a deeper understanding of system-wide dynamics. Multi-task learning has long been recognized as an effective strategy for improving generalization by optimizing several related objectives together and leveraging shared structural information [330, 331, 332]. In graph learning, numerous approaches have explored multi-task strategies to enhance model robustness and transferability, demonstrating strong performance across diverse datasets and downstream tasks [333, 334, 184, 335]. However, while these methods have produced major advances for static graphs, their extension to temporal settings remains limited. Temporal graphs introduce evolving node and edge dynamics, where dependencies change over time and task relationships vary across snapshots. Addressing multi-task learning in this setting requires architectures that can cap-

ture global graph behavior over time while dynamically balancing multiple, and often competing, predictive objectives.

Transferability Across Networks. The second challenge is transferability across networks. In real-world dynamic systems, new networks frequently emerge, data can be sparse or incomplete, and retraining models from scratch for every case is both inefficient and impractical. A practical solution is to pre-train temporal graph models on multiple observed networks and transfer them to unseen ones without retraining [336, 337, 338]. Achieving this requires learning representations that capture generalizable temporal patterns while scaling effectively with data size and model capacity.Recent advances in temporal graph learning have produced models specialized for tasks such as link prediction [114, 119, 6], achieving strong results on individual datasets. However, these models typically lack the ability to generalize across different networks or domains. Temporal graphs introduce additional challenges for transfer, including evolving topologies, distribution shifts, and nonstationary dynamics that change over time. These complexities make it difficult for a model trained on one network to maintain predictive performance when applied to another. Addressing this limitation is essential for practical applications where networks evolve continuously and collecting large-scale labeled data for each is infeasible [339, 14, 340].To meet this challenge, future temporal graph models must learn representations that preserve temporal dependencies and structural invariants across heterogeneous networks [341, 342]. Such models would allow knowledge gained from past networks or tasks to be reused efficiently in unseen ones, enabling scalable and continuous learning in dynamic environments. Ultimately, achieving transferability across networks is a fundamental step toward building adaptable and general-purpose temporal graph models capable of operating across diverse real-world systems.

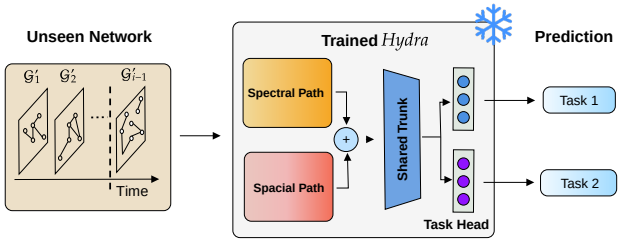


Figure 7.1: **Hydra overview.** Hydra is trained once on multiple networks and, without any additional training, can be directly applied to unseen networks for predicting multiple properties simultaneously.

In this work, we address both challenges by introducing **Hydra**, a novel temporal multi-task graph neural network designed for graph-level multi-property predictions with strong transferability across diverse networks. To capture global structural information, Hydra leverages graph Laplacian descriptors, while local node interactions and temporal dynamics are modeled through a temporal graph neural network equipped with a self-attention pooling layer. The resulting spectral and spatio-temporal representations are fused and passed through task-specific prediction heads (Figure 7.1). Trained end-to-end in a multi-task setting, Hydra achieves state-of-the-art performance on six benchmark tasks spanning classification and regression. Significantly, it generalizes across unseen networks, consistently outperforming models trained from scratch on those networks. We also analyze its scaling behavior, showing that performance improves as more training networks are incorporated. Our contributions are summarized as follows:

- **First study on multi-task learning for temporal graphs.** We propose the first model capable of handling multiple temporal graph prediction tasks without retraining, called Hydra, achieving state-of-the-art results on graph-level prediction.
- **Novel integration of TGNNs with spectral methods.** To effectively capture global information of the graph, we incorporate Laplacian descriptors into TGNNs for graph-level prediction, supported by strong empirical results.
- **Comprehensive study of transfer and scaling.** We demonstrate that Hydra surpasses the strongest baselines, achieving an 8.9% relative gain on AUC in multi-task classification and an 8.2% relative gain on MAE in two regression tasks, while remaining competitive on the third without any task-specific training. We further show that transferability to unseen networks improves as the number of training datasets increases.

7.1 Hydra: A Multi-task, Transfer Learning Model

7.1.1 Problem Definition

We define a discrete time temporal graph as a sequence of snapshots $\mathcal{G} = \{\mathcal{G}_t = (V_t, E_t, X_t)\}_{t=1}^{T}$, where V_t and E_t denote the node and edge sets at time t, and X_t represents associated node or edge features. In this work, we consider a collection of temporal graphs $\mathcal{D} = \{\mathcal{G}^{(i)}\}_{i=1}^{N}$, each evolving independently over time. For each temporal graph $\mathcal{G}^{(i)} = \{\mathcal{G}_t^{(i)}\}_{t=1}^{T_i}$ we associate task-specific labels $\{y_t^{(i,k)}\}_{k=1}^{K}$ for K prediction tasks.

Definition 1 (Multi-Task Temporal Graph Property Prediction) *The objective is to learn a single model*

$$f_\theta : \left\{ \{\{\mathcal{G}_t^{(i)}\}_{t=1}^{T-1}\}_{i=1}^{|\mathcal{D}|} \right\} \mapsto \left\{ \left(\hat{y}_t^{(i,1)}, \ldots, \hat{y}_t^{(i,K)}\right) \right\}_{i=1}^{|\mathcal{D}|},$$

that ingests all snapshots of all graphs in $\mathcal{D}$ up to time $t-1$ and jointly predicts K graph-level properties at time t for each graph. The parameters θ are optimized jointly across $\mathcal{D}$, and the trained model is required to generalize to temporal graphs not included in $\mathcal{D}$.

Hydra is evaluated on two categories of prediction tasks across multiple temporal graphs. In the **classification setting**, for $k \in \mathcal{K}_{\text{cls}}$, the objective is to predict binary outcomes $\hat{y}_t^{(i,k)} \in \{0, 1\}$ indicating the directional change of graph-level properties between consecutive snapshots. In the **regression setting**, for $k \in \mathcal{K}_{\text{reg}}$, the model predicts numerical values $\hat{y}_t^{(i,k)} \in \mathbb{R}$ that quantify the magnitude of graph-level properties at the next snapshot. In principle, this division allows Hydra to handle both discrete and continuous objectives within a single unified architecture.

Multi-task learning provides the foundation for this setting [332, 343]. Instead of training and maintaining a separate model for each task, it enables simultaneous prediction, allowing shared temporal representations to capture interdependencies between tasks [344]. Hydra builds on this foundation by unifying diverse temporal prediction tasks under a single architecture. Specifically, we focus on a *shared trunk* architecture with task-specific heads as they have proven effective in

balancing information sharing with the avoidance of negative transfer [344]. Learning tasks jointly allows the model to exploit shared temporal dynamics rather than treating each task in isolation. Figure 7.2 shows the Hydra framework.

7.1.2 Hydra: Shared Trunk with Task-Specific Heads

The central design choice of Hydra is to couple a shared representation trunk with lightweight task-specific heads. We adopt this structure because the underlying challenges in temporal graph property prediction require representations that are both transferable across heterogeneous graphs and adaptable to multiple tasks. A single trunk allows common structural and temporal information to be captured once, while task-specific heads ensure that the model retains flexibility for individual objectives.

Hydra instantiates the predictor f_θ (Definition 1) with the shared trunk that produces embeddings $H_t^{(i)}$ for $\mathcal{G}_t^{(i)}$, followed by task-specific heads $\{g_{\psi_k}\}_{k=1}^{K}$ that map $H_t^{(i)}$ to predictions $\hat{y}_t^{(i,k)}$.

We build the trunk using two complementary methodological traditions. Spectral methods, grounded in Laplacian descriptors, are chosen because they yield stable, graph-size–independent characterizations of global connectivity patterns. Such descriptors capture long-range phenomena such as expansion, bottlenecks, or fragmentation that node-level embeddings often miss [345, 346, 347]. In contrast, spatial methods based on temporal GNNs, such as T-GCN [348], excel at propagating node-level signals over time, making them effective at modeling localized interactions and dynamic neighborhood changes. By combining these two paths, Hydra unifies global invariants with local temporal dependencies, creating representations that transfer well across networks with diverse structures.

As shown in Figure 7.2, the trunk $H_t^{(i)}$ consists of a spectral path, which encodes global descriptors, and a spatial path, which models local temporal interactions. Their outputs are concatenated into a shared embedding $H_t^{(i)}$. This representation is then passed to task-specific decoder heads $g_{\psi_k}{}_{k=1}^{K}$, each producing a prediction $\hat{y}_t^{(i,k)}$. In this way, Hydra combines global reasoning for cross-network transfer with task-specific adaptation.

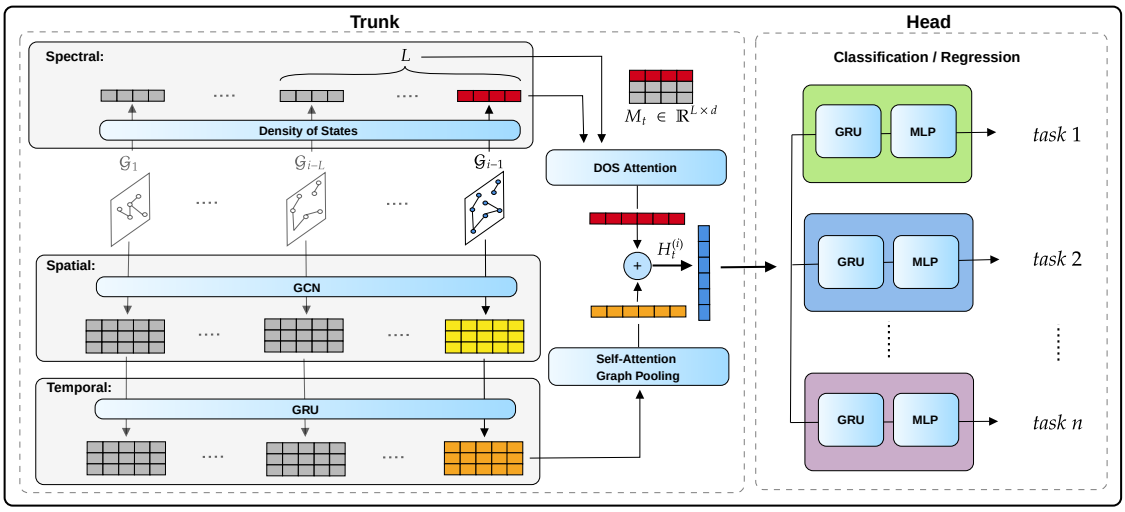


Figure 7.2: **Hydra framework.** The architecture integrates spatial and spectral paths with a unified multi-task design, enabling Hydra to learn transferable representations across diverse temporal graph tasks.

Spectral Path for Global Connectivity

Laplacian eigenvalues effectively encode global graph structural properties [349]. For example, in star, path, and fully-connected graphs, the distribution of the Laplacian eigenvalues is distinct and captures their unique topologies. However, computing all eigenvalues of the graph Laplacian matrix is highly expensive and only feasible for small graphs with thousands of nodes. Thus, it is important to efficiently approximate the spectrum of the Laplacian.

Network Density of States. To capture information from the Laplacian spectrum, we use the network density of states (DOS), the distribution of the eigenvalues of the graph Laplacian matrix [350], as follows:

Definition 2 (Density of States) *Given a temporal graph snapshot $\mathcal{G}_t$ with adjacency matrix $\mathbf{A}$ and degree matrix $\mathbf{D}$, let the (unnormalized) Laplacian be $\mathbf{L} = \mathbf{D} - \mathbf{A}$. We use the normalized Laplacian $\mathbf{L}_{\text{sym}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ with eigendecomposition $\mathbf{L}_{\text{sym}} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\top}$, where $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_{|V_t|})$ contains the eigenvalues. The* density of states *is then defined as a discrete*

distribution supported on the spectrum of $\mathbf{L}_{\text{sym}}$*:*

$$\mu(\lambda) = \frac{1}{|V_t|} \sum_{i=1}^{|V_t|} \delta(\lambda - \lambda_i),$$

Essentially, $\mu(\lambda)$ measures the fraction of eigenvalues equal to λ. In practice, the range of the eigenvalues is discretized into multiple equal-sized intervals, and the number of eigenvalues within each interval is approximated. The Kernel Polynomial Method [351] with Chebyshev expansions is used to efficiently approximate the spectrum.

Spectral Memory. To stabilize spectral signals and capture short-term trends, we maintain a fixed-length FIFO memory of recent DOS descriptors. At snapshot t, let $s_t \in \mathbb{R}^d$ denote the spectral feature (e.g., a d=20 DOS vector). For prediction at time t, the memory contains the last L descriptors up to $t-1$: $M_t = [s_{t-L}; \ldots; s_{t-1}] \in \mathbb{R}^{L \times d}$. After prediction at t, the new descriptor s_t is computed and rolled into the memory: $M_{t+1} = \text{roll}(M_t), \quad M_{t+1}[-1, :] = s_t$. This bounded memory reduces variance in non-stationary settings and preserves context across long temporal traces, typical of sparse graphs.

Attention over Spectral Memory. Rather than uniform averaging, we apply attention to emphasize historically similar or predictive patterns. With learnable projections $W_q, W_k, W_v \in \mathbb{R}^{d \times d_a}$, we compute $q_t = s_{t-1}W_q, \quad K_t = M_t W_k, \quad V_t = M_t W_v$. Attention weights and the attentive spectral embedding are $\alpha_t = \text{softmax}\left(\frac{q_t K_t^\top}{\sqrt{d_a}}\right), \quad z_t = \alpha_t V_t$. This mechanism highlights relevant history, suppresses noise, and captures structural–spectral motifs that recur across networks.

Spatial Path for Local Connectivity

We integrate spatial and temporal dynamics by combining a Graph Convolutional Network (GCN) [352] with Gated Recurrent Units (GRUs) [353]. Similar approaches have proven effective in temporal settings [348]. At snapshot t, a GCN encodes structural features $\mathbf{Z}_t = \text{GCN}(\mathbf{X}_t, \mathcal{E}_t)$, and a GRU updates node states with temporal memory $\mathbf{H}_t = \text{GRU}(\mathbf{Z}_t, \mathbf{H}_{t-1})$. The representation $\mathbf{H}_t$ is thus spatially grounded and temporally enriched.

Pooling and Graph Readout. We apply Self-Attention Graph Pooling (SAGPool) [354] to emphasize structurally important nodes. Node scores are computed as $s = \mathrm{GCN}(\mathbf{H}_t, \mathcal{E}_t)W_s$, where $W_s \in \mathbb{R}^{d\times 1}$ projects node embeddings to scalar scores. The top-k nodes are retained to form the pooled graph $(\mathcal{V}'_t, \mathcal{E}'_t)$ with embeddings $\mathbf{H}'_t$. We then compute a graph-level vector using mean pooling over the selected nodes: $g_t = \frac{1}{|\mathcal{V}'_t|}\sum_{v\in\mathcal{V}'_t} h_v, \quad h_v \in \mathbf{H}'_t$.

SAGPool reduces noise from low-activity nodes and highlights structurally central actors, which is particularly relevant in sparse networks where activity is concentrated in a few key nodes [17].

7.1.3 Prediction Head

We fuse spatial and spectral signals to produce a joint temporal state. At each step, the shared hidden state of the prediction head $u_t = \mathrm{GRU}([g_t \| z_t], u_{t-1})$, where $[g_t \| z_t]$ denotes concatenation of spatial and spectral features. The recurrent update captures cross-modal dependencies over time. Task-specific MLPs then map the shared state to predictions:

$$\hat{y}_t^{(k)} = \mathrm{MLP}_k(u_t), \quad k = 1, \ldots, K.$$

This balances efficiency through a shared backbone with flexibility through task-specific heads.

7.1.4 End-to-End Training

Hydra supports both classification and regression tasks. While multi-task learning is the intended setting, we found that naïvely combining heterogeneous objectives degraded stability, consistent with prior reports [355, 356]. A principled treatment of loss balancing is an open challenge and beyond our scope. We therefore train classification and regression groups separately, which ensures stable optimization while preserving Hydra 's transferability across unseen networks.

Classification. For each task $k \in \mathcal{T}_{\text{cls}}$, with labels $y_i \in \{0, 1\}$ and predictions $\hat{y}_i$,

$$\mathcal{L}_{\text{cls}}^{(k)} = -\frac{1}{N} \sum_{i=1}^{N} \Big[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \Big], \quad \mathcal{L}_{\text{total}}^{\text{cls}} = \sum_{k \in \mathcal{T}_{\text{cls}}} \mathcal{L}_{\text{cls}}^{(k)}.$$

Regression. For each task $k \in \mathcal{T}_{\text{reg}}$, with labels y_i and predictions $\hat{y}_i$,

$$\mathcal{L}_{\text{reg}}^{(k)} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2, \quad \mathcal{L}_{\text{total}}^{\text{reg}} = \sum_{k \in \mathcal{T}_{\text{reg}}} \mathcal{L}_{\text{reg}}^{(k)}.$$

7.2 Experimental Evaluation

7.2.1 Dataset

We evaluate Hydra on temporal transaction networks from the MiNT benchmark [18], the first large-scale dataset designed for training and evaluating temporal graph models across multiple heterogeneous networks. MiNT contains 84 ERC-20 token transaction graphs collected from the Ethereum blockchain between 2017 and 2023, spanning more than six years of activity. The biggest MiNT token network contains $128,159$ unique addresses and $554,705$ transactions, while the smallest token network has $1,454$ nodes. Each network is represented as a sequence of weekly snapshots, where nodes correspond to wallet addresses and edges represent token transfers. Across their full duration, networks range from tens of thousands to over 100K unique nodes and up to several million edges, reflecting the scale of real-world token ecosystems.

A distinctive feature of MiNT is its strong inductiveness: new addresses and transactions appear continuously, making prediction inherently open-world. This reflects the real behavior of blockchain networks, where adoption cycles, liquidity shocks, and market events constantly introduce novel participants and structural patterns. The long temporal duration further amplifies this effect, as networks exhibit bursts of rapid growth, periods of stagnation, and sudden fragmentation or collapse. Such novelty and surprise factors make MiNT particularly challenging, as models must generalize across highly dynamic trajectories rather than stationary or repetitive patterns.

We summarize detailed statistics of each token network in MiNT datasets in Table 6.1. Most networks have more than 10k nodes and over 100k edges. The lifespan of MiNT networks varies from 107 days to 6 years, and there exists at least one transaction each day. As the table shows, the token networks have quite high surprise values with an average of 0.82.

For our experiments, we follow the MiNT protocol and split the dataset into 64 financial networks for training and 20 additional networks for unseen testing. This setup allows Hydra to learn transferable temporal representations from diverse source networks and evaluate zero-shot generalization on new target networks that are only available at inference time.

7.2.2 Task Formalizations

In this section, we provide detailed definitions for all classification and regression tasks considered in Hydra. Each temporal snapshot corresponds to a 7-day interval, and the property prediction setup follows GraphPulse [17]. We also note that some of these tasks and the corresponding labels were processed specifically for this work, ensuring consistency and comparability across networks.

Setting $n = 7$, $\delta_1 = 3$, and $\delta_2 = 10$ days, we establish a practical graph property with a 7-day prediction window. This choice is particularly relevant in financial contexts, such as Ethereum asset networks, where it can guide investment decisions [266].

Classification Tasks

Node Growth/Shrinkage (Node G/S). Let $V(t_1, t_n)$ denote the set of unique nodes active between times t_1 and t_n. The task predicts whether the number of nodes increases in the prediction interval $[t_{n+\delta_1}, t_{n+\delta_2}]$:

$$P_{\text{nodes}}(\mathcal{G}, t_1, t_n, \delta_1, \delta_2) = \begin{cases} 1, & \text{if } |V(t_{n+\delta_1}, t_{n+\delta_2})| > |V(t_1, t_n)|, \\ 0, & \text{otherwise.} \end{cases} \tag{7.1}$$

Importance. Node growth measures adoption, reflecting the entry of new addresses, while shrinkage signals attrition. In token ecosystems, this corresponds to market expansion or decline.

Edge Growth/Shrinkage (Edge G/S). Let $E(t_1, t_n)$ denote the set of transactions in $[t_1, t_n]$. The model predicts whether transaction activity grows in the next interval:

$$P_{\text{edges}}(\mathcal{G}, t_1, t_n, \delta_1, \delta_2) = \begin{cases} 1, & \text{if } |E(t_{n+\delta_1}, t_{n+\delta_2})| > |E(t_1, t_n)|, \\ 0, & \text{otherwise.} \end{cases} \tag{7.2}$$

Importance. Edge growth captures changes in liquidity and market engagement, with direct implications for trading activity and token valuation.

Largest Connected Component Growth/Shrinkage (LCC G/S). Let $C(t_1, t_n)$ denote the size of the largest connected component during $[t_1, t_n]$. The task is to predict whether connectivity expands:

$$P_{\text{LCC}}(\mathcal{G}, t_1, t_n, \delta_1, \delta_2) = \begin{cases} 1, & \text{if } |C(t_{n+\delta_1}, t_{n+\delta_2})| > |C(t_1, t_n)|, \\ 0, & \text{otherwise.} \end{cases} \tag{7.3}$$

Importance. LCC growth reflects stronger structural integration, improving liquidity and market stability in blockchain networks.

Regression Tasks

New Node Count. Let $V_{\text{new}}(t_{n+\delta_1}, t_{n+\delta_2})$ denote the set of nodes that appear for the first time in the prediction interval. The regression target is:

$$\hat{y}_{\text{new-nodes}}(\mathcal{G}, t_1, t_n, \delta_1, \delta_2) = |V_{\text{new}}(t_{n+\delta_1}, t_{n+\delta_2})|. \tag{7.4}$$

Importance. New node prediction quantifies adoption and user acquisition, a key indicator of ecosystem growth.

Edge Count. The task is to predict the number of transactions in the future interval:

$$\hat{y}_{\text{edges}}(\mathcal{G}, t_1, t_n, \delta_1, \delta_2) = |E(t_{n+\delta_1}, t_{n+\delta_2})|. \tag{7.5}$$

Importance. Transaction forecasts provide fine-grained estimates of liquidity and demand surges in decentralized markets.

Influential Node Count. Define influential nodes as those with a degree greater than 5 in the prediction interval. Empirically, degree distributions of ERC-20 token graphs are heavy-tailed [18]: most nodes appear only once or twice, and a small fraction appear hundreds or thousands of times. When we plot the cumulative distribution of node degrees, there is usually a sharp drop in frequency after degree 1–2, followed by a long but thinner tail. Setting the threshold at 5 sits just beyond this long tail cutoff, ensuring that only the top 10–20% of nodes in activity are retained as influential. In practice, this excludes wallets that perform only a handful of transfers while capturing the repeat participants who actually shape liquidity and flow.

Let $V_{\text{inf}}(t_{n+\delta_1}, t_{n+\delta_2}) = \{v \in V : \deg(v) > 5\}$. The regression target is:

$$\hat{y}_{\text{inf}}(\mathcal{G}, t_1, t_n, \delta_1, \delta_2) = |V_{\text{inf}}(t_{n+\delta_1}, t_{n+\delta_2})|. \tag{7.6}$$

Importance. Influential nodes represent hubs such as active traders, liquidity providers, or contracts, which shape the stability and price dynamics of token ecosystems.

7.2.3 Training Setup and Algorithm

We train Hydra using the same multi-network framework introduced in MINT [18], extending it to support multi-task learning across both classification and regression objectives. Hydra has been trained in two multi-task settings: a classification model with three classification heads and a regression model with three regression heads for jointly prediction. Temporal networks are divided into 70%–15%–15% splits for training, validation, and testing, and during each epoch, networks are presented in random order with historical embeddings reset at the start of each net-

work to avoid cross-network leakage. The shared trunk is optimized jointly across all datasets, while task-specific heads are updated under a unified multi-task loss. Binary Cross-Entropy is used for classification tasks, Mean Squared Error for regression, and optimization is performed with Adam [319] using an initial learning rate of 1.5×10^{-3}. Models are trained for 100–250 epochs with early stopping based on validation AUC, using a patience of 20 epochs and a tolerance of 5×10^{-2}. Each task, defined in section 6.2.1, is evaluated under a zero-shot transfer setting on 20 held-out temporal networks, repeated three times with independent random seeds to obtain 60 evaluation points per task. ROC-AUC and Mean Absolute Error (MAE) are reported for classification and regression objectives, respectively. We pre-train Hydra with 8, 16, and 32 source networks to study scalability, and adopt Hydra-32 as the main configuration, as it consistently achieves the best performance. For comparison,We evaluate Hydra in both multi-task classification and regression settings against state-of-the-art temporal graph learning baselines, including HTGN [126], GC-LSTM [127], EvolveGCN [128], GraphPulse [17], ROLAND [108], TGCN [125], and WinGNN [129] (As detailed in section 2.2.6). For classification, we additionally include MiNT [18], the only existing framework explicitly designed for transferability across temporal graphs. MiNT is excluded from regression since it does not support regression objectives. We distinguish between two categories of approaches: (i) *single models*, trained separately on each dataset, and (ii) *transferable models*, trained once and applied directly to unseen networks without retraining. Hydra belongs to the latter category, enabling true zero-shot transfer. All experiments are conducted on a dedicated compute node with an NVIDIA H100 GPU (80 GB), dual 64-core AMD EPYC CPUs, and 512 GB RAM.

7.2.4 Classification and Regression Results

Table 7.1 presents the aggregate classification results across three tasks: LCC Growth/Shrinkage, Node Growth/Shrinkage, and Edge Growth/Shrinkage. For each task, we report three metrics: (*1st*) the number of datasets where the method achieved first place, (*Rank*) the average ranking across datasets, and (*AUC*) the average area under the ROC curve (metrics formulations are provided in

Table 7.1: Classification task summary results. Across LLC-GS, Node-GS, and Edge-G/S classification, Hydra consistently outperforms both specifically trained and transferable baselines.

	LLC-G/S			Node-G/S			Edge-G/S		
	1st↑	Rank↓	AUC↑	1st↑	Rank↓	AUC↑	1st↑	Rank↓	AUC↑
Single Models									
HTGN	2	4.40	0.648	0	3.65	0.615	1	4.85	0.684
GC-LSTM	0	6.00	0.572	0	6.65	0.489	1	5.80	0.609
EvolveGCN	0	6.30	0.585	0	7.00	0.487	0	6.10	0.626
GraphPulse	3	3.50	0.686	2	2.95	0.652	6	3.80	0.712
ROLAND	0	6.95	0.556	0	6.70	0.480	0	6.30	0.542
TGCN	1	6.10	0.583	0	6.15	0.530	0	5.60	0.633
WinGNN	0	6.10	0.585	0	6.20	0.528	0	5.55	0.642
Transfer Models									
MiNT	1	3.95	0.672	3	3.55	0.616	4	3.30	0.727
Hydra (ours)	**13**	**1.70**	**0.759**	**15**	**1.95**	**0.734**	**8**	**2.80**	**0.753**

Arrows indicate directionality: ↑ higher is better, ↓ lower is better.

Table 7.2: Regression task summary results. Across New Node Count, Influential Node Count, and Edge Count regression, Hydra achieves the best performance on two tasks and is competitive on the third, all without additional training.

	New Node Count			Influential Node Count			Edge Count		
	1st↑	Rank↓	MAE↓	1st↑	Rank↓	MAE↓	1st↑	Rank↓	MAE↓
Single Models									
HTGN	4	3.45	0.039	4	**3.52**	**0.050**	2	3.95	0.049
TGCN	1	4.10	0.042	1	4.85	0.059	1	4.60	0.054
GCLSTM	2	4.00	0.043	**5**	3.80	0.053	4	4.58	0.057
ROLAND	3	4.35	0.052	2	4.55	0.057	3	4.58	0.050
EGCN	0	3.90	0.040	0	4.20	0.052	1	5.30	0.054
GraphPulse	1	5.75	0.071	1	5.90	0.078	2	5.72	0.073
WinGNN	4	4.05	0.054	4	4.28	0.059	5	3.93	0.057
Transfer Model									
Hydra (ours)	**8**	**2.50**	**0.035**	**5**	4.90	0.056	**6**	**3.35**	**0.046**

Arrows indicate directionality: ↑ higher is better, ↓ lower is better.

section 2.2.3). Hydra consistently achieves the strongest performance across all metrics, outperforming both state-of-the-art specifically trained models and the transferable baseline. Detailed per-dataset classification outcomes are provided in section 7.2.9.

Table 7.2 summarizes the regression results across New Node Count, Influential Node Count, and Edge Count Regression. For each task, we report three metrics: (*1st*) the number of datasets where the method achieved first place, (*Rank*) the average ranking across datasets, and (*MAE*) the average mean absolute error. Hydra is the best model on two of the three tasks and remains very close to the best baseline on Influential Node Count, demonstrating strong overall transferability

for regression objectives. Full results by dataset are included in section 7.2.9.

Hydra produces the most top-ranked results in both classification and regression settings, outperforming all baselines in five out of six tasks. To keep the main paper concise, we report summarized results in the main text and provide extended per-dataset outcomes in section 7.2.9. Even in the one regression task where Hydra isn't the leader, it stays competitive with the strongest baseline. Notably, Hydra is always tested on unseen networks, while single models specifically trained for a task are evaluated on the test portion of the same networks they were trained on. Despite this tougher setting, Hydra consistently delivers better or comparable results, making it a reliable and adaptable framework for temporal graph learning.

Ablation Studies. We assess the impact of Hydra's key components by disabling the DOS module and the pooling layer. As summarized in Table 7.3, DOS improves performance from an average AUC of 0.697 to 0.719, raising the number of best-performing datasets from 6 to 14 and lowering the mean rank from 1.70 to 1.30. Pooling yields an even larger effect, increasing average AUC from 0.719 to 0.753 and improving the mean rank from 1.85 to 1.15, with gains on 17 out of 20 datasets. These results confirm that DOS provides complementary global spectral information, while pooling enhances the spatial path by emphasizing structurally important nodes, together strengthening Hydra's generalization ability.

Table 7.3: Meta ablation summary for Edge G/S classification. Average AUC and #1st-Place counts are reported with and without each component. Per-network results are provided in Tables 7.4 and 7.5.

Ablation	Avg. AUC		#1st-Place	
	w/o	w/	w/o	w/
Density of States	0.697	**0.719**	6	**14**
Attention Pooling	0.719	**0.753**	3	**17**

7.2.5 Hydra Generalization and Efficiency

In this section, we analyze how the number of networks used during the pre-training of Hydra affects its performance in predicting the characteristics of unseen networks at inference time. The results are presented in Fig. 7.3.

For classification tasks, we observe that increasing the number of training networks generally

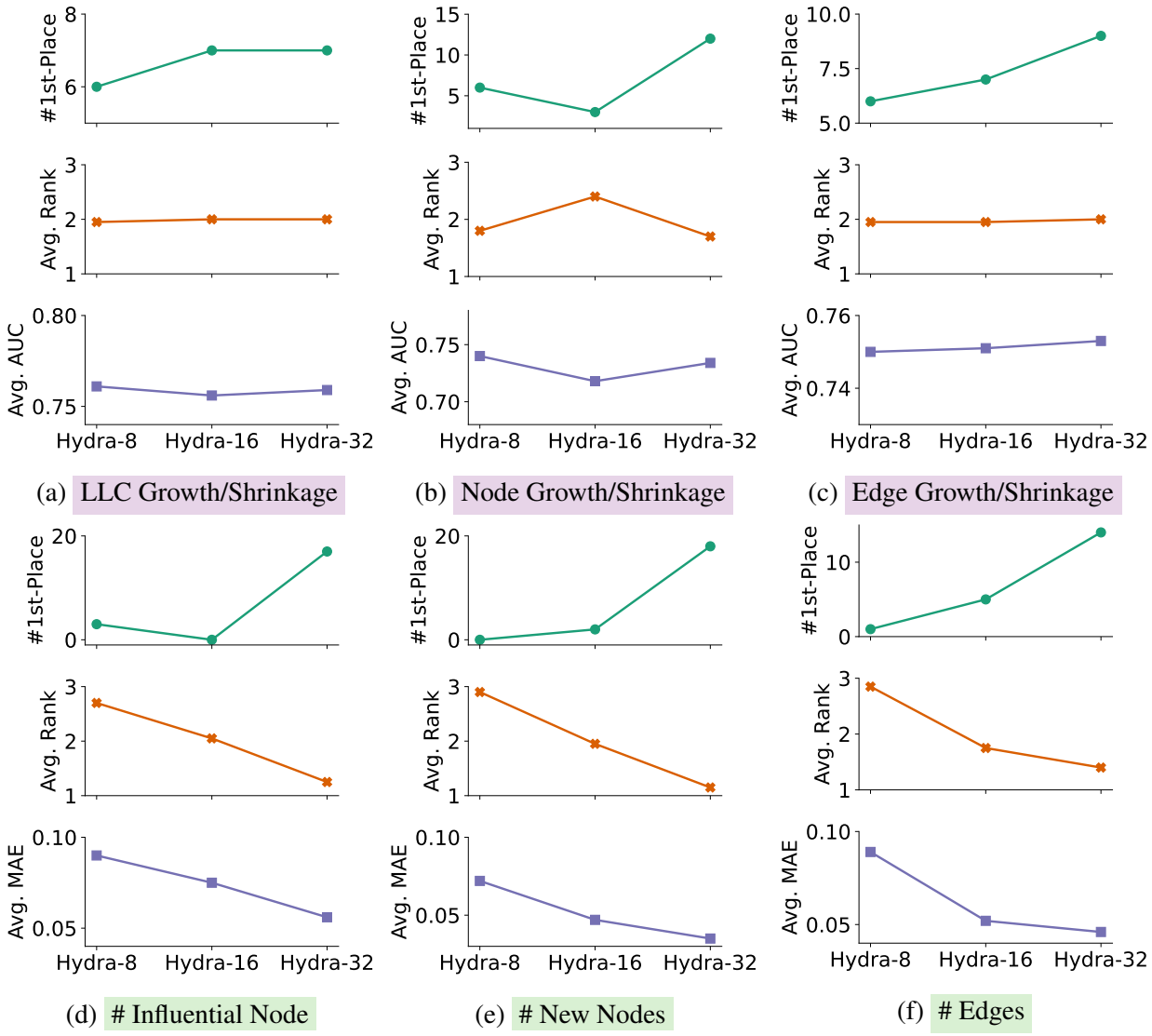
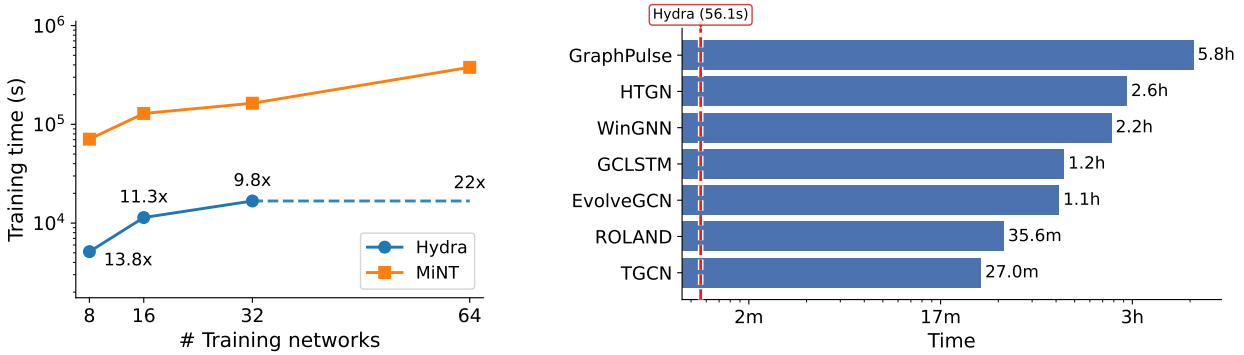


Figure 7.3: Impact of scaling the number of training networks for Hydra (from 8 to 32) on classification and regression tasks. A clear scaling trend is observed in most cases, with **Hydra-32** consistently delivering the best performance across all tasks. Evaluation measures are $\#1^{st}$-*Place*↑ (higher is better), *Avg. Rank*↓ (lower is better), *Avg. AUC*↑ (higher is better), and *Avg. MAE*↓ (lower is better).

leads to a higher $\#1^{st}$-*Place*. This metric is a strong indicator of performance, as it reflects the number of unseen networks on which the model achieves the best results at inference time. In contrast, the correlation between the number of pre-training networks and metrics such as Avg. Rank or Avg. AUC is weaker, with only negligible differences across settings. Notably, Hydra-32 consistently achieves the best or on-par performance across all classification benchmarks. For regression tasks, we observe a clearer scaling trend: performance steadily improves from Hydra-8



(a) **Total training cost for three classification tasks.** Hydra trains once for all tasks jointly, while MiNT must be retrained separately for each task.

(b) **Inference cost on TRAC dataset.** Hydra performs a single zero-shot classification pass for all three tasks jointly, whereas single-network baselines require full training before they can be used for inference, even on a single task (LCC–G/S here); reported times reflect the cost of inference on an unseen network using Hydra versus single models.

Figure 7.4: **Training and inference efficiency of Hydra.** (a) Hydra reduces training cost by avoiding retraining. (b) Hydra drastically lowers inference time by requiring only one forward pass.

to Hydra-32 across all indicators. Specifically, $\#1^{st}$-*Place* increases with scale, while both Avg. Rank and Avg. MAE decrease, indicating better predictive performance.

Overall, Hydra-32 consistently achieves the strongest results, with the highest $\#1^{st}$-*Place*, the lowest Avg. Rank, and the best average performance metric (i.e., highest Avg. AUC for classification and lowest Avg. MAE for regression). Given its consistent superiority across all six tasks, we adopt **Hydra-32** as our primary model and center our subsequent evaluation on its performance. For completeness, Table 7.6 and Table 7.7 in section 7.2.8 reports the performance of all Hydra variants.

Computation and Time Efficiency. The per-snapshot complexity of Hydra reduces to a scalable $\mathcal{O}(N \cdot m + n \log n)$, where m is the number of edges, n is the number of nodes, and N is the number of Chebyshev moments used in the DOS module (see section 7.2.7 for details). Because MiNT must be retrained separately for each task, training three tasks requires about three full runs on 64 networks. In contrast, Hydra trains all tasks jointly in a single run on 32 networks and achieves a $22\times$ speedup, completing all three tasks in about $16{,}700$ seconds compared to more than $370{,}000$ seconds for MiNT (Figure 7.4a). This efficiency makes Hydra far more practical and scalable for

multi-task training and inference.

Hydra inference time also shows a significant gain. Hydra requires only a single zero-shot inference pass to predict all three tasks on an unseen network, completing end-to-end evaluation on the largest test network (TRAC) in about 56 seconds (see Figure 7.4b). Single-network baselines, in contrast, require retraining from scratch for every dataset and task before they can be used for inference, with training times ranging from several thousand seconds to over $9{,}000$ seconds even for a single task. Extending these models to multiple tasks or multiple networks multiplies the cost linearly, making them infeasible at scale. MiNT 's design eliminates the need for repeated training and heavy preprocessing, providing unique efficiency in both training and inference.

7.2.6 Ablation Studies

Table 7.4: DOS ablation study on task Edge G/S. AUC values are reported. Best results per dataset are in **bold**.

Dataset	Hydra **w/o DOS**	Hydra **w DOS**
MIR	**0.791 (±0.014)**	0.755 (±0.013)
DOGE2.0	0.500 (±0.102)	**0.732 (±0.044)**
MUTE	0.675 (±0.030)	**0.699 (±0.011)**
EVERMOON	0.426 (±0.129)	**0.796 (±0.219)**
DERC	0.748 (±0.029)	**0.761 (±0.030)**
ADX	0.678 (±0.019)	**0.703 (±0.006)**
HOICHI	**0.792 (±0.052)**	0.572 (±0.016)
SDEX	**0.599 (±0.134)**	0.261 (±0.129)
BAG	0.759 (±0.294)	**0.874 (±0.017)**
XCN	0.822 (±0.073)	**0.825 (±0.038)**
ETH2x-FLI	0.716 (±0.020)	**0.736 (±0.026)**
stkAAVE	0.708 (±0.021)	**0.719 (±0.006)**
GLM	0.755 (±0.194)	**0.810 (±0.015)**
QOM	0.639 (±0.015)	**0.698 (±0.024)**
WOJAK	0.548 (±0.081)	**0.633 (±0.081)**
DINO	0.672 (±0.044)	**0.843 (±0.031)**
Metis	**0.785 (±0.050)**	0.713 (±0.027)
REPv2	**0.761 (±0.039)**	0.736 (±0.028)
TRAC	**0.794 (±0.041)**	0.732 (±0.007)
BEPRO	0.775 (±0.022)	**0.786 (±0.009)**
1st-Place Count↑	6	**14**
Avg. Rank ↓	1.70	**1.30**
Avg. AUC ↑	0.697	**0.719**

Table 7.5: SAG pooling ablation study on task Edge G/S. AUC values are reported. Best results per dataset are in **bold**.

Dataset	Hydra **w/o Pooling**	Hydra **w Pooling**
MIR	0.755 (±0.013)	**0.793 (±0.002)**
DOGE2.0	0.732 (±0.044)	**0.897 (±0.089)**
MUTE	0.699 (±0.011)	**0.701 (±0.098)**
EVERMOON	0.796 (±0.219)	**0.818 (±0.046)**
DERC	0.761 (±0.030)	**0.839 (±0.008)**
ADX	0.703 (±0.006)	**0.722 (±0.087)**
HOICHI	0.572 (±0.016)	**0.591 (±0.103)**
SDEX	0.261 (±0.129)	**0.348 (±0.066)**
BAG	0.874 (±0.017)	**0.969 (±0.008)**
XCN	0.825 (±0.038)	**0.844 (±0.012)**
ETH2x-FLI	**0.736 (±0.026)**	0.712 (±0.045)
stkAAVE	0.719 (±0.006)	**0.732 (±0.011)**
GLM	0.810 (±0.015)	**0.850 (±0.009)**
QOM	0.698 (±0.024)	**0.745 (±0.003)**
WOJAK	**0.633 (±0.081)**	0.585 (±0.067)
DINO	0.843 (±0.031)	**0.895 (±0.003)**
Metis	0.713 (±0.027)	**0.733 (±0.004)**
REPv2	0.736 (±0.028)	**0.772 (±0.016)**
TRAC	**0.732 (±0.007)**	0.722 (±0.001)
BEPRO	0.786 (±0.009)	**0.800 (±0.002)**
1st-Place Count↑	3	**17**
Avg. Rank ↓	1.85	**1.15**
Avg. AUC ↑	0.719	**0.753**

To better understand Hydra components, we conduct an ablation study on the edge growth/shrinkage task. The ablations reflect different stages in the model's incremental development. First, we com-

pare the initial version of Hydra without pooling, evaluating the impact of adding Density of States (DOS) features in the spectral path. Next, we consider models that already include DOS and assess the effect of introducing attention-based pooling in the spatial path. Results are reported in Tables 7.4 and 7.5. Together, these comparisons show that DOS enhances the capture of global spectral structure, while pooling improves the selection of informative subgraphs, confirming their complementary roles in Hydra's design and performance.

Impact of DOS. To assess the contribution of the DOS module, we compare Hydra with and without DOS features (Table 7.4). This setup isolates the effect of spectral information on predictive performance, holding all other components fixed. The results show that incorporating DOS substantially improves performance on most datasets: Hydra with DOS achieves the best results in 14 out of 20 networks, compared to 6 without DOS. On average, DOS raises AUC from 0.697 to 0.719 and improves the mean rank from 1.70 to 1.30. Gains are particularly large on challenging datasets such as *EVERMOON*, *DOGE2.0*, and *DINO*, where DOS more effectively captures global spectral structure. These findings highlight that DOS provides complementary information to the spatial path, leading to stronger generalization across temporal networks.

Impact of Attention-based Pooling. The ablation on the edge growth/shrinkage task (Table 7.5) shows the contribution of the pooling mechanism within Hydra's spatial path. When pooling is enabled, the model achieves higher AUC on 17 out of 20 datasets, with the average score improving from 0.719 to 0.753 and the mean rank from 1.85 to 1.15. The improvement is particularly noticeable when pooling helps the spatial path focus on structurally important nodes, resulting in stronger graph-level representations. These findings indicate that pooling is an essential component for producing compact, informative graph-level representations and strengthen Hydra's ability to capture temporal dynamics.

7.2.7 Computational Complexity of Hydra

For a snapshot $G_t = (V_t, E_t)$ with $n = |V_t|$ nodes and $m = |E_t|$ edges, the per-snapshot complexity of Hydra is

$$\mathcal{O}\Big(m \cdot d + n \cdot d^2 + n \cdot \log n + N \cdot m + K \cdot d\Big),$$

where d is the hidden dimension, K is the number of task-specific heads, and N is the number of Chebyshev moments used in the kernel polynomial method approximation of the DOS. The term $m \cdot d$ arises from spatial message passing, since each edge propagates a d-dimensional embedding. The term $n \cdot d^2$ comes from recurrent or attention-based updates. Pooling with top-k selection contributes $n \cdot \log n$, reflecting the computation of attention scores for all nodes followed by partial sorting. The spectral DOS module adds $N \cdot m$, as each moment requires one sparse matrix–vector multiplication. Finally, the cost of task-specific heads is $K \cdot d$. Over N networks with T snapshots each, the overall training cost scales as

$$\mathcal{O}\Big(N \cdot T\,[\,m \cdot (d + N) + n \cdot d^2 + n \cdot \log n + K \cdot d\,]\Big)\,.$$

In our implementation, the spatial hidden dimension is fixed to 16, the DOS descriptor dimension is 20, and we train with $K = 3$ task-specific heads. Substituting these values, the per-snapshot complexity of Hydra becomes

$$\mathcal{O}\Big(16 \cdot m + 256 \cdot n + n \log n + N \cdot m + 108\Big),$$

where $m = |E_t|$ and $n = |V_t|$. The terms correspond to spatial message passing ($16 \cdot m$), recurrent/attention updates ($256 \cdot n$), pooling with top-k selection ($n \log n$), DOS spectral approximation with N Chebyshev moments ($N \cdot m$), and three task heads on a 36-dimensional joint embedding (108). For each snapshot, the overall training complexity scales as

$$\mathcal{O}\Big(\Big((16 + N)\,m + 256\,n + n \log n\Big)\Big)\,.$$

In practice, with hidden size and Chebyshev moment parameters fixed, the per-snapshot complexity of Hydra reduces to $\mathcal{O}(N \cdot m + n \log n)$, dominated by sparse matrix–vector multiplications in the DOS module.

7.2.8 Scaling Behavior in Hydra

We conducted scaling trend experiments to evaluate Hydra under different training pack sizes systematically. Four model variations were trained using 8, 16, and 32 networks, enabling us to observe how increasing the number of networks in the training loop affects performance. This setup provides a consistent method to study Hydra's behavior when exposed to varying amounts of training data across multiple tasks. For each task, we present results for all four Hydra variants. These results demonstrate how Hydra scales with the number of training networks and are shown in the following tables. Each table provides a detailed breakdown by task and model variation, offering a comprehensive view of performance under various scaling setups. The detailed results of each Hydra variation trained with different datapacks across all three classification tasks are reported in Table 7.6. Subsections (a)–(c) present classification tasks (Node G/S, Edge G/S, LCC G/S). Also, detailed results for three regression are reported in Table 7.7 while (a)–(c) correspond to regression tasks (New Node Count, Influential Node Count, Edge Count).

Table 7.6: Performance of Hydra on three classification tasks as the number of training networks increases from 8 to 32. The results show how scaling the number of training networks improves the model's accuracy and generalization, highlighting the benefit of multi network learning in temporal graph classification.

(a) Classification : Node G/S

Dataset	Hydra-8	Hydra-16	Hydra-32
MIR	**0.769 (±0.003)**	**0.769 (±0.007)**	0.764 (±0.001)
DOGE2.0	0.613 (±0.083)	0.573 (±0.133)	**0.633 (±0.064)**
MUTE	0.755 (±0.016)	**0.763 (±0.003)**	0.748 (±0.025)
EVERMOON	0.624 (±0.005)	0.582 (±0.031)	**0.655 (±0.040)**
DERC	0.734 (±0.010)	0.725 (±0.006)	**0.742 (±0.004)**
ADX	0.753 (±0.027)	**0.767 (±0.004)**	0.718 (±0.051)
HOICHI	**0.582 (±0.029)**	0.552 (±0.050)	0.558 (±0.047)
SDEX	**0.762 (±0.008)**	0.748 (±0.068)	0.743 (±0.026)
BAG	0.963 (±0.010)	0.952 (±0.021)	**0.969 (±0.009)**
XCN	0.862 (±0.017)	0.821 (±0.070)	**0.878 (±0.009)**
ETH2x-FLI	**0.717 (±0.030)**	0.710 (±0.009)	0.678 (±0.031)
stkAAVE	0.776 (±0.009)	0.776 (±0.003)	**0.779 (±0.007)**
GLM	0.750 (±0.011)	0.746 (±0.010)	**0.763 (±0.005)**
QOM	0.703 (±0.012)	0.707 (±0.013)	**0.719 (±0.012)**
WOJAK	**0.502 (±0.103)**	0.352 (±0.157)	0.412 (±0.060)
DINO	0.903 (±0.029)	0.905 (±0.021)	**0.910 (±0.013)**
Metis	0.685 (±0.002)	0.632 (±0.091)	**0.693 (±0.007)**
REPv2	**0.728 (±0.022)**	0.721 (±0.023)	0.689 (±0.013)
TRAC	0.756 (±0.005)	0.745 (±0.014)	**0.765 (±0.003)**
BEPRO	0.858 (±0.023)	0.806 (±0.094)	**0.865 (±0.010)**
1st-Place Count↑	6	3	**12**
Avg. Rank↓	1.85	2.45	**1.70**
Avg. AUC↑	**0.740**	0.718	0.734

(b) Classification : Edge G/S

Dataset	Hydra-8	Hydra-16	Hydra-32
MIR	**0.800 (±0.007)**	0.796 (±0.001)	0.793 (±0.002)
DOGE2.0	0.859 (±0.022)	**0.897 (±0.089)**	**0.897 (±0.089)**
MUTE	0.757 (±0.004)	**0.764 (±0.004)**	0.701 (±0.098)
EVERMOON	0.698 (±0.180)	0.750 (±0.024)	**0.818 (±0.046)**
DERC	0.823 (±0.018)	0.826 (±0.002)	**0.839 (±0.008)**
ADX	0.760 (±0.004)	**0.768 (±0.015)**	0.722 (±0.087)
HOICHI	0.606 (±0.015)	**0.617 (±0.026)**	0.591 (±0.103)
SDEX	0.288 (±0.023)	0.331 (±0.049)	**0.348 (±0.066)**
BAG	**0.969 (±0.009)**	0.955 (±0.015)	**0.969 (±0.008)**
XCN	**0.847 (±0.006)**	0.845 (±0.010)	0.844 (±0.012)
ETH2x-FLI	0.735 (±0.006)	**0.738 (±0.005)**	0.712 (±0.045)
stkAAVE	**0.744 (±0.008)**	0.743 (±0.007)	0.732 (±0.011)
GLM	0.849 (±0.003)	0.841 (±0.015)	**0.850 (±0.009)**
QOM	0.755 (±0.015)	**0.762 (±0.008)**	0.745 (±0.003)
WOJAK	**0.627 (±0.014)**	0.561 (±0.101)	0.585 (±0.067)
DINO	0.875 (±0.030)	0.889 (±0.009)	**0.895 (±0.003)**
Metis	**0.735 (±0.004)**	0.697 (±0.062)	0.733 (±0.004)
REPv2	0.778 (±0.001)	**0.784 (±0.009)**	0.772 (±0.016)
TRAC	0.713 (±0.010)	0.711 (±0.011)	**0.722 (±0.001)**
BEPRO	0.783 (±0.009)	0.743 (±0.085)	**0.800 (±0.002)**
1st-Place Count↑	6	7	**9**
Avg. Rank↓	**1.95**	**1.95**	2
Avg. AUC↑	0.75	0.751	**0.753**

(c) Classification : LCC G/S

Dataset	Hydra-8	Hydra-16	Hydra-32
MIR	**0.819 (±0.004)**	0.817 (±0.001)	0.815 (±0.007)
DOGE2.0	**0.762 (±0.010)**	0.667 (±0.175)	0.702 (±0.160)
MUTE	0.714 (±0.049)	**0.756 (±0.006)**	0.704 (±0.085)
EVERMOON	0.600 (±0.057)	0.639 (±0.029)	**0.667 (±0.045)**
DERC	**0.819 (±0.003)**	0.813 (±0.005)	0.808 (±0.022)
ADX	0.691 (±0.066)	**0.770 (±0.021)**	0.727 (±0.093)
HOICHI	0.610 (±0.044)	**0.640 (±0.041)**	0.638 (±0.075)
SDEX	0.784 (±0.017)	0.810 (±0.048)	**0.816 (±0.025)**
BAG	0.968 (±0.013)	0.957 (±0.021)	**0.976 (±0.006)**
XCN	0.863 (±0.011)	0.863 (±0.029)	**0.887 (±0.014)**
ETH2x-FLI	**0.708 (±0.011)**	0.703 (±0.001)	0.687 (±0.032)
stkAAVE	0.753 (±0.011)	**0.757 (±0.003)**	0.748 (±0.012)
GLM	**0.853 (±0.006)**	0.844 (±0.020)	0.848 (±0.012)
QOM	0.725 (±0.009)	**0.730 (±0.002)**	0.729 (±0.010)
WOJAK	**0.558 (±0.032)**	0.458 (±0.142)	0.500 (±0.083)
DINO	0.878 (±0.021)	**0.883 (±0.032)**	0.818 (±0.049)
Metis	0.727 (±0.006)	0.669 (±0.100)	**0.731 (±0.001)**
REPv2	0.781 (±0.004)	**0.785 (±0.009)**	0.764 (±0.020)
TRAC	0.772 (±0.006)	0.768 (±0.008)	**0.781 (±0.004)**
BEPRO	0.826 (±0.015)	0.800 (±0.058)	**0.830 (±0.006)**
1st-Place Count↑	6	**7**	**7**
Avg. Rank↓	**1.95**	2	2
Avg. AUC↑	**0.761**	0.756	0.759

Table 7.7: Performance of Hydra on three regression tasks as the number of training networks increases from 8 to 32. The results illustrate how larger and more diverse training data enhance the model's predictive consistency and regression accuracy across temporal graph tasks.

(a) Regression : New Node Count

Dataset	Hydra-8	Hydra-16	Hydra-32
MIR	0.056 (±0.057)	0.040 (±0.042)	**0.013 (±0.004)**
DOGE2.0	0.068 (±0.023)	**0.060 (±0.027)**	0.092 (±0.008)
MUTE	0.073 (±0.036)	0.036 (±0.025)	**0.025 (±0.005)**
EVERMOON	0.070 (±0.053)	0.038 (±0.045)	**0.012 (±0.005)**
DERC	0.071 (±0.049)	0.038 (±0.040)	**0.015 (±0.004)**
ADX	0.069 (±0.047)	0.029 (±0.035)	**0.016 (±0.005)**
HOICHI	0.064 (±0.038)	0.048 (±0.030)	**0.029 (±0.008)**
SDEX	0.074 (±0.021)	0.072 (±0.011)	**0.063 (±0.006)**
BAG	0.072 (±0.023)	0.056 (±0.009)	**0.052 (±0.002)**
XCN	0.067 (±0.056)	0.036 (±0.044)	**0.009 (±0.002)**
ETH2x-FLI	0.053 (±0.034)	0.032 (±0.023)	**0.030 (±0.004)**
stkAAVE	0.085 (±0.037)	**0.057 (±0.024)**	0.078 (±0.017)
GLM	0.097 (±0.013)	0.103 (±0.008)	**0.094 (±0.007)**
QOM	0.073 (±0.054)	0.034 (±0.040)	**0.012 (±0.001)**
WOJAK	0.076 (±0.059)	0.034 (±0.042)	**0.009 (±0.000)**
DINO	0.065 (±0.053)	0.046 (±0.042)	**0.018 (±0.007)**
Metis	0.072 (±0.030)	0.037 (±0.016)	**0.034 (±0.010)**
REPv2	0.083 (±0.024)	0.074 (±0.015)	**0.066 (±0.004)**
TRAC	0.074 (±0.042)	0.029 (±0.026)	**0.022 (±0.005)**
BEPRO	0.081 (±0.067)	0.034 (±0.046)	**0.004 (±0.001)**
1st-Place Count↑	0	2	**18**
Avg. Rank↓	2.90	1.95	**1.15**
Avg. MAE↓	0.072	0.047	**0.035**

(b) Regression : Influential Node Count

Dataset	Hydra-8	Hydra-16	Hydra-32
MIR	0.080 (±0.059)	0.059 (±0.004)	**0.039 (±0.005)**
DOGE2.0	**0.065 (±0.034)**	0.094 (±0.003)	0.126 (±0.032)
MUTE	0.103 (±0.070)	0.078 (±0.034)	**0.045 (±0.027)**
EVERMOON	0.107 (±0.067)	0.072 (±0.018)	**0.038 (±0.024)**
DERC	0.063 (±0.065)	0.056 (±0.012)	**0.033 (±0.017)**
ADX	0.084 (±0.070)	0.054 (±0.009)	**0.030 (±0.001)**
HOICHI	0.115 (±0.066)	0.088 (±0.045)	**0.055 (±0.028)**
SDEX	0.087 (±0.027)	0.083 (±0.068)	**0.042 (±0.019)**
BAG	0.141 (±0.064)	0.109 (±0.045)	**0.075 (±0.027)**
XCN	**0.071 (±0.033)**	0.096 (±0.008)	0.126 (±0.033)
ETH2x-FLI	0.066 (±0.059)	0.052 (±0.007)	**0.033 (±0.020)**
stkAAVE	0.067 (±0.049)	0.063 (±0.008)	**0.042 (±0.013)**
GLM	0.108 (±0.045)	0.088 (±0.014)	**0.075 (±0.017)**
QOM	0.066 (±0.055)	0.055 (±0.004)	**0.039 (±0.020)**
WOJAK	0.116 (±0.068)	0.074 (±0.006)	**0.036 (±0.027)**
DINO	0.085 (±0.066)	0.059 (±0.009)	**0.034 (±0.002)**
Metis	0.082 (±0.056)	0.072 (±0.022)	**0.046 (±0.008)**
REPv2	**0.127 (±0.015)**	0.130 (±0.004)	0.129 (±0.022)
TRAC	0.069 (±0.053)	0.059 (±0.003)	**0.043 (±0.023)**
BEPRO	0.105 (±0.068)	0.064 (±0.005)	**0.033 (±0.018)**
1st-Place Count↑	3	0	**17**
Avg. Rank↓	2.70	2.05	**1.25**
Avg. MAE↓	0.090	0.075	**0.056**

(c) Regression : Edge Count

Dataset	Hydra-8	Hydra-16	Hydra-32
MIR	0.087 (±0.057)	0.035 (±0.027)	**0.016 (±0.004)**
DOGE2.0	**0.115 (±0.023)**	0.135 (±0.032)	0.187 (±0.008)
MUTE	0.092 (±0.036)	0.048 (±0.017)	**0.021 (±0.005)**
EVERMOON	0.085 (±0.053)	0.041 (±0.029)	**0.017 (±0.005)**
DERC	0.082 (±0.049)	0.033 (±0.024)	**0.021 (±0.004)**
ADX	0.082 (±0.047)	0.029 (±0.018)	**0.025 (±0.005)**
HOICHI	0.090 (±0.038)	0.062 (±0.006)	**0.027 (±0.008)**
SDEX	0.080 (±0.021)	**0.077 (±0.018)**	0.095 (±0.006)
BAG	0.099 (±0.023)	0.065 (±0.005)	**0.041 (±0.002)**
XCN	0.066 (±0.056)	**0.027 (±0.009)**	0.062 (±0.002)
ETH2x-FLI	0.084 (±0.034)	0.035 (±0.019)	**0.023 (±0.004)**
stkAAVE	0.065 (±0.037)	**0.025 (±0.004)**	0.051 (±0.017)
GLM	0.106 (±0.013)	0.101 (±0.007)	**0.087 (±0.007)**
QOM	0.074 (±0.054)	**0.032 (±0.019)**	0.036 (±0.001)
WOJAK	0.090 (±0.059)	0.034 (±0.035)	**0.010 (±0.000)**
DINO	0.085 (±0.053)	0.040 (±0.028)	**0.017 (±0.007)**
Metis	0.087 (±0.030)	0.041 (±0.005)	**0.034 (±0.010)**
REPv2	0.127 (±0.024)	**0.105 (±0.003)**	0.111 (±0.004)
TRAC	0.086 (±0.042)	0.029 (±0.020)	**0.021 (±0.005)**
BEPRO	0.094 (±0.067)	0.037 (±0.036)	**0.011 (±0.001)**
1st-Place Count↑	1	5	**14**
Avg. Rank↓	2.85	1.75	**1.40**
Avg. MAE↓	0.089	0.052	**0.046**

7.2.9 Extended Network and Task Results

This section presents the complete per-network results for all classification and regression tasks, following the same evaluation protocol described in the main text. These tables provide a detailed view of model behavior across individual datasets, complementing the summary figures by illustrating the full distribution of baseline and Hydra performance. They allow a closer examination of consistency, generalization, and task-specific trends across diverse temporal networks.

Classification. Extended classification results are reported for *Edge-G/S*, *LCC-G/S*, and *Node-G/S* in Table 7.8, Table 7.9, and Table 7.10, respectively.

- **Table 7.8:** This table presents AUC results for the Edge Growth/Shrinkage classification task. Hydra achieves the strongest overall performance with 8 first-place datasets, the lowest average rank (2.80), and the highest mean AUC (0.753). These consistent improvements demonstrate that Hydra effectively captures temporal edge dynamics and transfers learned interaction patterns across heterogeneous networks, achieving stable performance even on volatile datasets such as *DOGE2.0* and *EVERMOON*.

- **Table 7.9:** This table summarizes the LCC Growth/Shrinkage classification outcomes. Hydra delivers the most consistent performance with 13 first-place results, an average AUC of 0.759, and the best rank (1.70) across all models. This strong generalization suggests that Hydra's shared temporal embeddings effectively model the evolution of large connected components, transferring structural knowledge across domains more reliably than individually trained networks.

- **Table 7.10:** This table reports AUC values for the Node Growth/Shrinkage classification task. Hydra dominates with 15 first-place datasets, achieving an average AUC of 0.734 and the best rank (1.95). The results confirm that Hydra's multi-network training captures universal temporal growth signatures at the node level, yielding robust transfer even across highly diverse graph domains.

These tables compare models trained individually on each dataset with transfer models trained jointly across multiple networks. Hydra achieves the highest AUC on most datasets and consistently ranks among the top two models across all three classification tasks, confirming its strong generalization ability and stability across domains.

Regression. Extended regression results are provided for *Edge Count*, *New Node Count*, and *Influential Node Count* in Table 7.11, Table 7.12, and Table 7.13, respectively.

- **Table 7.11:** This table shows MAE scores for the Edge Count regression task. Without any dataset-specific fine-tuning, Hydra attains the lowest average MAE (0.046) and the best overall rank (3.35) with 6 first-place results. This indicates that Hydra's learned representations generalize quantitative edge evolution across networks, supporting accurate zero-shot predictions of edge volume dynamics.

- **Table 7.13:** This table presents MAE results for the Influential Node Count regression task. Hydra remains competitive with 5 first-place datasets, achieving an average MAE of 0.056 and a rank of 4.90. Although some single models outperform Hydra due to dataset-specific optimization, its comparable results without fine-tuning demonstrate robust transferability, even for complex and domain-sensitive node influence estimation.

- **Table 7.12:** This table reports MAE values for the New Node Count regression task. Hydra achieves the overall best performance with 8 first-place datasets, the lowest mean MAE (0.035), and the best rank (2.50). These results show that Hydra effectively captures cross-domain temporal priors governing node arrivals, achieving strong forecasting accuracy for growth trends across unseen networks.

Hydra achieves the lowest or near-lowest MAE across most datasets, demonstrating effective cross-network transfer in regression settings. Notably, even for the *Influential Node Count* task, where no additional training is performed, Hydra maintains performance comparable to the best single models, highlighting its robustness and ability to generalize in a zero-shot setting.

Table 7.8: AUC results for the Edge Growth or Shrinkage classification task. Best results are in **bold**, second best are underlined. Hydra and MiNT are transfer learning models compared against single models trained separately on each dataset, with Hydra showing superior overall performance.

	Single Model on Individual Networks							Transfer Models	
Dataset	**HTGN**	**GC-LSTM**	**EvolveGCN**	**GraphPulse**	**ROLAND**	**TGCN**	**WinGNN**	**MiNT**	Hydra **(Ours)**
MIR	0.750 ±0.005	0.768 ±0.026	0.745 ±0.015	0.689 ±0.097	0.228 ±0.060	0.749 ±0.026	0.742 ±0.015	**0.836 ±0.016**	0.793 ±0.026
DOGE2.0	0.590 ±0.059	0.538 ±0.000	0.551 ±0.022	0.384 ±0.180	0.513 ±0.022	0.487 ±0.044	0.577 ±0.038	0.538 ±0.038	**0.897 ±0.044**
MUTE	0.649 ±0.015	0.593 ±0.030	0.617 ±0.010	**0.779 ±0.004**	0.289 ±0.042	0.557 ±0.068	0.593 ±0.054	0.673 ±0.013	0.701 ±0.068
EVERMOON	0.512 ±0.023	0.562 ±0.179	0.451 ±0.046	0.519 ±0.130	0.349 ±0.119	0.463 ±0.149	0.525 ±0.114	0.517 ±0.039	**0.818 ±0.149**
DERC	0.683 ±0.013	0.703 ±0.022	0.669 ±0.009	0.769 ±0.040	0.405 ±0.357	0.743 ±0.077	0.674 ±0.044	0.798 ±0.027	**0.839 ±0.077**
ADX	0.769 ±0.018	0.723 ±0.002	0.718 ±0.004	**0.784 ±0.002**	0.761 ±0.011	0.674 ±0.034	0.733 ±0.023	0.679 ±0.024	0.722 ±0.034
HOICHI	0.807 ±0.047	**0.857 ±0.000**	0.856 ±0.001	0.714 ±0.010	0.815 ±0.036	0.836 ±0.034	0.769 ±0.101	0.765 ±0.018	0.591 ±0.034
SDEX	**0.762 ±0.034**	0.720 ±0.002	0.733 ±0.028	0.436 ±0.030	0.483 ±0.254	0.759 ±0.039	0.726 ±0.000	0.614 ±0.020	0.348 ±0.039
BAG	0.673 ±0.227	0.196 ±0.179	0.329 ±0.040	0.934 ±0.020	0.418 ±0.016	0.334 ±0.171	0.485 ±0.105	0.931 ±0.028	**0.969 ±0.171**
XCN	0.668 ±0.099	0.306 ±0.092	0.512 ±0.067	0.821 ±0.004	0.765 ±0.015	0.703 ±0.037	0.586 ±0.029	**0.851 ±0.043**	0.844 ±0.037
ETH2x-FLI	0.610 ±0.059	0.670 ±0.009	0.688 ±0.010	0.666 ±0.047	0.621 ±0.023	0.647 ±0.020	0.617 ±0.056	**0.729 ±0.015**	0.712 ±0.020
stkAAVE	0.702 ±0.042	0.368 ±0.011	0.397 ±0.022	**0.743 ±0.006**	0.591 ±0.122	0.577 ±0.129	0.572 ±0.018	0.709 ±0.022	0.732 ±0.129
GLM	0.830 ±0.029	0.451 ±0.003	0.501 ±0.033	0.769 ±0.018	0.559 ±0.357	0.531 ±0.008	0.530 ±0.004	0.831 ±0.024	**0.850 ±0.008**
QOM	0.633 ±0.017	0.612 ±0.001	0.618 ±0.002	**0.775 ±0.011**	0.641 ±0.003	0.647 ±0.032	0.645 ±0.099	0.647 ±0.019	0.745 ±0.032
WOJAK	0.479 ±0.005	0.484 ±0.000	0.505 ±0.023	0.467 ±0.030	0.529 ±0.005	0.516 ±0.021	0.511 ±0.026	0.524 ±0.027	**0.585 ±0.021**
DINO	0.730 ±0.195	0.874 ±0.028	0.868 ±0.029	0.801 ±0.020	0.497 ±0.092	0.544 ±0.314	0.628 ±0.251	0.779 ±0.113	**0.895 ±0.314**
Metis	0.715 ±0.122	0.646 ±0.023	0.688 ±0.027	**0.812 ±0.011**	0.696 ±0.108	0.709 ±0.033	0.690 ±0.039	0.760 ±0.025	0.733 ±0.033
REPv2	0.760 ±0.012	0.725 ±0.014	0.709 ±0.002	**0.830 ±0.001**	0.751 ±0.003	0.696 ±0.035	0.744 ±0.026	0.789 ±0.020	0.772 ±0.035
TRAC	0.712 ±0.071	0.748 ±0.000	0.748 ±0.000	0.767 ±0.001	0.495 ±0.223	0.741 ±0.012	0.752 ±0.007	**0.785 ±0.008**	0.722 ±0.012
BEPRO	0.655 ±0.038	0.632 ±0.019	0.610 ±0.012	0.783 ±0.003	0.439 ±0.125	0.744 ±0.074	0.736 ±0.018	0.782 ±0.003	**0.800 ±0.074**
1st-Place Count↑	1	1	0	6	0	0	0	4	**8**
Avg. Rank ↓	4.85	5.80	6.10	3.80	6.30	5.60	5.55	3.30	**2.80**
Avg. AUC ↑	0.684	0.609	0.626	0.712	0.542	0.633	0.642	0.727	**0.753**

Table 7.9: AUC results for the LCC Growth or Shrinkage classification task. Best results are in **bold**, second best are underlined. **Hydra** and **MiNT** represent transfer learning models evaluated against single models trained on individual datasets, with **Hydra** achieving the strongest overall results.

	Single Model on Individual Networks							Transfer Models	
Dataset	**HTGN**	**GC-LSTM**	**EvolveGCN**	**GraphPulse**	**ROLAND**	**TGCN**	**WinGNN**	**MiNT**	Hydra **(Ours)**
MIR	0.745 (±0.023)	0.585 (±0.128)	0.575 (±0.146)	0.800 (±0.008)	0.536 (±0.275)	0.585 (±0.055)	0.749 (±0.020)	**0.845 (±0.035)**	0.815 (±0.007)
DOGE2.0	0.446 (±0.164)	0.387 (±0.294)	0.583 (±0.115)	0.333 (±0.042)	0.411 (±0.232)	0.464 (±0.182)	0.595 (±0.176)	0.661 (±0.047)	**0.702 (±0.160)**
MUTE	0.574 (±0.022)	0.579 (±0.022)	0.578 (±0.033)	0.647 (±0.014)	0.624 (±0.037)	0.567 (±0.007)	0.641 (±0.061)	0.582 (±0.078)	**0.704 (±0.085)**
EVERMOON	0.494 (±0.127)	0.512 (±0.112)	0.548 (±0.152)	0.463 (±0.034)	0.491 (±0.157)	0.624 (±0.004)	0.603 (±0.041)	0.527 (±0.118)	**0.667 (±0.045)**
DERC	0.717 (±0.035)	0.591 (±0.010)	0.553 (±0.044)	0.727 (±0.009)	0.481 (±0.131)	0.523 (±0.103)	0.582 (±0.043)	0.689 (±0.096)	**0.808 (±0.022)**
ADX	**0.753 (±0.013)**	0.599 (±0.012)	0.604 (±0.030)	0.661 (±0.006)	0.606 (±0.059)	0.621 (±0.017)	0.611 (±0.062)	0.587 (±0.014)	0.727 (±0.093)
HOICHI	0.746 (±0.010)	0.749 (±0.001)	0.745 (±0.003)	0.730 (±0.017)	0.360 (±0.121)	**0.750 (±0.002)**	0.635 (±0.183)	0.722 (±0.034)	0.638 (±0.075)
SDEX	**0.911 (±0.104)**	0.721 (±0.138)	0.601 (±0.105)	0.808 (±0.050)	0.825 (±0.047)	0.770 (±0.231)	0.575 (±0.282)	0.382 (±0.280)	0.816 (±0.025)
BAG	0.493 (±0.043)	0.291 (±0.180)	0.480 (±0.052)	0.900 (±0.010)	0.463 (±0.019)	0.463 (±0.141)	0.490 (±0.080)	0.893 (±0.074)	**0.976 (±0.006)**
XCN	0.566 (±0.199)	0.481 (±0.160)	0.533 (±0.257)	0.681 (±0.005)	0.569 (±0.204)	0.638 (±0.045)	0.549 (±0.133)	0.827 (±0.025)	**0.887 (±0.014)**
ETH2x-FLI	0.561 (±0.037)	0.529 (±0.017)	0.547 (±0.009)	0.653 (±0.047)	0.499 (±0.135)	0.549 (±0.019)	0.505 (±0.090)	0.618 (±0.025)	**0.687 (±0.032)**
stkAAVE	0.623 (±0.077)	0.581 (±0.085)	0.551 (±0.102)	0.662 (±0.004)	0.532 (±0.140)	0.543 (±0.102)	0.489 (±0.105)	0.688 (±0.019)	**0.748 (±0.012)**
GLM	0.761 (±0.031)	0.481 (±0.073)	0.636 (±0.123)	0.749 (±0.014)	0.802 (±0.037)	0.425 (±0.005)	0.489 (±0.079)	0.818 (±0.074)	**0.848 (±0.012)**
QOM	0.658 (±0.150)	0.509 (±0.100)	0.562 (±0.022)	**0.747 (±0.006)**	0.627 (±0.134)	0.419 (±0.044)	0.546 (±0.152)	0.645 (±0.109)	0.729 (±0.010)
WOJAK	0.378 (±0.028)	0.489 (±0.133)	0.394 (±0.079)	**0.550 (±0.036)**	0.360 (±0.005)	0.481 (±0.092)	0.415 (±0.017)	0.492 (±0.107)	0.500 (±0.083)
DINO	0.706 (±0.120)	0.796 (±0.023)	0.710 (±0.034)	0.661 (±0.026)	0.523 (±0.238)	0.773 (±0.043)	0.731 (±0.037)	0.561 (±0.006)	**0.818 (±0.049)**
Metis	0.679 (±0.039)	0.687 (±0.018)	0.672 (±0.016)	**0.783 (±0.007)**	0.672 (±0.103)	0.657 (±0.014)	0.634 (±0.042)	0.780 (±0.041)	0.731 (±0.001)
REPv2	0.730 (±0.007)	0.653 (±0.015)	0.644 (±0.027)	0.752 (±0.001)	0.658 (±0.103)	0.646 (±0.025)	0.683 (±0.014)	0.742 (±0.041)	**0.764 (±0.020)**
TRAC	0.733 (±0.009)	0.629 (±0.005)	0.623 (±0.004)	0.686 (±0.001)	0.606 (±0.117)	0.620 (±0.005)	0.599 (±0.026)	0.762 (±0.028)	**0.781 (±0.004)**
BEPRO	0.694 (±0.009)	0.595 (±0.008)	0.557 (±0.058)	0.725 (±0.004)	0.482 (±0.146)	0.536 (±0.031)	0.582 (±0.063)	0.628 (±0.017)	**0.830 (±0.006)**
1st-Place Count↑	2	0	0	3	0	1	0	1	**13**
Avg. Rank ↓	4.40	6.00	6.30	3.50	6.95	6.10	6.10	3.95	**1.70**
Avg. AUC ↑	0.648	0.572	0.585	0.686	0.556	0.583	0.585	0.672	**0.759**

Table 7.10: AUC results for the Node Growth or Shrinkage classification task. Best results are in **bold**, second best are underlined. **Hydra** and **MiNT** are transfer learning models compared with single models trained on individual datasets, where **Hydra** demonstrates superior overall results.

	Single Model on Individual Networks							Transfer Models	
Dataset	**HTGN**	**GC-LSTM**	**EvolveGCN**	**GraphPulse**	**ROLAND**	**TGCN**	**WinGNN**	**MiNT**	Hydra **(Ours)**
MIR	0.545 (±0.030)	0.537 (±0.033)	0.528 (±0.081)	0.633 (±0.066)	0.472 (±0.040)	0.532 (±0.021)	0.525 (±0.043)	0.622 (±0.030)	**0.764 (±0.001)**
DOGE2.0	0.427 (±0.065)	0.633 (±0.014)	0.400 (±0.061)	0.403 (±0.035)	0.260 (±0.000)	0.627 (±0.034)	0.693 (±0.041)	**0.750 (±0.014)**	0.633 (±0.064)
MUTE	0.518 (±0.023)	0.448 (±0.008)	0.475 (±0.051)	0.677 (±0.008)	0.411 (±0.053)	0.458 (±0.009)	0.562 (±0.015)	0.606 (±0.056)	**0.748 (±0.025)**
EVERMOON	0.585 (±0.059)	0.606 (±0.021)	0.488 (±0.088)	0.463 (±0.002)	0.427 (±0.137)	0.567 (±0.030)	0.548 (±0.116)	0.614 (±0.071)	**0.655 (±0.040)**
DERC	0.662 (±0.051)	0.492 (±0.069)	0.503 (±0.084)	0.611 (±0.049)	0.551 (±0.013)	0.447 (±0.003)	0.517 (±0.034)	0.569 (±0.004)	**0.742 (±0.004)**
ADX	0.678 (±0.017)	0.505 (±0.043)	0.509 (±0.022)	0.701 (±0.003)	0.557 (±0.082)	0.484 (±0.048)	0.504 (±0.018)	0.507 (±0.037)	**0.718 (±0.051)**
HOICHI	0.687 (±0.004)	0.718 (±0.007)	0.685 (±0.020)	**0.745 (±0.006)**	0.347 (±0.084)	0.718 (±0.002)	0.526 (±0.188)	0.492 (±0.120)	0.558 (±0.047)
SDEX	0.824 (±0.106)	0.364 (±0.148)	0.817 (±0.032)	**0.865 (±0.011)**	0.779 (±0.018)	0.755 (±0.202)	0.757 (±0.072)	0.861 (±0.025)	0.743 (±0.026)
BAG	0.735 (±0.075)	0.337 (±0.089)	0.166 (±0.066)	0.897 (±0.016)	0.390 (±0.088)	0.391 (±0.219)	0.515 (±0.008)	0.685 (±0.038)	**0.969 (±0.009)**
XCN	0.476 (±0.012)	0.466 (±0.012)	0.407 (±0.176)	0.671 (±0.020)	0.430 (±0.144)	0.483 (±0.036)	0.355 (±0.017)	0.505 (±0.002)	**0.878 (±0.009)**
ETH2x-FLI	0.628 (±0.022)	0.548 (±0.001)	0.548 (±0.002)	0.615 (±0.020)	0.488 (±0.063)	0.553 (±0.036)	0.586 (±0.098)	0.411 (±0.066)	**0.678 (±0.031)**
stkAAVE	0.517 (±0.093)	0.543 (±0.043)	0.456 (±0.069)	0.643 (±0.005)	0.661 (±0.037)	0.425 (±0.029)	0.465 (±0.036)	0.561 (±0.007)	**0.779 (±0.007)**
GLM	0.706 (±0.014)	0.566 (±0.001)	0.516 (±0.105)	0.595 (±0.003)	0.493 (±0.149)	0.575 (±0.019)	0.610 (±0.024)	0.720 (±0.045)	**0.763 (±0.005)**
QOM	0.647 (±0.094)	0.492 (±0.003)	0.485 (±0.001)	0.705 (±0.002)	0.592 (±0.080)	0.495 (±0.004)	0.409 (±0.051)	0.572 (±0.017)	**0.719 (±0.012)**
WOJAK	0.417 (±0.143)	0.338 (±0.068)	0.357 (±0.104)	0.500 (±0.000)	0.202 (±0.018)	0.488 (±0.080)	0.314 (±0.029)	**0.618 (±0.035)**	0.412 (±0.060)
DINO	0.845 (±0.015)	0.323 (±0.148)	0.444 (±0.052)	0.686 (±0.007)	0.330 (±0.115)	0.615 (±0.070)	0.600 (±0.292)	0.735 (±0.005)	**0.910 (±0.013)**
Metis	0.589 (±0.049)	0.483 (±0.052)	0.566 (±0.012)	0.652 (±0.029)	0.574 (±0.040)	0.549 (±0.011)	0.510 (±0.005)	0.616 (±0.012)	**0.693 (±0.007)**
REPv2	0.650 (±0.004)	0.519 (±0.023)	0.515 (±0.019)	0.662 (±0.008)	0.597 (±0.028)	0.534 (±0.029)	0.626 (±0.058)	**0.710 (±0.129)**	0.689 (±0.013)
TRAC	0.670 (±0.031)	0.527 (±0.016)	0.524 (±0.003)	0.610 (±0.055)	0.546 (±0.024)	0.524 (±0.001)	0.528 (±0.020)	0.600 (±0.027)	**0.765 (±0.003)**
BEPRO	0.500 (±0.055)	0.332 (±0.010)	0.356 (±0.015)	0.707 (±0.005)	0.490 (±0.016)	0.372 (±0.100)	0.420 (±0.018)	0.561 (±0.022)	**0.865 (±0.010)**
1st-Place Count↑	0	0	0	2	0	0	0	3	**15**
Avg. Rank↓	3.65	6.65	7.00	2.95	6.70	6.15	6.20	3.55	**1.95**
Avg. AUC↑	0.615	0.489	0.487	0.652	0.480	0.530	0.528	0.616	**0.734**

Table 7.11: MAE results for the Edge Count regression task. Best results are in **bold**, second best are underlined. **Hydra** is the only transfer learning model and has better overall performance without any training on these datasets.

	Single Model on Individual Networks							Transfer Models
Dataset	**HTGN**	**TGCN**	**GCLSTM**	**ROLAND**	**EGCN**	**GraphPulse**	**WinGNN**	Hydra **(Ours)**
MIR	0.059 (±0.007)	0.044 (±0.009)	0.047 (±0.003)	0.039 (±0.000)	0.057 (±0.015)	0.059 (±0.001)	0.046 (±0.010)	**0.016 (±0.005)**
DOGE2.0	0.101 (±0.035)	0.063 (±0.017)	0.106 (±0.029)	0.052 (±0.003)	0.092 (±0.031)	0.046 (±0.000)	**0.045 (±0.003)**	0.187 (±0.008)
MUTE	0.025 (±0.006)	0.038 (±0.004)	**0.017 (±0.001)**	0.040 (±0.007)	0.049 (±0.005)	0.025 (±0.002)	0.027 (±0.003)	0.021 (±0.022)
EVERMOON	**0.010 (±0.001)**	0.021 (±0.013)	0.025 (±0.007)	0.016 (±0.016)	0.030 (±0.010)	0.235 (±0.005)	0.025 (±0.004)	0.017 (±0.000)
DERC	0.038 (±0.015)	0.059 (±0.011)	**0.016 (±0.005)**	0.060 (±0.008)	0.023 (±0.007)	0.023 (±0.003)	0.032 (±0.001)	0.021 (±0.003)
ADX	0.017 (±0.001)	0.018 (±0.003)	**0.016 (±0.001)**	0.017 (±0.002)	0.021 (±0.002)	0.019 (±0.001)	**0.016 (±0.000)**	0.025 (±0.008)
HOICHI	0.034 (±0.010)	**0.020 (±0.001)**	0.046 (±0.013)	**0.020 (±0.003)**	0.034 (±0.013)	0.044 (±0.002)	0.028 (±0.005)	0.027 (±0.020)
SDEX	0.080 (±0.029)	0.128 (±0.046)	**0.058 (±0.007)**	0.121 (±0.002)	0.085 (±0.025)	0.106 (±0.005)	0.128 (±0.008)	0.095 (±0.043)
BAG	**0.022 (±0.003)**	0.023 (±0.017)	0.025 (±0.002)	0.030 (±0.016)	0.027 (±0.005)	0.063 (±0.001)	0.260 (±0.064)	0.041 (±0.020)
XCN	0.074 (±0.006)	0.112 (±0.042)	0.107 (±0.024)	0.120 (±0.035)	0.121 (±0.011)	0.118 (±0.000)	0.072 (±0.018)	**0.062 (±0.015)**
ETH2x-FLI	0.055 (±0.015)	0.079 (±0.019)	0.177 (±0.057)	0.040 (±0.026)	0.066 (±0.016)	0.144 (±0.009)	0.030 (±0.011)	**0.023 (±0.001)**
stkAAVE	0.083 (±0.008)	0.087 (±0.005)	0.092 (±0.017)	0.104 (±0.012)	0.079 (±0.010)	0.096 (±0.006)	0.100 (±0.004)	**0.051 (±0.022)**
GLM	0.072 (±0.010)	0.058 (±0.005)	0.063 (±0.002)	0.058 (±0.003)	0.060 (±0.001)	0.076 (±0.001)	**0.054 (±0.003)**	0.087 (±0.008)
QOM	0.042 (±0.005)	0.085 (±0.020)	0.053 (±0.030)	0.057 (±0.030)	0.069 (±0.013)	0.055 (±0.001)	0.046 (±0.006)	**0.036 (±0.013)**
WOJAK	0.009 (±0.002)	0.012 (±0.003)	0.013 (±0.004)	0.016 (±0.008)	0.013 (±0.007)	0.057 (±0.008)	**0.006 (±0.002)**	0.010 (±0.004)
DINO	0.069 (±0.020)	0.025 (±0.009)	0.040 (±0.019)	**0.014 (±0.004)**	0.039 (±0.011)	0.087 (±0.002)	0.021 (±0.008)	0.017 (±0.007)
Metis	0.038 (±0.002)	0.054 (±0.001)	0.047 (±0.003)	0.057 (±0.008)	0.053 (±0.004)	0.066 (±0.006)	0.043 (±0.001)	**0.034 (±0.010)**
REPv2	0.117 (±0.013)	0.108 (±0.004)	0.115 (±0.004)	**0.106 (±0.001)**	0.128 (±0.036)	0.119 (±0.001)	0.118 (±0.001)	0.111 (±0.008)
TRAC	0.026 (±0.004)	0.036 (±0.010)	0.061 (±0.006)	0.023 (±0.004)	0.036 (±0.003)	**0.017 (±0.000)**	0.040 (±0.014)	0.021 (±0.001)
BEPRO	0.009 (±0.001)	0.009 (±0.003)	0.009 (±0.002)	0.015 (±0.017)	**0.007 (±0.001)**	**0.007 (±0.000)**	**0.007 (±0.002)**	0.011 (±0.008)
1st-Place Count↑	2	1	4	3	1	2	5	**6**
Avg. Rank ↓	3.95	4.60	4.58	4.58	5.30	5.72	3.93	**3.35**
Avg. MAE ↓	0.049	0.054	0.057	0.050	0.054	0.073	0.057	**0.046**

Table 7.12: MAE results for the New Node Count regression task. Best results are in **bold**, second best are underlined. **Hydra**, the only transfer learning model, achieves better overall performance than all single models trained on individual datasets.

	Single Model on Individual Networks							Transfer Models
Dataset	**HTGN**	**TGCN**	**GCLSTM**	**ROLAND**	**EGCN**	**GraphPulse**	**WinGNN**	Hydra **(ours)**
MIR	0.031 (±0.002)	0.028 (±0.001)	0.039 (±0.005)	0.025 (±0.018)	0.030 (±0.001)	0.025 (±0.001)	0.037 (±0.011)	**0.013 (±0.004)**
DOGE2.0	0.046 (±0.009)	0.064 (±0.021)	**0.044 (±0.014)**	0.073 (±0.019)	0.051 (±0.031)	0.157 (±0.010)	0.098 (±0.009)	0.092 (±0.008)
MUTE	**0.021 (±0.003)**	0.041 (±0.005)	0.030 (±0.003)	0.048 (±0.002)	0.053 (±0.003)	0.031 (±0.001)	0.051 (±0.015)	0.025 (±0.005)
EVERMOON	**0.010 (±0.004)**	0.017 (±0.008)	0.026 (±0.009)	0.029 (±0.019)	0.028 (±0.017)	0.222 (±0.005)	0.022 (±0.002)	0.012 (±0.005)
DERC	0.028 (±0.009)	0.034 (±0.018)	0.016 (±0.024)	0.043 (±0.024)	0.019 (±0.004)	0.043 (±0.011)	0.027 (±0.003)	**0.015 (±0.004)**
ADX	0.014 (±0.001)	**0.010 (±0.001)**	0.011 (±0.001)	0.024 (±0.296)	0.012 (±0.000)	0.024 (±0.002)	0.011 (±0.000)	0.016 (±0.005)
HOICHI	0.044 (±0.004)	0.035 (±0.025)	0.053 (±0.028)	0.204 (±0.028)	0.039 (±0.023)	0.066 (±0.001)	**0.027 (±0.004)**	0.029 (±0.008)
SDEX	0.075 (±0.002)	0.093 (±0.006)	0.069 (±0.002)	0.088 (±0.002)	0.077 (±0.008)	0.087 (±0.002)	0.090 (±0.025)	**0.063 (±0.006)**
BAG	0.023 (±0.007)	0.031 (±0.008)	**0.019 (±0.007)**	0.053 (±0.006)	0.030 (±0.004)	0.054 (±0.000)	0.230 (±0.062)	0.052 (±0.002)
XCN	0.014 (±0.007)	0.015 (±0.006)	0.017 (±0.007)	0.012 (±0.012)	0.017 (±0.007)	0.040 (±0.001)	0.015 (±0.002)	**0.009 (±0.002)**
ETH2x-FLI	0.031 (±0.010)	0.041 (±0.001)	0.069 (±0.010)	**0.020 (±0.004)**	0.030 (±0.002)	0.092 (±0.001)	0.028 (±0.003)	0.030 (±0.004)
stkAAVE	0.128 (±0.018)	0.136 (±0.008)	0.128 (±0.018)	0.154 (±0.005)	0.124 (±0.005)	0.151 (±0.000)	0.147 (±0.008)	**0.078 (±0.017)**
GLM	**0.066 (±0.000)**	0.068 (±0.001)	0.068 (±0.000)	0.068 (±0.002)	0.067 (±0.002)	0.092 (±0.000)	**0.066 (±0.000)**	0.094 (±0.007)
QOM	0.035 (±0.006)	0.038 (±0.010)	0.032 (±0.020)	0.018 (±0.010)	0.035 (±0.020)	0.033 (±0.004)	0.029 (±0.007)	**0.012 (±0.001)**
WOJAK	0.008 (±0.001)	0.009 (±0.001)	0.015 (±0.003)	0.029 (±0.017)	0.014 (±0.003)	0.067 (±0.005)	**0.007 (±0.001)**	0.009 (±0.000)
DINO	0.061 (±0.017)	0.024 (±0.002)	0.028 (±0.019)	**0.013 (±0.005)**	0.030 (±0.005)	0.085 (±0.001)	0.051 (±0.029)	0.018 (±0.007)
Metis	**0.034 (±0.010)**	0.045 (±0.006)	0.041 (±0.006)	0.043 (±0.005)	0.042 (±0.004)	0.054 (±0.001)	**0.034 (±0.002)**	**0.034 (±0.010)**
REPv2	0.061 (±0.003)	0.061 (±0.004)	0.075 (±0.003)	**0.055 (±0.002)**	0.063 (±0.004)	0.068 (±0.000)	0.063 (±0.000)	0.066 (±0.004)
TRAC	0.043 (±0.021)	0.030 (±0.003)	0.071 (±0.021)	0.021 (±0.006)	0.025 (±0.009)	**0.018 (±0.000)**	0.044 (±0.010)	0.022 (±0.005)
BEPRO	0.012 (±0.002)	0.010 (±0.002)	0.011 (±0.017)	0.011 (±0.011)	0.009 (±0.000)	0.012 (±0.000)	0.006 (±0.001)	**0.004 (±0.001)**
1st-Place Count↑	4	1	2	3	0	1	4	**8**
Avg. Rank ↓	3.45	4.10	4.00	4.35	3.90	5.75	4.05	**2.50**
Avg. MAE ↓	0.039	0.042	0.043	0.052	0.040	0.071	0.054	**0.035**

Table 7.13: MAE results for the Influential Node Count regression task. Best results are in **bold**, second best are underlined. **Hydra**, the only transfer learning model, achieves performance comparable to single models trained individually on each dataset, despite requiring no additional training.

	Single Model on Individual Networks							Transfer Models
Dataset	**HTGN**	**TGCN**	**GCLSTM**	**ROLAND**	**EGCN**	**GraphPulse**	**WinGNN**	Hydra **(ours)**
MIR	0.114 (±0.003)	0.119 (±0.005)	0.105 (±0.020)	0.115 (±0.020)	0.114 (±0.017)	0.127 (±0.002)	0.082 (±0.007)	**0.039 (±0.005)**
DOGE2.0	0.064 (±0.031)	0.090 (±0.023)	**0.053 (±0.021)**	0.070 (±0.015)	0.059 (±0.021)	0.087 (±0.004)	0.091 (±0.009)	0.126 (±0.032)
MUTE	0.028 (±0.004)	0.042 (±0.002)	**0.018 (±0.002)**	0.051 (±0.006)	0.042 (±0.012)	0.021 (±0.001)	0.045 (±0.033)	0.045 (±0.027)
EVERMOON	**0.011 (±0.002)**	0.014 (±0.006)	0.018 (±0.008)	0.014 (±0.004)	0.032 (±0.015)	0.235 (±0.004)	0.026 (±0.006)	0.038 (±0.024)
DERC	0.069 (±0.007)	0.084 (±0.001)	0.053 (±0.002)	0.104 (±0.011)	0.058 (±0.010)	0.048 (±0.001)	0.077 (±0.003)	**0.033 (±0.017)**
ADX	0.016 (±0.003)	0.015 (±0.000)	**0.012 (±0.001)**	0.022 (±0.009)	0.015 (±0.001)	0.020 (±0.001)	0.015 (±0.004)	0.030 (±0.001)
HOICHI	0.039 (±0.011)	**0.020 (±0.007)**	0.046 (±0.017)	0.030 (±0.015)	0.034 (±0.016)	0.047 (±0.009)	0.024 (±0.016)	0.055 (±0.028)
SDEX	0.037 (±0.011)	0.049 (±0.022)	**0.031 (±0.012)**	0.058 (±0.017)	0.037 (±0.005)	0.067 (±0.009)	0.066 (±0.025)	0.042 (±0.019)
BAG	**0.009 (±0.000)**	0.017 (±0.002)	0.030 (±0.008)	0.019 (±0.010)	0.030 (±0.009)	0.064 (±0.001)	0.074 (±0.094)	0.075 (±0.027)
XCN	0.061 (±0.008)	0.155 (±0.009)	**0.055 (±0.003)**	0.124 (±0.005)	0.066 (±0.008)	0.080 (±0.004)	0.159 (±0.029)	0.126 (±0.033)
ETH2x-FLI	0.050 (±0.007)	0.053 (±0.035)	0.100 (±0.014)	**0.023 (±0.005)**	0.044 (±0.010)	0.108 (±0.003)	0.033 (±0.010)	0.033 (±0.020)
stkAAVE	0.062 (±0.001)	0.080 (±0.009)	0.062 (±0.001)	0.057 (±0.005)	0.061 (±0.006)	0.108 (±0.009)	0.066 (±0.003)	**0.042 (±0.013)**
GLM	0.066 (±0.002)	0.057 (±0.001)	0.067 (±0.004)	0.057 (±0.005)	0.054 (±0.002)	0.082 (±0.000)	**0.053 (±0.002)**	0.075 (±0.017)
QOM	0.060 (±0.004)	0.088 (±0.040)	0.061 (±0.032)	0.074 (±0.018)	0.075 (±0.015)	0.076 (±0.004)	0.062 (±0.012)	**0.039 (±0.020)**
WOJAK	**0.007 (±0.001)**	0.008 (±0.003)	0.012 (±0.003)	0.027 (±0.010)	0.011 (±0.004)	0.066 (±0.019)	**0.007 (±0.001)**	0.036 (±0.027)
DINO	0.078 (±0.036)	0.024 (±0.004)	0.068 (±0.001)	0.024 (±0.003)	0.036 (±0.007)	0.087 (±0.003)	**0.022 (±0.000)**	0.034 (±0.002)
Metis	0.056 (±0.003)	0.078 (±0.004)	0.068 (±0.005)	0.085 (±0.030)	0.073 (±0.004)	0.071 (±0.010)	0.063 (±0.001)	**0.046 (±0.008)**
REPv2	**0.115 (±0.000)**	0.118 (±0.006)	0.119 (±0.002)	**0.115 (±0.001)**	0.138 (±0.027)	0.127 (±0.000)	0.126 (±0.001)	0.129 (±0.022)
TRAC	0.045 (±0.005)	0.058 (±0.002)	0.079 (±0.002)	0.055 (±0.009)	0.056 (±0.005)	**0.033 (±0.005)**	0.074 (±0.009)	0.043 (±0.023)
BEPRO	0.016 (±0.006)	0.014 (±0.003)	0.012 (±0.002)	0.013 (±0.001)	0.012 (±0.002)	0.012 (±0.000)	**0.010 (±0.001)**	0.033 (±0.018)
1st-Place Count↑	4	1	5	2	0	1	4	**5**
Avg. Rank ↓	**3.52**	4.85	3.80	4.55	4.20	5.90	4.28	4.90
Avg. MAE ↓	**0.050**	0.059	0.053	0.057	0.052	0.078	0.059	0.056

7.3 Conclusion

We have introduced Hydra, a novel temporal graph-level multi-task and multi-network model designed to address the complexities of dynamic network analysis. Hydra combines a spatial path, which captures local connectivity through temporal GNNs, along with a spectral path that attends to global structural patterns, forming a new architectural design for temporal graph learning. This unified framework enables Hydra to handle diverse prediction tasks simultaneously while transferring effectively to unseen networks. Empirically, Hydra consistently outperforms state-of-the-art baselines across benchmarks, achieving strong performance without requiring additional training on target networks. These results highlight Hydra as the first architecture to bring together spectral and spatial pathways for transferable, multi-task learning on temporal graphs. Looking ahead, this work opens up a promising research direction toward more generalizable temporal models that can support a wide range of tasks.

Chapter 8

Conclusion and Future Directions

This thesis presented a continuous progression of ideas, methods, and findings that together form a unified exploration of how to learn from dynamic and evolving networks. The work began with fundamental questions about how structure and topology influence model efficiency, then addressed the scarcity of standardized temporal graph data, introduced principled frameworks for modeling time-dependent graph evolution, and ultimately advanced to scalable, transferable temporal models that generalize across networks and tasks. Each stage built upon the previous one, shaping a coherent scientific trajectory toward the vision of temporal graph foundation models.

The first contribution, *Topological Forest*, investigated how topological insights can improve learning efficiency and interpretability. By embedding decision trees into a topological space using the Mapper algorithm, this model selected representative structures that preserved diversity and predictive power while greatly reducing computational complexity. Beyond the technical gain, it demonstrated that topology can act as a guiding principle in understanding and organizing complex learning systems, laying the conceptual foundation for structure-aware modeling.

Building on this foundation, *Chartalist* addressed the critical lack of large-scale temporal graph datasets. Before its introduction, most studies relied on limited or synthetic data that failed to capture the evolving nature of real-world systems. Chartalist transformed blockchain transactions into a rich collection of machine-learning-ready temporal graphs that reflect both account-based and

UTXO-based systems. This effort provided the first open benchmark for large-scale temporal network analysis, creating a lasting resource that enables reproducibility, scalability, and standardized evaluation across models and domains.

The third contribution, *GraphPulse*, shifted the focus from static graph structures to temporal evolution. It introduced a framework that models networks as sequences of evolving snapshots and learns their global behaviors through topological and structural summaries. By integrating topological data analysis with temporal modeling, GraphPulse captured macro-level changes that traditional node or edge-based models often overlook. This approach successfully predicted global graph properties such as growth, shrinkage, and connectivity variation, establishing topology as a bridge between structure and time in graph learning.

The next step, *MiNT*, expanded the scope from single-network understanding to learning across multiple networks. MiNT introduced a large-scale benchmark of temporal transaction networks and a new training paradigm that alternates across independent graphs while maintaining temporal consistency. This framework demonstrated that temporal graph models could learn transferable representations, generalizing to unseen networks without retraining. MiNT revealed the first scaling trends in temporal graph learning, showing that performance improves with the diversity and number of training networks, thus setting the foundation for studying transferability and neural scaling in temporal graphs.

The final contribution, *Hydra*, advanced this foundation by addressing the need for efficiency and adaptability in temporal graph learning. While MiNT required separate retraining for each predictive task and involved high computational cost, Hydra introduced a unified model that performs multiple prediction objectives, including both classification and regression, within the same framework. It achieved up to twenty-two times faster training and inference while maintaining strong generalization across diverse networks. This contribution moves beyond transferability by enabling efficient multi-task learning and represents a practical step toward adaptable temporal graph foundation models.

Taken together, these contributions form a continuous and interconnected research path, from

using topology to guide efficient learning, to constructing standardized data foundations, to modeling temporal evolution, transferability, and finally adaptability across networks and tasks. This body of work collectively defines the conceptual and methodological groundwork for the emerging field of temporal graph foundation models. These studies contribute a unified research trajectory that enable systematic and reproducible progress in temporal graph learning. By uniting efficiency, scalability, transferability, and adaptability within a single framework, this thesis lays the groundwork for the next generation of graph learning systems that can understand, predict, and adapt to how complex networks evolve over time.

Research Outlook and Future Directions

The research presented in this thesis establishes a comprehensive foundation for temporal graph learning and opens new frontiers in understanding and modeling the evolution of complex dynamic networks. Through the development of Topological Forest, Chartalist, GraphPulse, MiNT, and Hydra, this work demonstrates a complete progression from large-scale data creation and topological representation to multi-task, multi-network modeling and generalizable temporal reasoning. Together, these contributions define a new research direction toward universal and transferable temporal graph frameworks. Building on this foundation, several promising directions naturally emerge. The first is the extension of temporal graph learning beyond social and financial domains to broader contexts such as biological, transportation, communication, and cyber-physical systems. Applying these frameworks across such diverse domains will further consolidate the concept of domain-agnostic temporal graph models and provide new insights into how structural and temporal dynamics interact in real-world systems. Another direction involves advancing modeling architectures that integrate the principles established in this thesis with emerging paradigms in sequence modeling and representation learning. Incorporating elements such as state-space formulations, attention-based temporal mechanisms, or continuous-time graph processing can deepen the understanding of long-range temporal dependencies while preserving the interpretability and

transferability central to this work. Expanding toward scalability and real-time adaptation also represents a natural continuation of this research. As temporal networks grow in both size and complexity, integrating adaptive and distributed components into temporal graph pipelines will enable continuous learning and efficient deployment across large, evolving systems. These developments will further align temporal graph learning with real-world industrial and scientific applications. The trajectory culminating in Hydra also provides the conceptual and methodological foundation for the emergence of temporal graph foundation models. Such models, trained on large and diverse temporal networks, will capture shared structural invariants and temporal rhythms across domains, enabling seamless transfer between tasks such as forecasting, anomaly detection, and relational reasoning. The datasets, architectures, and evaluation pipelines introduced in this thesis provide the essential building blocks for realizing this next stage. In conclusion, this thesis defines a comprehensive and forward-looking research agenda that lays the groundwork for the next generation of temporal graph models. It opens a new paradigm for learning across time and domains, setting the stage for temporal graph foundation models that can reason, adapt, and generalize across the interconnected systems that define our dynamic world.

Bibliography

[1] Kartik Sharma, Yeon-Chang Lee, Sivagami Nambi, Aditya Salian, Shlok Shah, Sang-Wook Kim, and Srijan Kumar. A survey of graph neural networks for social recommender systems. *ACM Computing Surveys*, 56(10):1–34, 2024.

[2] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019.

[3] Marco Bardoscia, Paolo Barucca, Stefano Battiston, Fabio Caccioli, Giulio Cimini, Diego Garlaschelli, Fabio Saracco, Tiziano Squartini, and Guido Caldarelli. The physics of financial networks. *Nature Reviews Physics*, 3(7):490–507, 2021.

[4] Fabian Schär. Decentralized finance: On blockchain-and smart contract-based financial markets. *FRB of St. Louis Review*, 2021.

[5] A Longa, V Lachi, G Santin, M Bianchini, B Lepri, P Liò, F Scarselli, A Passerini, et al. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *TRANSACTIONS ON MACHINE LEARNING RESEARCH*, 2023.

[6] Guangyin Jin, Yuxuan Liang, Yuchen Fang, Zezhi Shao, Jincai Huang, Junbo Zhang, and Yu Zheng. Spatio-temporal graph neural networks for predictive learning in urban computing: A survey. *IEEE transactions on knowledge and data engineering*, 36(10):5388–5408, 2023.

[7] Arijit Khan. Graph analysis of the ethereum blockchain data: A survey of datasets, methods, and future work. In *2022 IEEE International Conference on Blockchain*, pages 250–257, 2022.

[8] Zewen Liu, Guancheng Wan, B Aditya Prakash, Max SY Lau, and Wei Jin. A review of graph neural networks in epidemic modeling. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6577–6587, 2024.

[9] Xunlian Luo, Chunjiang Zhu, Detian Zhang, and Qing Li. Stg4traffic: A survey and benchmark of spatial-temporal graph neural networks for traffic prediction. *arXiv preprint arXiv:2307.00495*, 2023.

[10] Xiang Bian, Yu Xie, Chao Zhang, Yitong Liu, Jure Zhou, Hongwei Ma, and Shaoyang Zheng. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *ACM Computing Surveys (CSUR)*, 56(2):1–40, 2023.

[11] ZhengZhao Feng, Rui Wang, TianXing Wang, Mingli Song, Sai Wu, and Shuibing He. A comprehensive survey of dynamic graph neural networks: Models, frameworks, benchmarks, experiments and challenges. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[12] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[13] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[14] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael M. Bronstein, Guillaume Rabusseau, and Rei-

haneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *NeurIPS 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[15] Murat Ali Bayir, Kiarash Shamsi, Huseyincan Kaynak, and Cuneyt Gurcan Akcora. Topological forest. *IEEE Access*, 10:131711–131721, 2022.

[16] Kiarash Shamsi, Friedhelm Victor, Murat Kantarcioglu, Yulia Gel, and Cuneyt G Akcora. Chartalist: Labeled graph datasets for utxo and account-based blockchains. *Advances in Neural Information Processing Systems*, 35:34926–34939, 2022.

[17] Kiarash Shamsi, Farimah Poursafaei, Shenyang Huang, Bao Tran Gia Ngo, Baris Coskunuzer, and Cuneyt Gurcan Akcora. Graphpulse: Topological representations for temporal graph property prediction. In *The Twelfth International Conference on Learning Representations*, 2024.

[18] Kiarash Shamsi, Tran Gia Bao Ngo, Razieh Shirzadkhani, Shenyang Huang, Farimah Poursafaei, Poupak Azad, Reihaneh Rabbany, Baris Coskunuzer, Guillaume Rabusseau, and Cuneyt Gurcan Akcora. Mint: Multi-network transfer benchmark for temporal graph learning. *NeurIPS*, 2025.

[19] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[20] Herbert Edelsbrunner, John Harer, et al. Persistent homology-a survey. *Contemporary mathematics*, 453(26):257–282, 2008.

[21] Frédéric Chazal and Bertrand Michel. An introduction to topological data analysis: fundamental and practical aspects for data scientists. *Frontiers in artificial intelligence*, 4:667963, 2021.

[22] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 263–271, 2005.

[23] Panagiotis Kyriakis, Iordanis Fostiropoulos, and Paul Bogdan. Learning hyperbolic representations of topological features. In *International Conference on Learning Representations*, 2021.

[24] Edelsbrunner, Letscher, and Zomorodian. Topological persistence and simplification. *Discrete & computational geometry*, 28(4):511–533, 2002.

[25] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18(8):1–35, 2017.

[26] Peter Bubenik et al. Statistical topological data analysis using persistence landscapes. *J. Mach. Learn. Res.*, 16(1):77–102, 2015.

[27] Mathieu Carrière, Frédéric Chazal, Yuichi Ike, Théo Lacombe, Martin Royer, and Yuhei Umeda. Perslay: A neural network layer for persistence diagrams and new graph topological signatures. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 2786–2796. Proceedings of Machine Learning Research, 2020.

[28] Christoph Hofer, Florian Graf, Bastian Rieck, Marc Niethammer, and Roland Kwitt. Graph filtration learning. In *International Conference on Machine Learning*, pages 4314–4323. PMLR, 2020.

[29] Yuzhou Chen, Baris Coskunuzer, and Yulia Gel. Topological relational learning on graphs. *Advance in Neural Information Processing Systems*, 34:27029–27042, 2021.

[30] Max Horn, Edward De Brouwer, Michael Moor, Yves Moreau, Bastian Rieck, and Karsten Borgwardt. Topological graph neural networks. In *International Conference on Learning Representations*, 2021.

[31] Gurjeet Singh, Facundo Mémoli, and Gunnar E Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. In *Eurographics Symposium on Point-Based Graphics*, pages 91–100, 2007.

[32] G. Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2), 2009.

[33] Hendrik Jacob Van Veen, Nathaniel Saul, David Eargle, and Sam W Mangham. Kepler mapper: A flexible python implementation of the mapper algorithm. *Journal of Open Source Software*, 4(42):1315, 2019.

[34] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal M Medina-Mardones, Alberto Dassatti, and Kathryn Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration. *The Journal of Machine Learning Research*, 22(1):1834–1839, 2021.

[35] Monica Nicolau, Arnold J Levine, and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Nature Biotechnology*, 29(6):551–555, 2011.

[36] Pek Y Lum, Gurjeet Singh, Alan Lehman, Tigran Ishkanov, Mikael Vejdemo-Johansson, Muthu Alagappan, John Carlsson, and Gunnar Carlsson. Extracting insights from the shape of complex data using topology. *Scientific reports*, 3(1):1236, 2013.

[37] Yasuaki Hiraoka, Tomoyuki Nakamura, Akihiro Hirata, Ezra Garcia Escolar, Kaname Matsue, and Yasumasa Nishiura. Hierarchical structures of amorphous solids characterized by persistent homology. *Proceedings of the National Academy of Sciences*, 113(26):7035–7040, 2016.

[38] Marian Gidea and Yuri Katz. Topological data analysis of financial time series: Landscapes of crashes. *Physica A: Statistical Mechanics and its Applications*, 491:820–834, 2018.

[39] Giovanni Petri, Paul Expert, Federico Turkheimer, Timoteo Carletti, Rossana Silva, Rebekah Pool, Edward T Bullmore, and Francesco Vaccarino. Homological scaffolds of brain functional networks. *Journal of The Royal Society Interface*, 11(101):20140873, 2013.

[40] Methun Kamruzzaman, Ananth Kalyanaraman, Bala Krishnamoorthy, Stefan Hey, and Patrick S Schnable. Hyppo-x: A scalable exploratory framework for analyzing complex phenomics data. *IEEE/ACM transactions on computational biology and bioinformatics*, 18(4):1535–1548, 2019.

[41] Youjia Zhou, Methun Kamruzzaman, Patrick Schnable, Bala Krishnamoorthy, Ananth Kalyanaraman, and Bei Wang. Pheno-mapper: an interactive toolbox for the visual exploration of phenomics data. In *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 1–10, 2021.

[42] Mustafa Hajij, Bei Wang, and Paul Rosen. MOG: mapper on graphs for relationship preserving clustering. *CoRR*, abs/1804.11242, 2018.

[43] Cristian Bodnar, Cătălina Cangea, and Pietro Liò. Deep graph mapper: Seeing graphs through the neural lens. *Frontiers in Big Data*, 4:680535, 2021.

[44] Felix Hensel, Michael Moor, and Bastian Rieck. A survey of topological machine learning methods. *Frontiers in Artificial Intelligence*, 4:52, 2021.

[45] Cohen-Steiner D Edelsbrunner H Harer. J mileyko y lipschitz functions have l p-stable persistence. *Found. Comput. Math*, 10(2):127, 2010.

[46] Chao Chen and Herbert Edelsbrunner. Diffusion runs low on persistence fast. In *2011 International Conference on Computer Vision*, pages 423–430. IEEE, 2011.

[47] Nieves Atienza, Rocio Gonzalez-Diaz, and Matteo Rucco. Persistent entropy for separating topological features from noise in vietoris-rips complexes. *Journal of Intelligent Information Systems*, 52(3):637–655, 2019.

[48] Bastian Rieck, Filip Sadlo, and Heike Leitte. Topological machine learning with persistence indicator functions. In *Topological Methods in Data Analysis and Visualization*, pages 87–101. Springer, 2017.

[49] Yuhei Umeda. Time series classification via topological data analysis. *Information and Media Technologies*, 12:228–239, 2017.

[50] Ilya Chevyrev, Vidit Nanda, and Harald Oberhauser. Persistence paths and signature features in topological data analysis. *IEEE transactions on pattern analysis and machine intelligence*, 42(1):192–202, 2018.

[51] Qi Zhao, Ze Ye, Chao Chen, and Yusu Wang. Persistence enhanced graph neural network. In *International Conference on Artificial Intelligence and Statistics*, pages 2896–2906. PMLR, 2020.

[52] Michael Moor, Max Horn, Bastian Rieck, and Karsten Borgwardt. Topological autoencoders. In *International conference on machine learning*, pages 7045–7054. PMLR, 2020.

[53] Monisha Yuvaraj, Asim K Dey, Vyacheslav Lyubchich, Yulia R Gel, and H Vincent Poor. Topological clustering of multilayer networks. *Proceedings of the National Academy of Sciences*, 118(21):e2019994118, 2021.

[54] Paul Samuel Ignacio, Jay-Anne Bulauan, and John Rick Manzanares. A topology informed random forest classifier for ECG classification. In *2020 Computing in Cardiology*, pages 1–4. IEEE, 2020.

[55] Chao Chen, Xiuyan Ni, Qinxun Bai, and Yusu Wang. A topological regularizer for clas-

sifiers via persistent homology. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2573–2582. PMLR, 2019.

[56] Christoph Hofer, Florian Graf, Marc Niethammer, and Roland Kwitt. Topologically densified distributions. In *International Conference on Machine Learning*, pages 4304–4313. PMLR, 2020.

[57] Tamal Krishna Dey and Yusu Wang. *Computational topology for data analysis*. Cambridge University Press, 2022.

[58] Larry Wasserman. Topological data analysis. *Annual Review of Statistics and Its Application*, 5:501–532, 2018.

[59] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. Deep learning with topological signatures. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[60] Xavier Papillon, Andrew Blumberg, et al. Architectures of topological deep learning: a survey on topological neural networks. *arXiv preprint arXiv:2304.10031*, 2023.

[61] Ahsan Zia et al. Topological deep learning: a review. *Journal of Machine Learning Research*, 25(183):1–52, 2024.

[62] Baris Coskunuzer and Cüneyt Gürcan Akçora. Topological methods in machine learning: A tutorial for practitioners. *arXiv preprint arXiv:2409.02901*, 2024.

[63] Nurul A Asif, Yeahia Sarker, Ripon K Chakrabortty, Michael J Ryan, Md Hafiz Ahamed, Dip K Saha, Faisal R Badal, Sajal K Das, Md Firoz Ali, Sumaya I Moyeen, et al. Graph neural network: A comprehensive review on non-euclidean space. *Ieee Access*, 9:60588–60606, 2021.

[64] Abdul Majeed and Ibtisam Rauf. Graph theory: A comprehensive survey about graph theory applications in computer science and social networks. *Inventions*, 5(1):10, 2020.

[65] Georgios A Pavlopoulos, Maria Secrier, Charalampos N Moschopoulos, Theodoros G Soldatos, Sophia Kossida, Jan Aerts, Reinhard Schneider, and Pantelis G Bagos. Using graph theory to analyze biological networks. *BioData mining*, 4:1–27, 2011.

[66] Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert systems with applications*, 207:117921, 2022.

[67] Shunxin Xiao, Shiping Wang, Yuanfei Dai, and Wenzhong Guo. Graph neural networks in node classification: survey and evaluation. *Machine Vision and Applications*, 33(1):4, 2022.

[68] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37, 2016.

[69] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3558–3565, 2019.

[70] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *iEEE Access*, 9:79143–79168, 2021.

[71] Smriti Bhagat, Graham Cormode, and S Muthukrishnan. Node classification in social networks. *Social network data analytics*, pages 115–148, 2011.

[72] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[73] Ajay Kumar, Shashank Sheshar Singh, Kuldeep Singh, and Bhaskar Biswas. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications*, 553:124289, 2020.

[74] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM computing surveys (CSUR)*, 49(4):1–33, 2016.

[75] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

[76] Charu Aggarwal, Gewen He, and Peixiang Zhao. Edge classification in networks. In *2016 IEEE 32nd international conference on data engineering (ICDE)*, pages 1038–1049. IEEE, 2016.

[77] Biao Cai, Yanpeng Wang, Lina Zeng, Yanmei Hu, and Hongjun Li. Edge classification based on convolutional neural networks for community detection in complex network. *Physica A: statistical mechanics and its applications*, 556:124826, 2020.

[78] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *ICLR*, 2020.

[79] Koji Tsuda and Hiroto Saigo. Graph classification. *Managing and mining graph data*, pages 337–363, 2010.

[80] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1666–1674, 2018.

[81] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[82] Xiaodong Jiang, Pengsheng Ji, and Sheng Li. Censnet: Convolution with edge-node switching in graph neural networks. In *IJCAI*, pages 2656–2662, 2019.

[83] Xiaofeng Wang, Zhen Li, Mingjian Jiang, Shuang Wang, Shugang Zhang, and Zhiqiang Wei. Molecule property prediction based on spatial graph embedding. *Journal of chemical information and modeling*, 59(9):3817–3828, 2019.

[84] Oliver Wieder, Stefan Kohlbacher, Mélaine Kuenemann, Arthur Garon, Pierre Ducrot, Thomas Seidel, and Thierry Langer. A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, 37:1–12, 2020.

[85] Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. Graph-based semi-supervised learning: A comprehensive review. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8174–8194, 2022.

[86] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710. ACM, 2014.

[87] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[88] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[89] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *ICLR (Poster)*, 2(3):4, 2019.

[90] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and S Yu Philip. Graph self-supervised learning: A survey. *IEEE transactions on knowledge and data engineering*, 35(6):5879–5900, 2022.

[91] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[92] Gireen Naidu, Tranos Zuva, and Elias Mmbongeni Sibanda. A review of evaluation metrics in machine learning algorithms. In *Computer science on-line conference*, pages 15–25. Springer, 2023.

[93] Francesco Scarselli, Marco Gori, Alexander C. Tsoi, Michael Hagenbuchner, and Giovanni Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[94] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)*, New York, NY, USA, 2018. ACM.

[95] X. Li, Y. Zhang, X. Chen, Z. Li, and J. Li. Traffic flow prediction with big data: A deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2570–2582, 2018.

[96] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gomez-Bombarelli, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 2224–2232, Cambridge, MA, USA, 2015. MIT Press.

[97] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[98] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 3844–3852. Curran Associates, Inc., 2016.

[99] Zhiqian Chen, Fanglan Chen, Lei Zhang, Taoran Ji, Kaiqun Fu, Liang Zhao, Feng Chen, Lingfei Wu, Charu Aggarwal, and Chang-Tien Lu. Bridging the gap between spatial and

spectral domains: A survey on graph neural networks. *arXiv preprint arXiv:2201.04833*, 2022.

[100] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Proceedings of the 13th European Conference on Computer Vision (ECCV)*, Munich, Germany, 2018. Springer.

[101] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, 2017.

[102] Fan Liang, Cheng Qian, Wei Yu, David Griffith, and Nada Golmie. Survey of graph neural networks and applications. *Wireless Communications and Mobile Computing*, 2022:1–17, 2022.

[103] William L. Hamilton, Rex Ying, and Jure Leskovec. *Representation Learning on Graphs: Methods and Applications*. Cambridge University Press, 2017.

[104] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2017.

[105] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2018.

[106] B. Yu. Weisfeiler and A. A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsiya*, 2:12–16, 1968.

[107] Zhiyuan Xu, Wei Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.08473*, 2018.

[108] Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic

graphs. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 2358–2366, 2022.

[109] Jianfei Gao and Bruno Ribeiro. On the equivalence between temporal and static equivariant graph representations. In *International Conference on Machine Learning*, pages 7052–7076. Proceedings of Machine Learning Research, 2022.

[110] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[111] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[112] Menglin Yang, Min Zhou, Marcus Kalander, Zengfeng Huang, and Irwin King. Discrete-time temporal network embedding via implicit hierarchical learning in hyperbolic space. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1975–1985, 2021.

[113] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[114] Emanuele Rossi, Ben Chamberlain, Michael Bronstein, and Michaël Defferrard. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[115] Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards

better evaluation for dynamic link prediction. *Advances in Neural Information Processing Systems*, 35:32928–32941, 2022.

[116] Amauri Souza, Diego Mesquita, Samuel Kaski, and Vikas Garg. Provably expressive temporal graph networks. *Advances in Neural Information Processing Systems*, 35:32257–32269, 2022.

[117] Yuhong Luo and Pan Li. Neighborhood-aware scalable temporal network representation learning. In *Learning on Graphs Conference*, pages 1–1. Proceedings of Machine Learning Research, 2022.

[118] Ming Jin, Yuan-Fang Li, and Shirui Pan. Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs. *Advances in Neural Information Processing Systems*, 35:19874–19886, 2022.

[119] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael M. Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advance in Neural Information Processing Systems*, 2023.

[120] Ling Zhao, Huadong Wang, Xuebin Zheng, and Yu Zhang. T-gcn: A temporal graph convolutional network for traffic prediction. In *Proceedings of the 33rd Conference on Artificial Intelligence (AAAI)*, Honolulu, HI, USA, 2019. AAAI Press.

[121] Yejin Kim, Youngbin Lee, Minyoung Choe, Sungju Oh, and Yongjae Lee. Temporal graph networks for graph anomaly detection in financial networks. *arXiv preprint arXiv:2302.08047*, 2023.

[122] Mengqi Zhang, Shu Wu, Xueli Yu, Qiang Liu, and Liang Wang. Dynamic graph neural networks for sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1221–1234, 2021.

[123] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[124] Leonardo Cotta, Christopher Morris, and Bruno Ribeiro. Reconstruction for powerful graph representations. *Advances in Neural Information Processing Systems*, 34:1713–1726, 2021.

[125] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-GCN: A temporal graph convolutional network for traffic prediction. *IEEE Trans. Intell. Transp. Syst.*, 21(9):3848–3858, 2020.

[126] Menglin Yang, Min Zhou, Marcus Kalander, Zengfeng Huang, and Irwin King. Discrete-time temporal network embedding via implicit hierarchical learning in hyperbolic space. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 1975–1985. ACM, 2021.

[127] Jinyin Chen, Xueke Wang, and Xuanheng Xu. GC-LSTM: graph convolution embedded LSTM for dynamic network link prediction. *Appl. Intell.*, 52(7):7513–7528, 2022.

[128] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5363–5370. AAAI Press, 2020.

[129] Yifan Zhu, Fangpeng Cong, Dan Zhang, Wenwen Gong, Qika Lin, Wenzheng Feng, Yuxiao Dong, and Jie Tang. Wingnn: Dynamic graph neural networks with random gradient aggregation window. In *KDD*, pages 3650–3662, 2023.

[130] Xiangguo Sun, Hong Cheng, Jia Li, Bo Liu, and Jihong Guan. All in one: Multi-task prompting for graph neural networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2120–2131, 2023.

[131] Jiawei Liu, Cheng Yang, Zhiyuan Lu, Junze Chen, Yibo Li, Mengmei Zhang, Ting Bai, Yuan Fang, Lichao Sun, Philip S Yu, et al. Towards graph foundation models: A survey and beyond. *arXiv preprint arXiv:2310.11829*, 2023.

[132] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

[133] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.

[134] Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. Language is all a graph needs. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1955–1973, 2024.

[135] Chang Liu and Bo Wu. Evaluating large language models on graphs: Performance insights and comparative analysis. *arXiv preprint arXiv:2308.11224*, 2023.

[136] Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhuo Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. Graphformers: Gnn-nested transformers for representation learning on textual graph. *Advances in Neural Information Processing Systems*, 34:28798–28810, 2021.

[137] Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. Learning on large-scale text-attributed graphs via variational inference. *arXiv preprint arXiv:2210.14709*, 2022.

[138] Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang.

Self-supervised learning on graphs: Deep insights and new direction. *arXiv preprint arXiv:2006.10141*, 2020.

[139] Mingchen Sun, Kaixiong Zhou, Xin He, Ying Wang, and Xin Wang. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1717–1727, 2022.

[140] Zemin Liu, Xingtong Yu, Yuan Fang, and Xinming Zhang. Graphprompt: Unifying pre-training and downstream tasks for graph neural networks. In *Proceedings of the ACM Web Conference 2023*, pages 417–428, 2023.

[141] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

[142] Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-shot learning-the good, the bad and the ugly. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4582–4591, 2017.

[143] John Cai and Sheng Mei Shen. Cross-domain few-shot learning with meta fine-tuning. *CoRR*, abs/2005.10544, 2020.

[144] Taoran Fang, Yunchao Zhang, Yang Yang, Chunping Wang, and Lei Chen. Universal prompt tuning for graph neural networks. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[145] Ruosong Ye. Language is all a graph needs, 2021. Available at `https://arxiv.org/abs/2106.12916`.

[146] Heng Wang. Can language models solve graph problems?, 2023. Available at `https://arxiv.org/abs/2305.07181`.

[147] C. Liu and B. Wu. Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv preprint arXiv:2308.11224*, 2023.

[148] X. Li, J. Xu, H. Chen, et al. Gimlet: A unified graph-text model for instruction-based molecule zero-shot learning. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pages 1234–1245. PMLR, 2023.

[149] Jianan Zhao. Graphtext: Graph reasoning in text space, 2023.

[150] Frederik Wenkel, Zehong Liu, Siamak Shakeri, Tengfei Wang, and Davide Mottin. Pre-trained language models to solve graph tasks in natural language. In *International Conference on Learning Representations (ICLR)*, 2023.

[151] Shunyu Yao et al. Tree of thoughts: Deliberate problem solving with large language models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 5678–5689. PMLR, 2023.

[152] X. Chen, Y. Zhang, J. Li, et al. Exploring the potential of large language models (llms) in learning on graphs. *Journal of Machine Learning Research*, 24:1–20, 2023.

[153] Costas Mavromatis et al. Train your own gnn teacher: Graph-aware distillation on textual graphs. *arXiv preprint arXiv:2305.13051*, 2023.

[154] Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 491–500, 2024.

[155] Yaorui Shi et al. Relm: Leveraging language models for enhanced chemical reaction prediction. *arXiv preprint arXiv:2401.12345*, 2024.

[156] Jianing Wang et al. Instructgraph: Boosting large language models via graph-centric instruction tuning and preference alignment. *arXiv preprint arXiv:2402.67890*, 2024.

[157] Carl Edwards. Text2mol: Cross-modal molecule retrieval with natural language queries. *ArXiv preprint arXiv:2207.07282*, 2022.

[158] Philipp Seidl. Enhancing activity prediction models in drug discovery with the ability to understand human language. *arXiv preprint arXiv:2301.01234*, 2023.

[159] H. Xie, D. Zheng, J. Ma, H. Zhang, V. N. Ioannidis, X. Song, Q. Ping, S. Wang, C. J. Yang, Y. Xu, et al. Graph-aware language model pre-training on a large graph corpus can help multiple graph applications. In *Proceedings of the 29th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2023.

[160] Zhen Wu, Haoran Li, Jie Zhang, Tianyi Wang, and Yue Wang. Graphformers: Gnn-nested transformers for representation learning on textual graph. *arXiv preprint arXiv:2302.02318*, 2023.

[161] Yao Zhou et al. Glem: Language model empowered gnn pre-training. *arXiv preprint arXiv:2307.00000*, 2023.

[162] Keyu Duan, Zhen Li, Yifan Yang, Changlue Sun, and Jian Zhang. Simteg: A frustratingly simple approach improves textual graph learning. *arXiv preprint arXiv:2304.06589*, 2023.

[163] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S. Dhillon. Node feature extraction by self-supervised multi-scale neighborhood prediction. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[164] Zihan Zhou, Jiawei Liu, Qing Zhang, and Haifeng Wang. Textgt: A double-view graph transformer on text for aspect-based sentiment analysis. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[165] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[166] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[167] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[168] Alec Radford et al. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763. PMLR, 2021.

[169] B. Jin, W. Zhang, Y. Zhang, Y. Meng, X. Zhang, Q. Zhu, and J. Han. Patton: Language model pretraining on text-rich networks. In A. Rogers, J. L. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7005–7020, Toronto, Canada, 2023. Association for Computational Linguistics.

[170] M. Zhang, M. Sun, P. Wang, S. Fan, Y. Mo, X. Xu, H. Liu, C. Yang, and C. Shi. Graphtranslator: Aligning graph model to large language model for open-ended tasks. In T. Chua, C. Ngo, R. Kumar, H. W. Lauw, and R. K. Lee, editors, *Proceedings of the ACM on Web Conference 2024*, pages 1003–1014, Singapore, 2024. ACM.

[171] Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondrej Chum. Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5070–5079, 2019.

[172] Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibo Wang, and Ji-Rong Wen. Efficient algorithms for personalized pagerank computation: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[173] Jianan Zhao, Le Zhuo, Yikang Shen, Meng Qu, Kai Liu, Michael M. Bronstein, Zhaocheng Zhu, and Jian Tang. Graphtext: Graph reasoning in text space. *CoRR*, abs/2310.01089, 2023.

[174] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[175] Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Position: Graph foundation models are already here. In *Forty-first International Conference on Machine Learning*, 2024.

[176] Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. Towards foundation models for knowledge graph reasoning. In *The Twelfth International Conference on Learning Representations*, 2023.

[177] Mikhail Galkin, Jincheng Zhou, Bruno Ribeiro, Jian Tang, and Zhaocheng Zhu. Zero-shot logical query reasoning on any knowledge graph. *arXiv preprint arXiv:2404.07198*, 2024.

[178] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7), 2021.

[179] Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, et al. A generalist neural algorithmic learner. In *Learning on graphs conference*, pages 2–1. PMLR, 2022.

[180] Nima Shoghi, Adeesh Kolluru, John R Kitchin, Zachary W Ulissi, C Lawrence Zitnick, and

Brandon M Wood. From molecules to materials: Pre-training large generalizable models for atomic property prediction. *arXiv preprint arXiv:2310.16802*, 2023.

[181] Duo Zhang, Xinzijian Liu, Xiangyu Zhang, Chengqian Zhang, Chun Cai, Hangrui Bi, Yiming Du, Xuejian Qin, Jiameng Huang, Bowen Li, et al. Dpa-2: Towards a universal large atomic model for molecular and material simulation. *arXiv preprint arXiv:2312.15492*, 2023.

[182] Ilyes Batatia, Philipp Benner, Yuan Chiang, Alin M Elena, Dávid P Kovács, Janosh Riebesell, Xavier R Advincula, Mark Asta, William J Baldwin, Noam Bernstein, et al. A foundation model for atomistic materials chemistry. *arXiv preprint arXiv:2401.00096*, 2023.

[183] H Yang, C Hu, Y Zhou, X Liu, Y Shi, J Li, G Li, Z Chen, S Chen, C Zeni, et al. Mattersim: A deep learning atomistic model across elements, temperatures and pressures (2024). *arXiv preprint arXiv:2405.04967*, 2024.

[184] Kerstin Klaser, Blazej Banaszewski, Samuel Maddrell-Mander, Callum McLean, Luis Müller, Ali Parviz, Shenyang Huang, and Andrew W Fitzgibbon. Minimol: A parameter-efficient foundation model for molecular learning. In *ICML 2024 Workshop on Efficient and Accessible Foundation Models for Biological Discovery*, 2024.

[185] Maciej Sypetkowski, Frederik Wenkel, Farimah Poursafaei, Nia Dickson, Karush Suri, Philip Fradkin, and Dominique Beaini. On the scalability of gnns for molecular graphs. *arXiv preprint arXiv:2404.11568*, 2024.

[186] Jianan Zhao, Hesham Mostafa, Michael Galkin, Michael Bronstein, Zhaocheng Zhu, and Jian Tang. Graphany: A foundation model for node classification on any graph. *arXiv preprint arXiv:2405.20445*, 2024.

[187] Kaiwen Dong, Haitao Mao, Zhichun Guo, and Nitesh V Chawla. Universal link predictor by in-context learning. *arXiv preprint arXiv:2402.07738*, 2024.

[188] Lianghao Xia and Chao Huang. Anygraph: Graph foundation model in the wild. *arXiv preprint arXiv:2408.10700*, 2024.

[189] Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, Franco Scarselli, and Andrea Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *Trans. Mach. Learn. Res.*, 2023, 2023.

[190] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *CoRR*, abs/2007.08663, 2020.

[191] Jure Leskovec and Rok Sosic. SNAP: A general-purpose network analysis and graph-mining library. *ACM Trans. Intell. Syst. Technol.*, 8(1):1:1–1:20, 2016.

[192] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[193] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[194] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. OGB-LSC: A large-scale challenge for machine learning on graphs. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.

[195] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation?

In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 28877–28888, 2021.

[196] Ningyi Liao, Haoyu Liu, Zulun Zhu, Siqiang Luo, and Laks V. S. Lakshmanan. Benchmarking spectral graph neural networks: A comprehensive study on effectiveness and efficiency. *CoRR*, abs/2406.09675, 2024.

[197] Julia Gastinger, Shenyang Huang, Michael Galkin, Erfan Loghmani, Ali Parviz, Farimah Poursafaei, Jacob Danovitch, Emanuele Rossi, Ioannis Koutis, Heiner Stuckenschmidt, et al. Tgb 2.0: A benchmark for learning on temporal knowledge graphs and heterogeneous graphs. *Advances in neural information processing systems*, 37:140199–140229, 2024.

[198] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591*, 2019.

[199] Sijia Li, Gaopeng Gou, Chang Liu, Chengshang Hou, Zhenzhen Li, and Gang Xiong. Ttagn: Temporal transaction aggregation graph network for ethereum phishing scams detection. In *Proceedings of the ACM Web Conference 2022*, pages 661–669, 2022.

[200] Bingqiao Luo, Zhen Zhang, Qian Wang, and Bingsheng He. Multi-chain graphs of graphs: A new approach to analyzing blockchain datasets. *Advances in Neural Information Processing Systems*, 37:28490–28514, 2024.

[201] Qian Wang, Zhen Zhang, Zemin Liu, Shengliang Lu, Bingqiao Luo, and Bingsheng He. Ex-graph: A pioneering dataset bridging ethereum and X. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[202] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. GPT-GNN: generative pre-training of graph neural networks. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 1857–1867. ACM, 2020.

[203] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

[204] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In Aidong Zhang and Huzefa Rangwala, editors, *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pages 594–604. ACM, 2022.

[205] Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards better evaluation for dynamic link prediction. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *NeurIPS 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[206] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[207] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[208] Celestine Iwendi, Ali Kashif Bashir, Atharva Peshkar, R Sujatha, Jyotir Moy Chatterjee,

Swetha Pasupuleti, Rishita Mishra, Sofia Pillai, and Ohyun Jo. Covid-19 patient health prediction using boosted random forest algorithm. *Frontiers in public health*, 8:357, 2020.

[209] Mohammadreza Sheykhmousa, Masoud Mahdianpari, Hamid Ghanbari, Fariba Mohammadimanesh, Pedram Ghamisi, and Saeid Homayouni. Support vector machine vs. random forest for remote sensing image classification: A meta-analysis and systematic review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2020.

[210] Shibani Hamsa, Ismail Shahin, Youssef Iraqi, and Naoufel Werghi. Emotion recognition from speech using wavelet packet transform cochlear filter bank and random forest classifier. *IEEE Access*, 8:96994–97006, 2020.

[211] S Abijah Roseline, S Geetha, Seifedine Kadry, and Yunyoung Nam. Intelligent vision-based malware detection and classification using deep random forest paradigm. *IEEE Access*, 8:206303–206324, 2020.

[212] Yanyu Chen, Wenzhe Zheng, Wenbo Li, and Yimiao Huang. Large group activity security risk assessment and risk early warning based on random forest algorithm. *Pattern Recognition Letters*, 144:1–5, 2021.

[213] Long Cheng, Xuewu Chen, Jonas De Vos, Xinjun Lai, and Frank Witlox. Applying a random forest method approach to model travel mode choice behavior. *Travel behaviour and society*, 14:1–10, 2019.

[214] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[215] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[216] Jianguo Chen, Kenli Li, Zhuo Tang, Kashif Bilal, Shui Yu, Chuliang Weng, and Keqin Li. A parallel random forest algorithm for big data in a spark cloud computing environment. *IEEE Trans. Parallel Distributed Syst.*, 28(4):919–933, 2017.

[217] Yulong Xu. Research and implementation of improved random forest algorithm based on spark. In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, pages 499–503. IEEE, 2017.

[218] Murat Ali Bayir, Mingsen Xu, Yaojia Zhu, and Yifan Shi. Genie: An open box counterfactual policy estimator for optimizing sponsored search marketplace. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*, pages 465–473. ACM, 2019.

[219] Weiwei Lin, Ziming Wu, Longxin Lin, Angzhan Wen, and Jin Li. An ensemble random forest algorithm for insurance big data analysis. *Ieee access*, 5:16568–16575, 2017.

[220] Sara Del Río, Victoria López, José Manuel Benítez, and Francisco Herrera. On the use of mapreduce for imbalanced big data using random forest. *Information Sciences*, 285:112–137, 2014.

[221] Shuxi Zeng, Murat Ali Bayir, Joseph J Pfeiffer III, Denis Charles, and Emre Kiciman. Causal transfer random forest: Combining logged data and randomized experiments for robust prediction. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 211–219, 2021.

[222] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. Xai—explainable artificial intelligence. *Science Robotics*, 4(37), 2019.

[223] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 0210–0215. IEEE, 2018.

[224] Omer Sagi and Lior Rokach. Explainable decision forest: Transforming a decision forest into an interpretable tree. *Information Fusion*, 61:124–138, 2020.

[225] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. Explainable ai for trees: From local explanations to global understanding. *arXiv preprint arXiv:1905.04610*, 2019.

[226] Dragutin Petkovic, Russ Altman, Mike Wong, and Arthur Vigil. Improving the explainability of random forest classifier–user centered approach. In *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2018: Proceedings of the Pacific Symposium*, pages 204–215. World Scientific, 2018.

[227] Gözde Bakirli and Derya Birant. Dtreesim: A new approach to compute decision tree similarity using re-mining. *Turkish Journal of Electrical Engineering & Computer Sciences*, 25(1):108–125, 2017.

[228] Philipp Probst and Anne-Laure Boulesteix. To tune or not to tune the number of trees in random forest. *J. Mach. Learn. Res.*, 18(1):6673–6690, 2017.

[229] Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(3):e1301, 2019.

[230] Thais Mayumi Oshiro, Pedro Santoro Perez, and José Augusto Baranauskas. How many trees in a random forest? In *International workshop on machine learning and data mining in pattern recognition*, pages 154–168. Springer, 2012.

[231] Patrice Latinne, Olivier Debeir, and Christine Decaestecker. Limiting the number of trees in random forests. In *International workshop on multiple classifier systems*, pages 178–187. Springer, 2001.

[232] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[233] Per Hage and Frank Harary. Eccentricity and centrality in networks. *Social networks*, 17(1):57–63, 1995.

[234] Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Angela Ricciardello. A novel measure of edge centrality in social networks. *Knowledge-based systems*, 30:136–150, 2012.

[235] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On finding graph clusterings with maximum modularity. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 121–132. Springer, 2007.

[236] Thomas Schank and Dorothea Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9(2):265–275, 2005.

[237] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[238] Vladimir Batagelj and Andrej Mrvar. A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social networks*, 23(3):237–243, 2001.

[239] Jon M Kleinberg et al. Authoritative sources in a hyperlinked environment. In *SODA*, volume 98, pages 668–677. Citeseer, 1998.

[240] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177, 2001.

[241] Joshua O'Madadhain, Danyel Fisher, Padhraic Smyth, Scott White, and Yan-Biao Boey. Analysis and visualization of network data using jung. *Journal of Statistical Software*, 10(2):1–35, 2005.

[242] Gurjeet Singh, Facundo Memoli, and Gunnar Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. In M. Botsch, R. Pajarola, B. Chen, and M. Zwicker, editors, *Eurographics Symposium on Point-Based Graphics*. The Eurographics Association, 2007.

[243] Ron Kohavi et al. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, volume 96, pages 202–207, 1996.

[244] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[245] Robert Cattral, Franz Oppacher, and Dwight Deugo. Evolutionary data mining with automatic rule generalization. *Recent Advances in Computers, Computing and Communications*, 1(1):296–300, 2002.

[246] Peter W Frey and David J Slate. Letter recognition using holland-style adaptive classifiers. *Machine learning*, 6(2):161–182, 1991.

[247] Manuel Olave, Vladislav Rajkovic, and Marko Bohanec. An application for admission in public school systems. *Expert Systems in Public Administration*, 1:145–160, 1989.

[248] Harry Kalodner, Malte Möser, Kevin Lee, Steven Goldfeder, Martin Plattner, Alishah Chator, and Arvind Narayanan. Blocksci: Design and applications of a blockchain analysis platform. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2721–2738, 2020.

[249] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Available at SSRN 3440802*, 2008.

[250] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A survey on blockchain interoperability: Past, present, and future trends. *ACM Computing Surveys*, 54(8):168:1–168:41, 2022.

[251] Joachim Zahnentferner. Chimeric ledgers: translating and unifying utxo-based and account-based cryptocurrencies. *Cryptology ePrint Archive*, 2018.

[252] Rui Zhang, Rui Xue, and Ling Liu. Security and privacy on blockchain. *ACM Computing Surveys*, 52(3):51:1–51:34, 2019.

[253] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020.

[254] Cuneyt Gurcan Akcora, Yulia R. Gel, and Murat Kantarcioglu. Blockchain networks: Data structures of Bitcoin, Monero, Zcash, Ethereum, Ripple, and Iota. *WIREs Data Mining Knowledge and Discovery*, 12(1), 2022.

[255] Jie Shen, Jiajun Zhou, Yunyi Xie, Shanqing Yu, and Qi Xuan. Identity inference on blockchain using graph neural network. In Hong-Ning Dai, Xuanzhe Liu, Daniel Xiapu Luo, Jiang Xiao, and Xiangping Chen, editors, *Blockchain and Trustworthy Systems*, pages 3–17, Singapore, 2021. Springer Singapore.

[256] Cuneyt G. Akcora, Yitao Li, Yulia R. Gel, and Murat Kantarcioglu. Bitcoinheist: Topological data analysis for ransomware prediction on the Bitcoin blockchain. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4439–4445. International Joint Conferences on Artificial Intelligence Organization, 7 2020.

[257] Friedhelm Victor, Cuneyt G. Akcora, Yulia R. Gel, and Murat Kantarcioglu. Alphacore: Data depth based core decomposition. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 1625–1633, New York, NY, USA, 2021. Association for Computing Machinery.

[258] Jieli Liu, Jiatao Zheng, Jiajing Wu, and Zibin Zheng. Fa-gnn: Filter and augment graph

neural networks for account classification in ethereum. *IEEE Transactions on Network Science and Engineering*, 9(4):2579–2588, 2022.

[259] Rabia Musheer Aziz, Mohammed Farhan Baluch, Sarthak Patel, and Abdul Hamid Ganie. Lgbm: a machine learning approach for ethereum fraud detection. *International Journal of Information Technology*, pages 1–11, 2022.

[260] Jaehyeon Kim, Sejong Lee, Yushin Kim, Seyoung Ahn, and Sunghyun Cho. Graph learning-based blockchain phishing account detection with a heterogeneous transaction graph. *Sensors*, 23(1), 2023.

[261] Vatsal Patel, Lei Pan, and Sutharshan Rajasegarar. Graph deep learning based anomaly detection in ethereum blockchain network. In Mirosław Kutyłowski, Jun Zhang, and Chao Chen, editors, *Network and System Security*, pages 132–148, Cham, 2020. Springer International Publishing.

[262] Jinhuan Wang, Pengtao Chen, Xinyao Xu, Jiajing Wu, Meng Shen, Qi Xuan, and Xiaoniu Yang. Tsgn: Transaction subgraph networks assisting phishing detection in ethereum. *arXiv preprint arXiv:2208.12938*, 2022.

[263] Dunjie Zhang, Jinyin Chen, and Xiaosong Lu. Blockchain phishing scam detection via multi-channel graph classification. In Hong-Ning Dai, Xuanzhe Liu, Daniel Xiapu Luo, Jiang Xiao, and Xiangping Chen, editors, *Blockchain and Trustworthy Systems*, pages 241–256, Singapore, 2021. Springer Singapore.

[264] Dorcas Ofori-Boateng, Ignacio Segovia-Dominguez, Cuneyt Gurcan Akcora, Murat Kantarcioglu, and Yulia R. Gel. Topological anomaly detection in dynamic multilayer blockchain networks. In Nuria Oliver, Fernando Pérez-Cruz, Stefan Kramer, Jesse Read, and José Antonio Lozano, editors, *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021,*

Proceedings, Part I, volume 12975 of *Lecture Notes in Computer Science*, pages 788–804. Springer, 2021.

[265] Cuneyt Gurcan Akcora, Asim Kumer Dey, Yulia R. Gel, and Murat Kantarcioglu. Forecasting bitcoin price with graph chainlets. In Dinh Q. Phung, Vincent S. Tseng, Geoffrey I. Webb, Bao Ho, Mohadeseh Ganji, and Lida Rashidi, editors, *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part III*, volume 10939 of *Lecture Notes in Computer Science*, pages 765–776. Springer, 2018.

[266] Nazmiye Ceren Abay, Cuneyt Gurcan Akcora, Yulia R. Gel, Murat Kantarcioglu, Umar D. Islambekov, Yahui Tian, and Bhavani Thuraisingham. Chainnet: Learning on blockchain graphs with topological features. In Jianyong Wang, Kyuseok Shim, and Xindong Wu, editors, *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, pages 946–951. IEEE, 2019.

[267] Yeray Mezquita, Ana Belén Gil-González, Javier Prieto, and Juan Manuel Corchado. Cryptocurrencies and price prediction: A survey. In Javier Prieto, Alberto Partida, Paulo Leitão, and António Pinto, editors, *Blockchain and Applications*, pages 339–346, Cham, 2022. Springer International Publishing.

[268] Patrick Jaquart, Sven Köpke, and Christof Weinhardt. Machine learning for cryptocurrency market prediction and trading. *The Journal of Finance and Data Science*, 8:331–352, 2022.

[269] Yitao Li, Umar Islambekov, Cuneyt Gurcan Akcora, Ekaterina Smirnova, Yulia R. Gel, and Murat Kantarcioglu. Dissecting ethereum blockchain analytics: What we learn from topology and geometry of the ethereum graph? In Carlotta Demeniconi and Nitesh V. Chawla, editors, *Proceedings of the 2020 SIAM International Conference on Data Mining, SDM 2020, Cincinnati, Ohio, USA, May 7-9, 2020*, pages 523–531. SIAM, 2020.

[270] Yuzhou Chen, Ignacio Segovia-Dominguez, and Yulia R. Gel. Z-gcnets: Time zigzags

at graph convolutional networks for time series forecasting. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1684–1694. PMLR, 2021.

[271] Valeria D'Amato, Susanna Levantesi, and Gabriella Piscopo. Deep learning in predicting cryptocurrency volatility. *Physica A: Statistical Mechanics and its Applications*, 596:127158, 2022.

[272] Yuzhou Chen, Ignacio Segovia-Dominguez, Baris Coskunuzer, and Yulia R. Gel. Tamps2gcnets: Coupling time-aware multipersistence knowledge representation with spatio-supra graph convolutional networks for time-series forecasting. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[273] Yuzhou Chen, Yulia Gel, and H Vincent Poor. Time-conditioned dances with simplicial complexes: Zigzag filtration curve based supra-hodge convolution networks for time-series forecasting. *Advances in Neural Information Processing Systems*, 35:8940–8953, 2022.

[274] Cuneyt Gurcan Akcora, Matthew F Dixon, Yulia R Gel, and Murat Kantarcioglu. Bitcoin risk modeling with blockchain graphs. *Economics Letters*, 173:138–142, 2018.

[275] Adriano Zanin Zambom and Yulia R Gel. Testing for local covariate trend effects in volatility models. *Electronic Journal of Statistics*, 14(2):2529–2550, 2020.

[276] Masarah Paquet-Clouston, Bernhard Haslhofer, and Benoit Dupont. Ransomware payments in the Bitcoin ecosystem. *Journal of Cybersecurity*, 5(1):tyz003, 2019.

[277] Danny Yuxing Huang, Maxwell Matthaios Aliapoulios, Vector Guo Li, Luca Invernizzi, Elie Bursztein, Kylie McRoberts, Jonathan Levin, Kirill Levchenko, Alex C. Snoeren, and Damon McCoy. Tracking ransomware end-to-end. In *2018 IEEE Symposium on Security*

and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA, pages 618–631. IEEE Computer Society, 2018.

[278] Mauro Conti, Ankit Gangwal, and Sushmita Ruj. On the economic significance of ransomware campaigns: A Bitcoin transactions perspective. *Computers and Security*, 79:162–189, 2018.

[279] J. Hernandez-Castro, E. Boiten, and M. Barnoux. The 2nd kent cyber security survey. *Kent University reports*, 2014. https://kar.kent.ac.uk/52891/.

[280] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320. USENIX, 2004.

[281] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of Bitcoins: characterizing payments among men with no names. *Communications of the ACM*, 59(4):86–93, 2016.

[282] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[283] Friedhelm Victor and Bianca Katharina Lüders. Measuring ethereum-based ERC20 token networks. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2019.

[284] Commodity Futures Trading Commission (CFTC). Commodity exchange act & regulations, 2020.

[285] Friedhelm Victor and Andrea Marie Weintraud. Detecting and quantifying wash trading

on decentralized cryptocurrency exchanges. In *Proceedings of the Web Conference 2021*, WWW '21, page 23–32, New York, NY, USA, 2021. Association for Computing Machinery.

[286] Catarina Oliveira, João Torres, Maria Inês Silva, David Aparício, João Tiago Ascensão, and Pedro Bizarro. Guiltywalker: Distance to illicit nodes in the Bitcoin network. *CoRR*, abs/2102.05373, 2021.

[287] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[288] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[289] Albert-László Barabási. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1987):20120375, 2013.

[290] Isaac Newton. *Philosophiae naturalis principia mathematica*, volume 1. G. Brookman, 1833.

[291] Lada A Adamic and Bernardo A Huberman. Power-law distribution of the world wide web. *Science*, 287(5461):2115–2115, 2000.

[292] Sean Carroll. *The Biggest Ideas in the Universe: Space, Time and Motion*. Dutton, Hoboken, NJ, 2022.

[293] Oded Galor. *Discrete dynamical systems*. Springer Science & Business Media, 2007.

[294] Jeff Alstott, Ed Bullmore, and Dietmar Plenz. Powerlaw: a python package for analysis of heavy-tailed distributions. *PloS One*, 9(1):e85777, 2014.

[295] Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

[296] Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. In *Proceedings of the Ecological Inference Conference*, volume 17. Citeseer, 2011.

[297] Laura A Zager and George C Verghese. Graph similarity scoring and matching. *Applied mathematics letters*, 21(1):86–94, 2008.

[298] Friedhelm Victor. Address clustering heuristics for ethereum. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*, pages 617–633. Springer, 2020.

[299] Han-Min Kim, Gee-Woo Bock, and Gunwoong Lee. Predicting ethereum prices with machine learning based on blockchain information. *Expert Systems with Applications*, 184:115480, 2021.

[300] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 601–610, 2017.

[301] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 933–943. International World Wide Web Conferences Steering Committee, 2018.

[302] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I 25*, pages 362–373. Springer, 2018.

[303] Peter T Yamak, Li Yujian, and Pius K Gadosey. A comparison between arima, lstm, and gru for time series forecasting. In *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, pages 49–55, 2019.

[304] Nicola Pezzotti, Boudewijn PF Lelieveldt, Laurens Van Der Maaten, Thomas Höllt, Elmar Eisemann, and Anna Vilanova. Approximated and user steerable tsne for progressive visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 23(7):1739–1752, 2016.

[305] Xiao Li, Li Sun, Mengjie Ling, and Yan Peng. A survey of graph neural network based recommendation in social networks. *Neurocomputing*, 549:126441, 2023.

[306] Pietro Bongini, Niccolò Pancino, Franco Scarselli, and Monica Bianchini. Biognn: how graph neural networks can solve biological problems. In *Artificial Intelligence and Machine Learning for Healthcare: Vol. 1: Image and Data Analytics*, pages 211–231. Springer, 2022.

[307] José Suárez-Varela, Paul Almasan, Miquel Ferriol-Galmés, Krzysztof Rusek, Fabien Geyer, Xiangle Cheng, Xiang Shi, Shihan Xiao, Franco Scarselli, Albert Cabellos-Aparicio, et al. Graph neural networks for communication networks: Context, use cases and opportunities. *IEEE network*, 37(3):146–153, 2022.

[308] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[309] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[310] Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Hena Ghonia, Rishika Bhagwatkar, Arian Khorasani, Mohammad Javad Darvishi Bayazi, George Adamopoulos, Roland Riachi, Nadhir Hassen, Marin Biloš, Sahil Garg, Anderson Schneider, Nicolas Chapados, Alexandre Drouin, Valentina Zantedeschi, Yuriy Nevmyvaka, and Irina Rish. Lag-llama: Towards foundation models for probabilistic time series forecasting, 2024.

[311] Muhammad Awais, Muzammal Naseer, Salman Khan, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, and Fahad Shahbaz Khan. Foundational models defining a new era in vision: A survey and outlook. *arXiv preprint arXiv:2307.13721*, 2023.

[312] Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Position: Graph foundation models are already here. In *ICML*, 2024.

[313] Dominique Beaini, Shenyang Huang, Joao Alex Cunha, Zhiyi Li, Gabriela Moisescu-Pareja, Oleksandr Dymov, Samuel Maddrell-Mander, Callum McLean, Frederik Wenkel, Luis Müller, et al. Towards foundational models for molecular learning on large-scale multi-task datasets. In *The Twelfth International Conference on Learning Representations*, 2023.

[314] Duo Wang, Yuan Zuo, Fengzhi Li, and Junjie Wu. LLMs as zero-shot graph learners: Alignment of GNN representations with LLM token embeddings. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[315] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. Temporal graph networks for deep learning on dynamic graphs. *CoRR*, abs/2006.10637, 2020.

[316] Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. *NeurIPS*, 36, 2024.

[317] Zhen Zhang, Bingqiao Luo, Shengliang Lu, and Bingsheng He. Live graph lab: Towards open, dynamic and real transaction graphs with NFT. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *NeurIPS 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[318] Lin William Cong, Xi Li, Ke Tang, and Yang Yang. Crypto wash trading. *Management Science*, 69(11):6427–6454, 2023.

[319] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[320] Benedek Rozemberczki and Rik Sarkar. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, page 1325–1334. ACM, 2020.

[321] Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2009.

[322] Zhuxuanzi Wang, Xu Wang, and Hongbo Wang. Temporal graph neural networks for money laundering detection in cross-border transactions. *Academia Nexus Journal*, 3(2), 2024.

[323] Zahraa Al Sahili and Mariette Awad. Spatio-temporal graph neural networks: A survey. *arXiv preprint arXiv:2301.10569*, 2023.

[324] Mohammad Mehdi Hosseinzadeh, Mario Cannataro, Pietro Hiram Guzzi, and Riccardo Dondi. Temporal networks in biology and medicine: a survey on models, algorithms, and tools. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 12(1):10, 2022.

[325] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *The Journal of Machine Learning Research*, 21(1):2648–2720, 2020.

[326] Ashwin Senthilkumar, Mihir Gupte, and S Shridevi. Dynamic spatial-temporal graph model for disease prediction. *International Journal of Advanced Computer Science and Applications*, 13(6), 2022.

[327] Yifan Duan, Guibin Zhang, Shilong Wang, Xiaojiang Peng, Wang Ziqi, Junyuan Mao, Hao Wu, Xinke Jiang, and Kun Wang. Cat-gnn: Enhancing credit card fraud detection via causal temporal graph neural networks. *arXiv preprint arXiv:2402.14708*, 2024.

[328] Haoran Tang, Shiqing Wu, Xueyao Sun, Jun Zeng, Guandong Xu, and Qing Li. Tcgc: Temporal collaboration-aware graph co-evolution learning for dynamic recommendation. *ACM Transactions on Information Systems*, 43(1):1–27, 2025.

[329] Yan Chen, Tian Shu, Xiaokang Zhou, Xuzhe Zheng, Akira Kawai, Kaoru Fueda, Zheng Yan, Wei Liang, and Kevin I-Kai Wang. Graph attention network with spatial-temporal clustering for traffic flow forecasting in intelligent transportation system. *IEEE Transactions on Intelligent Transportation Systems*, 24(8):8727–8737, 2022.

[330] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[331] Yu Zhang and Qiang Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2018.

[332] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE transactions on knowledge and data engineering*, 34(12):5586–5609, 2021.

[333] Wei Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. Automated self-supervised learning for graphs. In *International Conference on Learning Representations*, 2022.

[334] Mingxuan Ju, Tong Zhao, Qianlong Wen, Wenhao Yu, Neil Shah, Yanfang Ye, and Chuxu Zhang. Multi-task self-supervised graph neural networks enable stronger task generalization. In *The Eleventh International Conference on Learning Representations*, 2023.

[335] Roula Nassif, Stefan Vlaski, Cédric Richard, Jie Chen, and Ali H Sayed. Multitask learning over graphs: An approach for distributed, streaming machine learning. *IEEE Signal Processing Magazine*, 37(3):14–25, 2020.

[336] Ron Levie, Wei Huang, Lorenzo Bucci, Michael Bronstein, and Gitta Kutyniok. Transferability of spectral graph convolutional neural networks. *Journal of Machine Learning Research*, 22(272):1–59, 2021.

[337] Luana Ruiz, Luiz FO Chamon, and Alejandro Ribeiro. Transferability properties of graph neural networks. *IEEE Transactions on Signal Processing*, 71:3474–3489, 2023.

[338] Yuxiang Wang, Wenqi Fan, Suhang Wang, and Yao Ma. Towards graph foundation models: A transferability perspective. *arXiv preprint arXiv:2503.09363*, 2025.

[339] Senzhang Wang, Jiannong Cao, and S Yu Philip. Deep learning for spatio-temporal data mining: A survey. *IEEE transactions on knowledge and data engineering*, 34(8):3681–3700, 2020.

[340] Yuxin Qi, Jun Wu, Hansong Xu, and Mohsen Guizani. Blockchain data mining with graph learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(2):729–748, 2023.

[341] Jiaxin Pan, Mojtaba Nayyeri, Osama Mohammed, Daniel Hernandez, Rongchuan Zhang, Cheng Cheng, and Steffen Staab. Towards foundation model on temporal knowledge graph reasoning. *arXiv preprint arXiv:2506.06367*, 2025.

[342] Yuxuan Liang, Haomin Wen, Yutong Xia, Ming Jin, Bin Yang, Flora Salim, Qingsong Wen, Shirui Pan, and Gao Cong. Foundation models for spatio-temporal data science: A tutorial and survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 6063–6073, 2025.

[343] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[344] Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.

[345] Daniel A Spielman. Spectral graph theory and its applications. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE, 2007.

[346] Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022.

[347] Shenyang Huang, Jacob Danovitch, Guillaume Rabusseau, and Reihaneh Rabbany. Fast and attributed change detection on dynamic graphs with density of states. In *Pacific-Asia Conference on Knowledge Discovery & Data Mining*, pages 15–26. Springer, 2023.

[348] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems*, 21(9):3848–3858, 2019.

[349] Bojan Mohar, Y Alavi, G Chartrand, and Ortrud Oellermann. The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, 2(871-898):12, 1991.

[350] Kun Dong, Austin R Benson, and David Bindel. Network density of states. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1152–1161, 2019.

[351] Alexander Weiße, Gerhard Wellein, Andreas Alvermann, and Holger Fehske. The kernel polynomial method. *Reviews of modern physics*, 78(1):275–306, 2006.

[352] TN Kipf. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[353] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[354] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International conference on machine learning*, pages 3734–3743. pmlr, 2019.

[355] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in neural information processing systems*, 33:5824–5836, 2020.

[356] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.