Bridging Artificial Intelligence with Building Information Modeling: Development and Evaluation of a Conversational Assistant for Real-Time Model Management

By

David Fernandes de Oliveira

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

In partial fulfillment of the requirements of the degree of

Master of Science

Department of Civil Engineering

Price Faculty of Engineering

University of Manitoba

Winnipeg

Copyright © 2024 by David Fernandes

Abstract

The integration of Artificial Intelligence (AI) into Building Information Modeling (BIM) has the potential to revolutionize the Architecture, Engineering, and Construction (AEC) industry by enhancing efficiency, improving decision-making processes, and making BIM models more accessible and intuitive for professionals across various disciplines. This thesis introduces DAVE (Digital Assistant for Virtual Engineering), a prototype of a Generative Pre-trained Transformer (GPT)-powered digital assistant designed to facilitate seamless, real-time interactions and updates within BIM environments through natural language processing (NLP) and voice commands. By integrating with Autodesk Revit using Python scripts, the Revit API, and the OpenAI API, DAVE demonstrates a practical application of conversational AI in the AEC sector, aiming to make BIM workflows more intuitive and reducing the cognitive load on users.

The development and implementation of DAVE involve a comprehensive system architecture that combines Python scripting, a Dynamic-Link Library (DLL), and a JavaScript Object Notation (JSON) file for efficient data management and interaction with BIM models. The prototype's testing and validation phase highlights its capability to handle a variety of commands, from simple tasks such as undoing actions to more complex operations like updating room names and numbers. Through a detailed analysis of system performance and user interaction, this thesis explores the challenges and limitations of integrating AI with BIM, including scalability, data integrity, and the quality of user queries. Despite these challenges, DAVE's successful demonstration of real-time BIM model management through voice and natural language commands marks a significant step forward in the digital transformation of the AEC industry.

This thesis not only contributes to the ongoing development of AI applications within BIM but also lays the groundwork for future research aimed at expanding the capabilities of AI-powered

assistants in the AEC sector. By addressing the current system limitations and exploring avenues for further enhancements, such as interoperability with other BIM software and improving the efficiency of information retrieval, the study paves the way for a more connected, intelligent, and user-centric future in the built environment. The development of DAVE embodies the potential of conversational AI to revolutionize architectural and construction methodologies, offering insights into how the AEC industry can leverage AI to achieve greater efficiency, sustainability, and innovation in project management and execution.

Dedication

To my beloved wife, Hingredy,

This thesis stands as a testament to your unwavering faith in me. Your boundless encouragement and steadfast belief in my potential led me to embark on this journey. If not for your initiative to find the university and your gentle, yet persistent nudging to apply, I would not be in the position I am today. Your support became my strength, transforming doubts into determination and dreams into reality. This achievement is not just mine but ours, for you have been my inspiration, and the clarity guiding me through moments of uncertainty. Thank you for being my constant source of motivation and for making me believe that together, we can conquer any challenge. This milestone is a celebration of your love, patience, and sacrifice, and it is with deep gratitude and love that I dedicate this thesis to you.

Acknowledgements

I extend my deepest gratitude to my *Lord Jesus Christ*, who has been my unwavering source of strength and transformation since 2016. His profound impact on my life, changing my character, motivations, and instilling in me a belief in myself and in the goodness of others, is the foundation upon which I stand today. Truly, I am nothing without His grace and guidance, which have blessed me abundantly and shaped the person I have become.

I am immensely grateful to my advisor, Dr. Gursans Guven Isin, for her invaluable insights, guidance, and support throughout this journey. Being her first student at the University of Manitoba has been an honor and a privilege that I cherish deeply. Her belief in my potential and her encouragement have been pivotal in my pursuit of this degree.

I also express my heartfelt thanks to my committee members, Dr. Babak Mehran and Dr. Masoud Asadzadeh, for their valuable insights and contributions to my thesis. Their expertise and feedback have been instrumental in refining my work and guiding me towards academic excellence.

My gratitude extends to my friend and highly skilled writer, Dr. Micael de Andrade-Lima, for his invaluable guidance and advice on academic writing. His insights have significantly enhanced the quality of my work.

Special thanks are owed to Matthew Nikkel for his early contributions to my thesis.

I am also grateful for the financial support provided by the University of Manitoba Graduate Fellowship (UMGF). This support not only alleviated financial concerns but also enabled me to fully immerse myself in my studies and research.

To all who have supported me directly or indirectly on this journey, your contributions have not gone unnoticed, and I am eternally thankful for each one of you.

Table of Contents

Abstract ii

Dedication iv

Acknowledgements v

List of Tables x

List of Figures xi

Abbreviations xii

1. Introduction 1

    1.1. Overview 1

    1.2. Need for Research 4

    1.3. Bridging DAVE and Civil Engineering: A Practical Example 5

    1.4. Objectives of Research 7

    1.5. Scope of the Work 7

    1.6. Thesis Structure 8

2. Literature Review 10

    2.1. BIM Overview 10

    2.2. AI Applications in BIM 12

    2.3. NLP, Chatbots and BIM 14

    2.4. LLM and BIM 15

3. Proposed Prototype 19

3.1. System Architecture 19
3.1.1. Data Extraction Component 21
3.1.2. Python Component 23
3.1.2.1. ChatBotGUI Class (Figure 3.3a) 23
3.1.2.2. OpenAI Assistant Class (Figure 3.3b) 25
3.1.2.3. Voice Recognition and Speaker Classes (Figure 3.3c) 27
3.1.2.4. Supporting Utilities (Figure 3.3d) 27
3.1.3. JSON Bridge Component 28
3.1.4. File Watcher – C# Revit API Component 29
3.1.4.1. Autodesk API – Autodesk.Revit.UI & Autodesk.Revit.DB 31
3.2. System Workflow 32
3.2.1. System Interaction Overview 32
3.2.2. User Input and Initial Processing 33
3.2.3. Command Interpretation and Function Invocation 34
3.2.4. Synchronous Execution and Error Handling 35
3.2.5. Interaction with Autodesk Revit and Specialized Function Handling 36
3.2.6. Completion and Response Formulation 38
4. Validation 40
4.1. Validation Methodology 40
4.2. Validation Results 41

4.2.1. Automated Functionality Testing Results 41
4.2.2. Error Analysis 43
4.2.3. Response Time Analysis 47
5. Discussion 49
5.1. Contributions 49
5.1.1. Advancement of Conversational AI in AEC 49
5.1.2. Customization and Automation in BIM-Authoring Tool 50
5.1.3. Enhancement of BIM Workflow 50
5.1.4. Accessibility and Inclusivity 52
5.2. Challenges and Limitations 52
5.2.1. Scalability, Data Integrity, Privacy and Security 53
5.2.2. Interoperability Challenges 54
5.2.3. Query Quality 55
5.2.4. System Efficiency and Economic Considerations 56
6. Summary, Conclusions and Recommendations 58
6.1. Summary 58
6.2. Conclusions 58
6.3. Recommendations for Future Work 61
7. References 63
Appendix A: *Python Component Code Snippet* 71

A.1. Utility Functions to Manage Bridge Interaction 71

A.2 Examples of Bridge Updating Functions 73

Appendix B: *Bridge Component Code Snippet* 76

B.1. Examples of Bridge JSON Schema 76

Appendix C: *C# Component Code Snippet* 77

C.1. File Watcher Class 77

C.2. Example of Revit Action Method 78

List of Tables

Table 3.1: System components and their roles.

Table 3.2: Python to C# funcitons and methods mapping.

Table 4.1: Error analysis.

List of Figures

Figure 2.1: BIM lifecycle for a construction project (Autodesk, 2024).

Figure 2.2: History of LLMs. Adapted from (MongoDB, 2024)

Figure 3.1: DAVE's system architecture overview.

Figure 3.2: Data Extraction Component of the prototype.

Figure 3.3: Dependency diagram of DAVE's Python component.

Figure 3.4: DAVE's GUI.

Figure 3.5: Assistant DAVE_demo1 setup on OpenAI Playground website.

Figure 3.6: JSON Schema for the function 'change_transpancey' incorporated as a tool.

Figure 3.7: *bridge.json* schema and how it interacts with Python and C# components.

Figure 3.8: DAVE's workflow diagram.

Figure 3.9: SSD of the unrelated or database inquiries process.

Figure 3.10: SSD of the general Revit-action process.

Figure 3.11: Example of the list of functions to be called produced by the GPT Assistant.

Figure 3.12: Example of *bridge.json* schema after a called function run.

Figure 3.13: SSD of the *write* operation process.

Figure 3.14: SSD of the *read* operation process.

Figure 3.15: (a) System logs for the exemplified user request, and (b) result in Revit application.

Figure 4.1: Confusion matrices for Query Type 1 (left) and Type 2 (right).

Figure 4.2: Evaluating query success and failure by function accuracy.

Figure 4.3: Average delay by function and call order.

Figure 5.1: Approximate tokens consumption for a single request.

Abbreviations

AEC	Architecture, Engineering, and Construction
AI	Artificial Intelligence
API	Application Programming Interface
APS	Autodesk Platform Services
BIM	Building Information Modeling
CSV	Comma-Separated Values
DAVE	Digital Assistant for Virtual Engineering
DLL	Dynamic-Link Library
GUI	Graphical User Interface
GPT	Generative Pre-trained Transformer
IoT	Internet of Things
JSON	JavaScript Object Notation
LLM	Large Language Model
ML	Machine Learning
NLP	Natural Language Processing
SSD	System Sequence Diagram

1. Introduction

1.1. Overview

Over the last two decades, the Architecture, Engineering, and Construction (AEC) industry has undergone a substantial change, largely fueled by the integration of Building Information Modeling (BIM) into its core practices. BIM, a digital representation of the physical and operational attributes of constructed assets, has become an indispensable tool, streamlining the creation, management, and utilization of building data (Sacks et al., 2018). This digital shift has not only enhanced collaboration and efficiency but also opened new avenues for innovation in project management and execution. Despite these advancements, the application of BIM within the industry is not without its challenges. The complexity and inherently static nature of BIM models often hinder real-time interaction and adaptability, limiting their potential in dynamic project environments (Enshassi et al., 2019).

Simultaneously, the field of Artificial Intelligence (AI) has evolved, offering new potential for overcoming these challenges. AI, with its diverse subfields such as Machine Learning (ML), Natural Language Processing (NLP), Computer Vision (CV), robotics and optimization, has been instrumental in revolutionizing industries by automating operations, enhancing data analysis, and supporting decision-making processes (Abioye et al., 2021). The introduction of AI in the AEC sector, particularly through the integration with BIM, promises to address BIM's limitations by enhancing its workflows (Pan & Zhang, 2021; Zhang et al., 2022).

The emergence of digital assistants, powered by advancements in NLP, has marked a significant milestone in the intersection of technology and human-computer interaction. The progress in NLP has led to the development of intelligent systems capable of understanding, interpreting, and generating human language in a way that is both meaningful and contextually

relevant (Rampini & Re Cecconi, 2022; A. Saka et al., 2024). In the realm of the AEC industry, these technologies offer unprecedented opportunities for enhancing the accessibility and usability of complex systems like BIM. By enabling natural language interfaces, digital assistants can simplify the interaction with BIM models, making them more intuitive and reducing the technical barriers for professionals across various disciplines.

Large Language Models (LLMs), such as the OpenAI Generative Pre-trained Transformer (GPT) series (OpenAI, 2024b) and Google Gemini (Google, 2024), represent a cutting-edge subset of NLP technologies. These models are trained on vast datasets of text from the Internet, enabling them to generate coherent, contextually relevant text based on the input they receive (Chang et al., 2024). LLMs can understand and produce natural language at a level that is often indistinguishable from human output, making them incredibly powerful tools for a wide range of applications, including but not limited to, conversational agents, content generation, and information extraction. Furthermore, the utility of LLM extends well beyond their ability to generate and comprehend text. These models can be 'fine-tuned' or adapted to excel at highly specific tasks through additional training on targeted datasets (OpenAI, 2024a). This process involves refining a pre-trained model on a smaller, task-specific body of text, which enables the model to develop a deeper understanding of and capability in particular domains or applications. In the context of the AEC industry, LLMs have introduced novel opportunities for information retrieval (W. Y. Lin, 2023; Zheng & Fischer, 2023) and interaction within BIM models (Ghosh et al., 2023; Jang & Lee, 2023), enabling even those without extensive technical expertise to engage with BIM through natural language interfaces.

Despite the promising synergy between AI and BIM, integrating these technologies presents unique challenges. The complexity of BIM models, non-standardized data structures, and

the scarcity of training data for AI models complicate their integration (A. B. Saka et al., 2023). Moreover, the AEC industry faces hurdles in adopting AI at a wider scale, including privacy and security concerns, cultural barriers, and the limitations of existing virtual assistants (Adel et al., 2022).

This study introduces DAVE (Digital Assistant for Virtual Engineering), a prototype of a virtual assistant designed at the convergence of BIM and AI technologies. DAVE leverages the advancements in NLP to provide an intuitive interface between users and BIM models, while facilitating real-time updates, queries, and interactions through voice commands and natural language. This dynamic interaction transforms the traditional, static nature of BIM models, making them more responsive and adaptable to users' needs. By enabling seamless, real-time modifications DAVE has the potential to significantly enhance the decision-making process and operational efficiency in the AEC industry.

The system's architecture combines the flexibility of Python scripting with the efficiency of a Dynamic-Link Library (DLL) and a JavaScript Object Notation (JSON) file to create a responsive system capable of bridging human input with complex BIM model adjustments. By reducing the cognitive load and offering a more accessible user experience, DAVE represents a leap forward in making BIM technologies more user-friendly and efficient. It also addresses the practical needs of AEC professionals who often grapple with the complexity of interacting with BIM models.

By addressing the existing gaps in BIM interaction and demonstrating the feasibility of integrating conversational AI, this study contributes to the ongoing digital transformation of the AEC industry. DAVE emerges not merely as a tool but as a partner in the construction process, embodying the potential of AI to revolutionize architectural and construction methodologies. In

doing so, this research paves the way for a more connected, intelligent, and user-centric future in the built environment, marking a significant step towards realizing the full potential of AI and BIM integration.

The principal contributions of this thesis are centered on the development and validation of a novel framework that integrates conversational AI, specifically by leveraging GPT, into BIM to enhance project management and execution within the AEC industry. These contributions are manifested in several key areas: (1) the development of a practical application that bridges conversational AI and BIM, enabling dynamic updates and queries within BIM models; (2) the demonstration of the feasibility of real-time BIM model management through voice and natural language, significantly reducing the complexity and technical barriers traditionally associated with BIM; and (3) the laying of foundational work for future research directions aimed at expanding the capabilities and addressing the limitations of AI-powered assistants in BIM processes.

1.2. Need for Research

The need for research in the integration of AI into BIM practices, particularly for real-time actions on authoring tools emerges from a gap in current technological applications. While the applications of AI and BIM have been expanding, with several chatbots and AI tools developed to enhance information retrieval and project database presentation, there exists a significant opportunity in leveraging AI for real-time modifications and interactions within BIM environments.

Such an integration is crucial for several reasons:

- **Enhanced Efficiency**: Real-time AI interactions can significantly reduce the time required for model adjustments, streamlining the workflow for BIM professionals.

- **Interactive Design**: Incorporating AI into the design phase can facilitate a more interactive design process, where changes can be made on-the-fly, allowing for immediate visualization and assessment.
- **Customization and Flexibility**: A GPT-powered assistant tailored to the specific needs of BIM professionals can offer unprecedented levels of customization and flexibility, adapting to various project requirements and user preferences.
- **Accessibility and Inclusivity**: Simplifying complex BIM tasks through AI can make these tools more accessible to professionals with varying levels of expertise, democratizing the design and modification process.

By focusing on real-time modifications and leveraging the capabilities of LLMs and the Revit API, this study proposes a forward-looking approach that could revolutionize how professionals interact with BIM tools, making the design and modification process more efficient, flexible, and user-friendly.

1.3. Bridging DAVE and Civil Engineering: A Practical Example

In the realm of Civil Engineering, merging AI with BIM through solutions such as DAVE marks a crucial step forward in tackling challenges unique to the industry. Civil Engineering projects, characterized by their complexity and the critical need for precision in planning and execution, stand to benefit significantly from the capabilities offered by DAVE.

As a practical example, the task of material quantity takeoff can be considered, a fundamental yet time-consuming activity in civil engineering. Traditionally, this involves manually reviewing construction drawings and specifications to estimate the quantities of materials required for a project. This process is not only labor-intensive but also prone to errors,

which can lead to significant cost overruns and project delays.

Without an intelligent system like DAVE, civil engineers face several challenges:

- **Manual Data Extraction**: Extracting data from complex BIM models can be cumbersome and error-prone.
- **Time-Consuming Calculations**: Estimating material quantities and costs manually is time-consuming, reducing operational efficiency.
- **Difficulty in Adapting to Changes**: Any change in design requires recalculating material quantities and costs, a process that can significantly delay project timelines.

By leveraging DAVE, these tasks can be transformed. DAVE, through its real-time interaction with Revit, has the potential to automate the process of generating material quantity takeoffs. By developing specific functions (explained more in Section 3.1.2.2) for these specific requirements, it can:

- **Automate Quantity Takeoffs**: Quickly calculate the quantities of materials directly from the BIM model, reducing manual work and potential errors.
- **Update Cost Estimates in Real-Time**: Check the latest prices of materials online and automatically update the cost estimates, ensuring budget accuracy.
- **Assess Impact of Changes**: Instantly evaluate the effect of any change in the model on material quantities and costs, facilitating agile project management.

This example underscores DAVE's potential to revolutionize Civil Engineering practices by enhancing accuracy, efficiency, and adaptability in project management and execution. Such advancements not only address the immediate challenges faced by civil engineers but also pave the way for more innovative and resilient infrastructure development.

1.4. Objectives of Research

This research aims to bridge the gap between the potential of AI in enhancing BIM workflows and the current state of its application. The primary goal of this study is to develop and evaluate a prototype of a chatbot for real-time updates and interactions within BIM environments. This innovative approach aims to enhance the efficiency and effectiveness of BIM workflows by leveraging the capabilities of an LLM. The research aims to answer the following specific questions:

- **Integration**: *How can an AI chatbot be integrated with a BIM authoring tool to enable real-time data exchange and updates effectively, thereby streamlining project workflows?*
- **Evaluation**: *In what ways is the used LLM effective in interpreting and executing BIM-related queries?*
- **Performance Assessment**: *How does the prototype perform when handling simple versus compound queries within the BIM context?*
- **Exploration**: *What potential does the prototype hold in improving project development, management, execution, collaboration, and decision-making in BIM projects?*

1.5. Scope of the Work

This study leverages the use of AI, NLP, and a specific LLM model (i.e., GPT), to enhance BIM workflows. The developed prototype integrates with a specific, yet widely adopted, BIM-authoring software (i.e., Autodesk Revit), showcasing the practical application of these technologies in real-world BIM tasks. The prototype can perform 12 actions within the BIM

environment – each one selected based on the daily tasks performed by a typical Virtual Design and Construction department in a construction company and mainstream uses of Revit – ranging from simple tasks like undoing the last action to more complex ones such as changing the family type of a selected element, and updating room names, numbers, and occupants. It utilizes a prepared comma-separated values (CSV) file with the project database for function arguments and project queries. The scope of work is justified by the need to explore the integration of cutting-edge AI technologies with established BIM processes, aiming to uncover potential efficiencies and improvements in project development, management, and execution. The prototype's testing utilizes both a sample model in Revit and a real-world office building model, covering all disciplines. The type of BIM project does not restrict the prototype's applicability which demonstrates its versatility across different project requirements. The current limitations of the prototype and some specific challenges, such as scalability and system efficiency, are discussed in Chapter 5 and areas for future improvements and research are identified.

1.6. Thesis Structure

The structure of this thesis is outlines as follows:

- **Chapter 1**, Introduction, presents the research topic, outlining the motivation behind integrating AI with BIM, the objectives pursued through this research, and the scope of the work. This chapter sets the foundation for understanding the significance and the innovative approach of the research.
- **Chapter 2**, Literature Review, delves into the existing body of knowledge, covering the evolution of BIM and AI integration, the role of NLP and chatbots in BIM, and the potential of LLMs in enhancing BIM functionalities. This chapter provides a critical analysis of the

current state of research and identifies the gaps that this thesis aims to address.

- **Chapter 3**, Proposed Prototype, details the system architecture of the prototype, including its components and their functionalities. It explains the system workflow, from data extraction and processing to user interaction and the execution of BIM tasks. This chapter emphasizes the innovative aspects of prototype, highlighting its potential to transform BIM model management and interaction.

- **Chapter 4**, Testing and Evaluation, presents the methodology adopted for testing the prototype, including automated and manual testing procedures, and discusses the results obtained. This chapter provides an in-depth analysis of the system's performance, its strengths, and areas for improvement, validated through systematic testing and evaluation.

- **Chapter 5**, Discussion, reflects on the findings, discussing the implications of prototype development for the AEC industry. It explores the system's strengths, challenges, and limitations, offering insights into the practical application and potential impact of integrating conversational AI within BIM workflows.

- **Chapter 6**, Summary, Conclusion, and Recommendations, summarizes the key findings of the research, draws conclusions about the significance and impact of the prototype, and outlines recommendations for future work. This chapter highlights the contribution of the thesis to the field and proposes directions for further research to advance AI applications in BIM.

2. Literature Review

The integration of AI into BIM signifies a crucial evolution in the AEC industry. This literature review covers the trajectory of AI applications within BIM, focusing on the contributions of chatbots, NLP, and LLMs applications in enhancing BIM processes. The review first provides an overview about BIM (Section 2.1), then it looks at the general applications of AI in BIM (Section 2.2), followed by a discussion of the specific roles of chatbots and NLP (Section 2.3) and finally examines the innovative uses of LLMs (Section 2.4) in facilitating real-time interactions and management within BIM environments.

2.1. BIM Overview

BIM represents a paradigm shift in the AEC industry, introducing a digital-first approach to the design, construction, and management of buildings and infrastructure. Defined broadly, BIM is a process supported by various tools, technologies, and contracts involving the generation and management of digital representations of physical and functional characteristics of places (Sacks et al., 2018). These digital models are used throughout the entire lifecycle of a project, from initial planning through design, construction, operation, and maintenance.

The concept of BIM has evolved significantly over the past few decades, with its roots traced back to the 1970s. However, it was not until the early 2000s that BIM began to gain widespread recognition in the AEC industry. The evolution of BIM technology has been closely linked to advancements in computing power and software development, enabling more sophisticated modeling capabilities and collaborative workflows (Succar, 2009).

BIM offers numerous advantages over traditional drafting methods, including enhanced visualization, improved collaboration, and increased efficiency. By using BIM, stakeholders can

better predict project outcomes, resolve conflicts before construction begins, and make more informed decisions throughout the project lifecycle. Additionally, BIM facilitates the creation of a shared knowledge resource for information about a facility, which supports decision-making about a building from the earliest conceptual stages through to its operational life (Azhar, 2011).

Despite its benefits, the adoption of BIM poses several challenges. These include the need for significant initial investment in software and training, the requirement for changes in traditional workflows, and issues related to interoperability among different BIM software applications. Moreover, there are concerns about data security, especially when sharing models across various stakeholders (Becerik-Gerber & Rice, 2010).

BIM's impact on the AEC sector is profound, offering a collaborative environment that fundamentally changes how projects are constructed and managed. The use of BIM goes beyond mere modeling; it involves the management of information throughout the lifecycle of a building, encouraging a more integrated and holistic approach to building design and construction, as illustrated in **Figure 2.1** by (Autodesk, 2024).

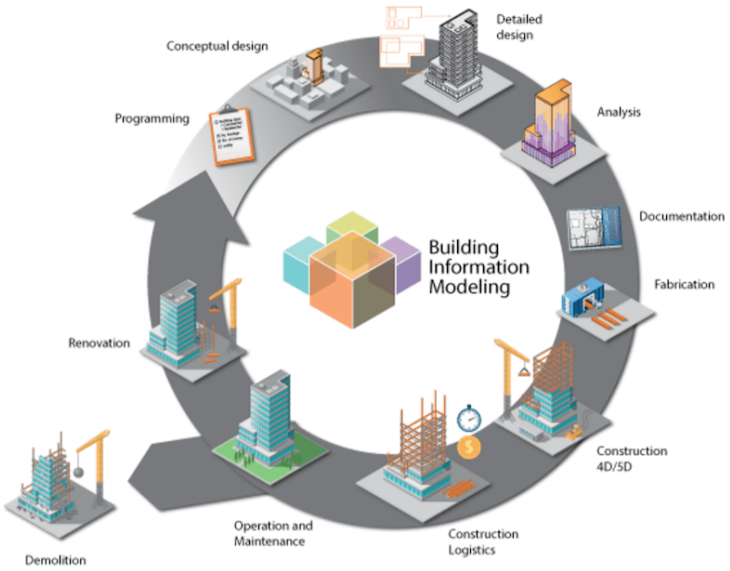


Figure 2.1: BIM lifecycle for a construction project (Autodesk, 2024).

The future of BIM is likely to see further integration with emerging technologies such as AI, ML, and the Internet of Things (IoT). These integrations promise to enhance the capabilities of BIM, enabling more predictive and adaptive building models that can lead to smarter and more sustainable built environments (Sacks et al., 2018).

2.2. AI Applications in BIM

The evolution of AI technologies, for instance, in ML and CV, has opened new avenues for enhancing BIM functionalities. ML algorithms have been increasingly applied to BIM databases for predictive analytics, facilitating more informed decision-making regarding project timelines, cost estimations, and resource allocation (Zabin et al., 2022). CV, on the other hand, has been instrumental in enhancing the interoperability between BIM models and real-time construction site data, enabling more accurate monitoring and progress tracking (Xu et al., 2021).

Recent literature has highlighted several workflow improvements attributed to the application of AI in BIM. A prominent example is the use of AI for automating clash detection in BIM models, which has significantly reduced manual labor while increasing the accuracy in identifying potential conflicts in project designs (Ahmadpanah et al., 2023). Furthermore, AI has been leveraged to enhance building energy performance analysis by automating simulation processes within BIM tools, thereby promoting the development of more sustainable and energy-efficient building designs (W. Wang et al., 2022). Another application includes the use of AI for automatic material selection, aligning choices with the current progress of the project to optimize resource allocation (Stephans et al., 2022). Additionally, AI's role extends to schedule generation and optimization, where it streamlines project timelines and improves efficiency (Singh & Anumba, 2022).

Furthermore, the integration of IoT with AI and BIM technologies has introduced a new dimension to real-time data collection and analysis in the AEC sector. IoT devices, such as sensors and actuators embedded within buildings, facilitate the continuous monitoring of structural health, environmental conditions, and energy consumption (Orooje & Latifi, 2021). This real-time data, when analyzed with AI algorithms, can significantly improve the maintenance and operation phases of facility management, enabling predictive maintenance strategies that pre-emptively address potential issues before they escalate (Pedral Sampaio et al., 2022).

The application of AI across all stages of the AEC industry — from design and construction to facility management — illustrates the comprehensive impact of these technologies. During the design phase, AI-driven generative design algorithms can explore a vast array of design alternatives based on predefined criteria, optimizing for factors such as spatial efficiency, material usage, and environmental impact (Ghosh et al., 2023; Jang & Lee, 2023; A. Saka et al., 2024). In the construction phase, AI has proven instrumental in progress tracking and quality control (Abioye et al., 2021). Finally, in facility management, AI applications extend to energy management, predictive maintenance models and automatic as-built classification (Pedral Sampaio et al., 2022).

Despite these advancements, the integration of AI in BIM is not without challenges. Issues related to data quality, model interoperability, and the need for large datasets for training AI models remain significant barriers to the widespread adoption of AI-enabled BIM (Zhang et al., 2022). Furthermore, concerns regarding data privacy and security have emerged as critical considerations, given the sensitive nature of the data involved in BIM processes (Abioye et al., 2021).

Future research in this domain is poised to address these challenges, focusing on

developing more robust AI algorithms that can operate with limited data and ensuring the security of data within AI-enabled BIM applications. Additionally, efforts are underway to enhance the interoperability of AI and BIM across different software platforms, aiming to create a more integrated and seamless workflow within the AEC industry (Darko et al., 2020).

2.3. NLP, Chatbots and BIM

NLP technologies facilitate the understanding, interpretation, and generation of human language by computers, allowing for a more natural interaction between users and BIM systems (Ding et al., 2022). When integrated with BIM, NLP enables users to perform complex queries, extract information, and even modify models using simple, conversational language. This has dramatically reduced the learning curve associated with BIM software, making it more accessible to a broader range of users (Erfani & Cui, 2021).

Chatbots, powered by NLP algorithms, offer a conversational interface through which users can interact with BIM databases. These AI-driven assistants can interpret user queries, retrieve relevant information from BIM models, and even execute commands to update models based on user inputs. Recent developments have seen chatbots being employed for various tasks within BIM, such as data/information retrieval (W. Y. Lin, 2023; Nabavi et al., 2023; N. Wang et al., 2022a, 2022b; Zheng & Fischer, 2023), construction contract summarization (Xue et al., 2022), compliance check (Wu et al., 2023), real-time project updates (Ghosh et al., 2023; Jang & Lee, 2023), question and answer systems (T. H. Lin et al., 2022) and live instructor for safety training (Hussain et al., 2024), showcasing the versatility and potential of chatbots in streamlining BIM processes.

The integration of NLP and chatbots into BIM offers several notable benefits. Firstly, it

enhances the efficiency of data management within BIM by allowing for quick and easy access to information through natural language queries. This reduces the time spent navigating complex software interfaces and increases productivity (Locatelli et al., 2021). Secondly, it democratizes access to BIM, enabling stakeholders with varying levels of technical expertise, including those without formal training in BIM software, to effectively interact with and utilize BIM data. This directly addresses a significant challenge in BIM implementation: the steep learning curve associated with mastering BIM software (Sacks et al., 2018). Lastly, the use of chatbots for routine tasks and queries can free up human resources for more complex and creative aspects of project management and design (A. B. Saka et al., 2023).

Despite the promising advancements, the application of NLP and chatbots in BIM has its challenges (Chung et al., 2023). One significant limitation is the complexity of accurately interpreting the vast variety of natural language inputs, especially when dealing with technical terminology specific to the AEC industry. Ensuring chatbots can understand and respond to such inputs accurately requires extensive training and continuous learning (Ding et al., 2022).

Another challenge is to integrate chatbots seamlessly with existing BIM software and workflows. This often requires custom development and may encounter resistance from users accustomed to traditional interfaces. Furthermore, concerns regarding data security and privacy arise when using cloud-based NLP services, necessitating robust security measures to protect sensitive project information (A. B. Saka et al., 2023).

2.4. LLM and BIM

The journey of LLMs began with the inception of ELIZA in 1967, a pioneering computer program that demonstrated the feasibility of machines understanding and mimicking human

conversation (A. B. Saka et al., 2023). This early experiment laid the groundwork for subsequent advancements in the field of NLP as illustrated in **Figure 2.2** from (MongoDB, 2024) . Over the decades, NLP research has seen significant breakthroughs, culminating in the development of the transformer architecture by Google in 2017 (Vaswani et al., 2017). This novel approach revolutionized the way machines understand text, enabling the creation of more sophisticated and powerful models.

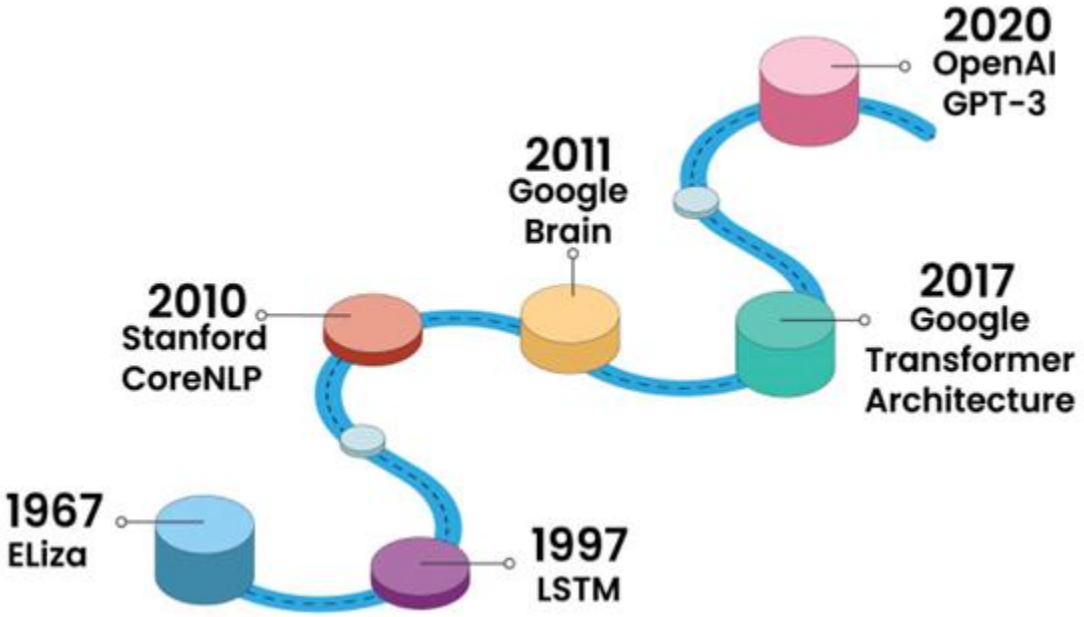


Figure 2.2: History of LLMs. Adapted from (MongoDB, 2024)

LLMs like GPT from OpenAI, Gemini (formerly known as Bard) from Google, RoBERTa from Meta (Liu et al., 2019), and Megatron-LM from Nvidia (Shoeybi et al., 2019) have been at the forefront of AI research due to their ability to understand, generate, and interpret human language with remarkable accuracy (Chang et al., 2024). In the context of BIM, LLMs are being explored for their potential to automate complex tasks, provide intelligent assistance to users, and facilitate more intuitive interactions with BIM systems. By training these models on a diverse range of construction-related datasets, researchers have begun to tap into their potential for

streamlining BIM workflows, enhancing project communication, and improving decision-making processes (N. Rane et al., 2023; N. L. Rane et al., 2023).

The integration of LLMs into BIM is still in its early stages, but significant advancements have already been noted such as ingenerating innovative design alternatives, streamlining communication among stakeholders, and optimizing construction schedules (Prieto et al., 2023; N. Rane et al., 2023; N. L. Rane et al., 2023). The capacities of this models to aid in education and training are also being investigated (Uddin et al., 2023). Recent frameworks, such as the interactive design automation framework (DAF), leverage past data and LLMs for automating design tasks across various engineering disciplines, demonstrating significant improvements in efficiency and quality of deliverables (Ghosh et al., 2023). Additionally, the integration of LLMs with BIM through tools like BIM2XML and GAIA showcases the potential for dynamic collaboration in architectural design, reinforcing the transformative impact of these technologies (Jang & Lee, 2023).

Innovative frameworks are central to harnessing the full potential of LLMs in BIM. The proposed frameworks focus on interoperability, data consistency, and user-friendly interfaces, ensuring seamless integration of AI with BIM processes (Ghimire et al., 2024; Jang & Lee, 2023; N. Rane et al., 2023; N. L. Rane et al., 2023). These initiatives aim to replace conventional design practices with intelligent, data-driven methodologies, highlighting the importance of leveraging past project data and advanced programming for design automation and optimization (Ghosh et al., 2023).

Despite the promising integration of LLMs with BIM, several challenges persist. Data privacy, ethical considerations, and the demand for extensive computational resources are among the primary concerns (N. Rane et al., 2023; N. L. Rane et al., 2023). Additionally, the construction

industry faces specific challenges related to trust, acceptability, and the technical nuances of domain-specific applications (A. Saka et al., 2024). LLMs also encounter limitations like hallucinations and reliability issues, which necessitate ongoing research and development to enhance their accuracy and applicability in construction contexts (A. Saka et al., 2024).

Nevertheless, the application of LLM in BIM holds great promise for transforming the AEC industry by enhancing efficiency, improving data management, and democratizing access to BIM technologies. As these models continue to evolve and become more sophisticated, their integration into BIM processes is expected to deepen, offering even more innovative solutions to complex challenges.

3. Proposed Prototype

3.1. System Architecture

Figure 3.1 illustrates the architecture of the proposed AI-assisted BIM system, its components, and their links. It is a blend of various technologies for smart, real-time interactions within a well-known and commonly used BIM authoring tool, Revit. The system possesses a preliminary stage called Data Extraction, further explained in Section 3.1.1. During this stage, elements' metadata are extracted from the Revit project file (*.rvt*) and converted to a JSON model derivatives and finally into the CSV format for efficient information retrieval.

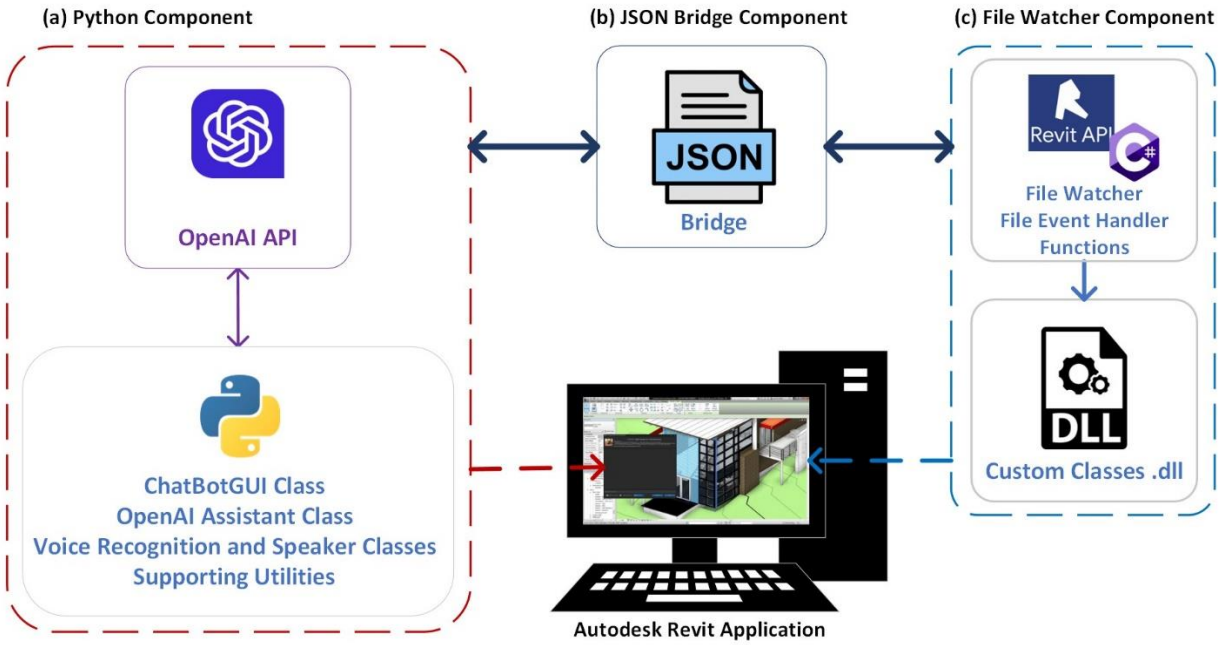


Figure 3.1: DAVE's system architecture overview.

The developed system consists of three main components as illustrated in **Figure 3.1**. The core of the prototype lies in the Python component (**Figure 3.1a**), which is empowered by OpenAI Application Programming Interface (API) and enables smart user interactions, including Text-to-Speech and Speech-to-Text features for voice responses. Users engage with the system via a

custom Graphical User Interface (GUI) that runs along with the Revit application. More details are provided in Section 3.1.2. The JSON Bridge Component (**Figure 3.1b**) is a key mechanism enabling real-time communication between the OpenAI Assistant Class and Revit. Structured around the *bridge.json* file, it seamlessly integrates Python and C# within BIM workflows, supporting essential actions with minimal resource use. This component is expanded upon in Section 3.1.3. Finally, the File Watcher component (**Figure 3.1c**) is crafted in C# and dynamically monitors changes on the bridge file, coordinating with the Python AI component through the Revit API and custom DLL classes. Its capabilities and structure are further detailed in Section 3.1.4. All the system components and their roles are summarized in **Table 3.1**.

Table 3.1: System components and their roles.

Component	Description	Role in the System
Data Extraction Component	Extracts and preprocesses data from Revit files, converting them into a structured format (JSON/CSV) for easy access by the GPT Assistant.	Powers the GPT Assistant with project-specific data retrieval capabilities for customized interactions.
Python Component	Manages user interactions, processes natural language commands, and communicates with the OpenAI API. Provides a visual interface for users to interact with DAVE, facilitating both text and voice command input.	Facilitates natural language processing, command execution, and voice recognition features.
JSON Bridge Component	A JSON file that acts as an intermediary for communication between the Python script and the C# component.	Ensures real-time update and action execution within Revit by transmitting commands and receiving status updates.
C# Revit API Component	Monitors changes in the *bridge.json* file and executes corresponding actions within Autodesk Revit.	Triggers updates or modifications in the Revit model based on instructions decoded from the *bridge.json* file.

3.1.1. Data Extraction Component

As depicted in **Figure 3.2**, the data extraction component is a crucial aspect of the project, designed to empower the AI assistant with information retrieval capabilities. This component focuses on the preprocessing of data and its integration into the assistant framework. The process initiates with the extraction of data from the Revit files using the Autodesk Platform Services (APS) API. This data that is stored initially in a cloud-based bucket, consists of model derivative and tree information, and is downloaded as JSON files. These files are then cleaned and manipulated into a singular CSV file, which is seamlessly integrated into the AI assistant. This approach ensures each assistant is closely linked to a specific project database, enabling customized interactions and responses.

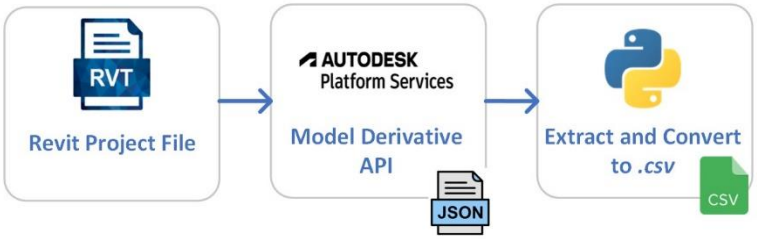


Figure 3.2: Data Extraction Component of the prototype.

In the GUI, users select the appropriate assistant based on the name of the project while the core instructions for the assistant remain consistent across different projects, ensuring a standardized user experience. The current version of DAVE is configured for two distinct projects: Demo1 and Demo2. Demo1 is the standard sample architectural project in Revit 2023, while Demo2 represents a more complex scenario that features a real-world office building in Manitoba, Canada with multiple discipline models including architecture, structural, mechanical, and electrical.

The main part of the data extraction is performed using a Python script that processes JSON

data from two files, *tree.json* and *properties.json*, generated by the APS API. The script navigates the hierarchical tree structure, extracting pertinent object data and concurrently cleans and filters the properties to remove superfluous or redundant information. The data is organized with the Industry Foundation Classes (IFC) Globally Unique Identifier (GUID) as the main key. This strategic choice compensates for the lack of Revit element IDs in the model derivative, aiding the AI assistant in accurately retrieving element-specific data for functionalities (such as *selected_element_info* or *room_info*). The script further refines the dataset, eliminating irrelevant or empty properties and adjusting specific element attributes. The processed data is then saved into a new JSON file named *extracted_data.json*, preparing it for the subsequent conversion stage.

The conversion from JSON to CSV format is executed through a secondary Python script, which flattens the nested JSON structures, primarily focusing on the *properties* key. The script processes each entry, amalgamating key information like *ifcGUID*, *Name*, *Family Type*, *Family Name*, and *Category* with the flattened properties. The resultant structured data is then saved as a CSV file, ensuring a coherent and accessible data format.

Finally, the CSV file, encapsulating the processed project data, is integrated into the AI assistant during its creation using the OpenAI API or OpenAI Playground website. The assistant is set up with the code interpreter tool enabled. This tool, in assistants API context, executes Python code by itself in a secure, sandboxed environment, allowing for data processing (OpenAI, 2024d). It supports iterative code execution and enables problem-solving through trial and improvement. This setup ensures that each assistant aligns with the specific nuances of a given project while maintaining the ability to execute uniform instructions across various projects. Consequently, the assistant is equipped to adapt its responses and functionalities based on the unique data and requirements of each project, significantly enhancing the precision and relevance.

3.1.2. Python Component

The Python component (**Figure 3.1a**) constitutes of primary classes that control its core functionalities and utility functions divided into eight files. The main classes include a GUI, the OpenAI Assistant (custom class to handle API requests and responses), and the voice recognition and speaker handling. This project was written in Python 3.12. Each class plays a pivotal role in the system's operation, seamlessly integrating with the overall architecture to provide a cohesive and efficient user experience. **Figure 3.3** presents how the different modules (i.e., Python files), and classes interact with each other within the DAVE system.

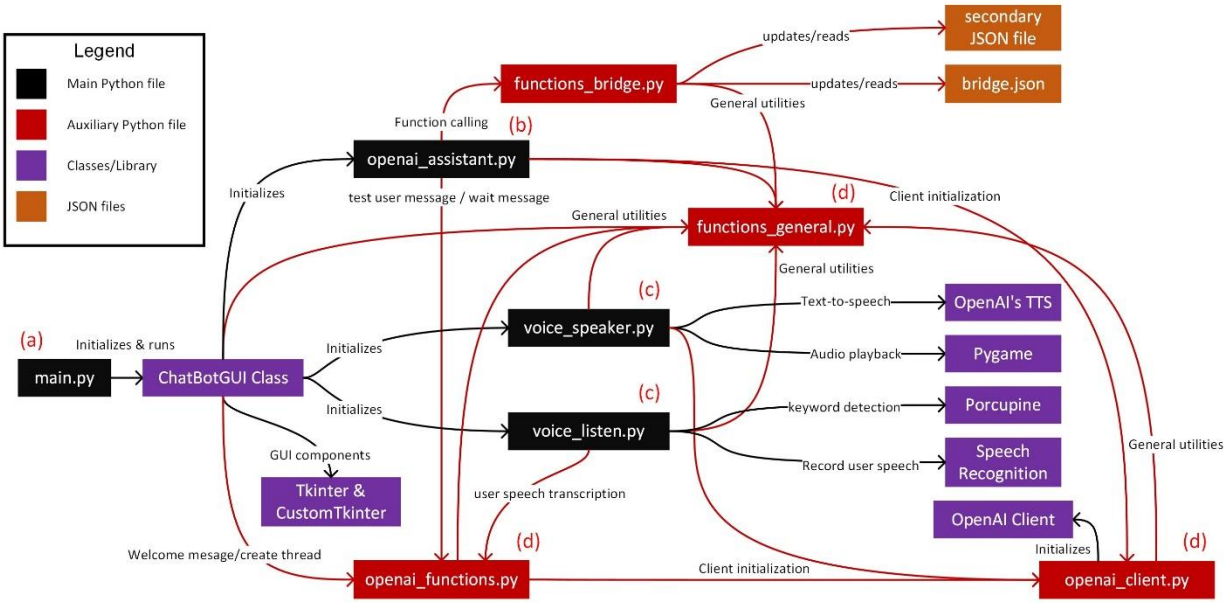


Figure 3.3: Dependency diagram of DAVE's Python component.

3.1.2.1. ChatBotGUI Class (Figure 3.3a)

The GUI class, the spine of the DAVE system and its front-end, is a user interface developed using *customtkinter* and *tkinter* Python modules. This class is responsible for the visual representation of the prototype, presenting a user-friendly platform for interaction. The class

initiates the OpenAI Assistant class and handles the interactions between GUI and the API. The class also handles the integration of audio systems, enabling voice recognition and text-to-speech functionalities by instantiating their respective classes. The GUI (**Figure 3.4**) is designed to be intuitive, allowing users to interact with Revit through both text and voice commands.

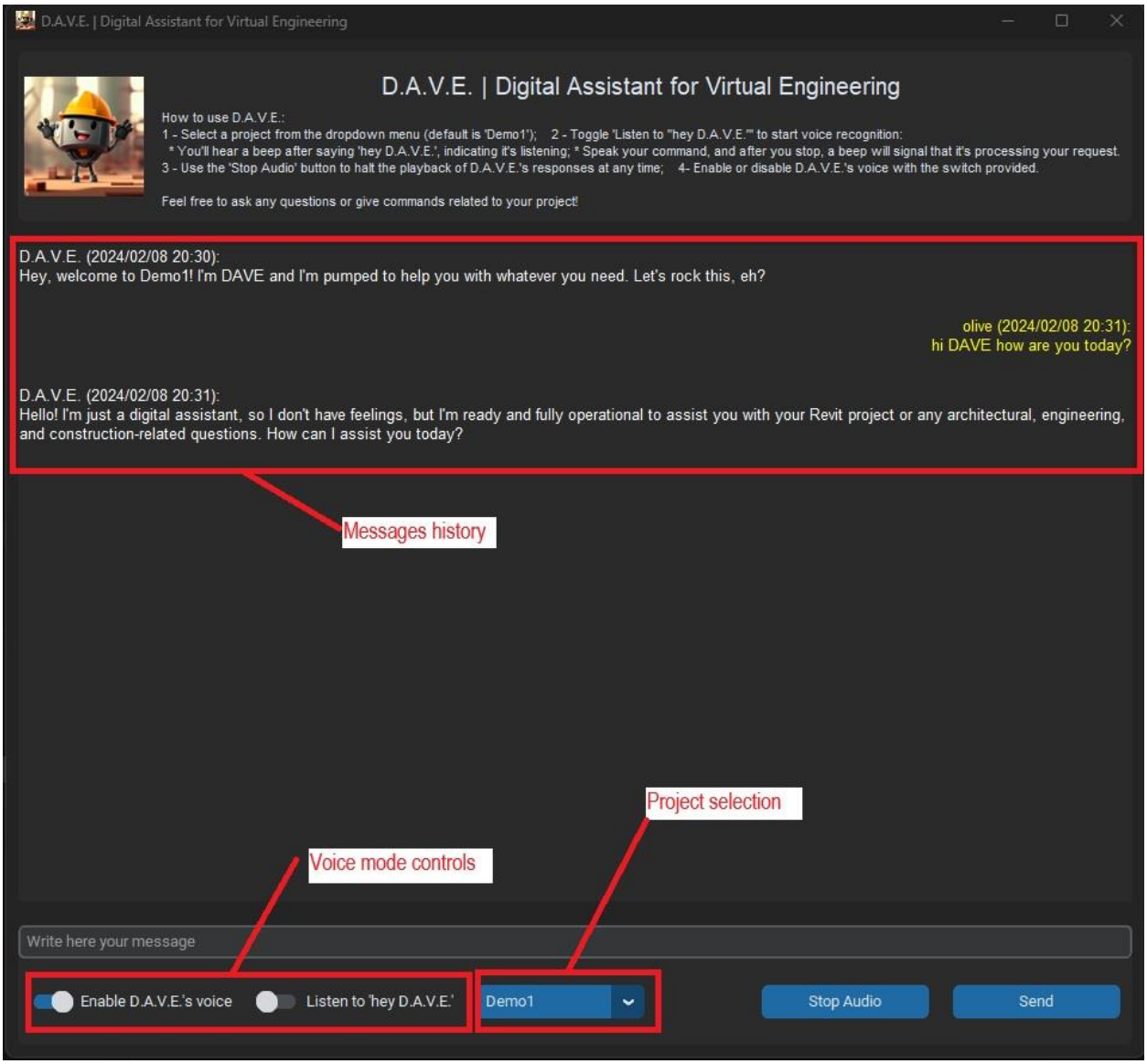


Figure 3.4: DAVE's GUI.

3.1.2.2. OpenAI Assistant Class (Figure 3.3b)

Central to the Python component is the OpenAI Assistant class (located in the *openai_assistant.py*), the system back-end, which orchestrates the interaction between the user and the AI model. This class leverages the GPT-4-1106-preview model (GPT-4 Turbo) by OpenAI, and other features such as creating API threads, handling user messages, and running GPT Assistants. The class is equipped to process and respond to user queries by dynamically calling functions from the *functions_bridge.py* script, which acts as a conduit for updating the *bridge.json* file. Each Python function interacts with a C# counterpart, as shown in **Table 3.2**. To be able to call the right function, all of them are incorporated in the GPT Assistant during its creation (**Figure 3.5**), each function is set up using a JSON schema as illustrated in **Figure 3.6** for the *change_transparency* function. The class is adept at handling complex queries, aggregating similar function calls for enhanced efficiency, and managing the nuances of user interactions, whether through text or voice commands.

Table 3.2: Python to C# functions and methods mapping.

C# Method Name	***Description***	***Triggered by (Python Function)***	***Python Arguments***
Transparency	Applies transparency to elements or categories	change_transparency	transparency_value (int) , mode (str) , items (list)
Isolation	Isolates elements or categories	isoalte	mode (str), items (list), reset (bool)
Hide	Hides elements or categories	hide	mode (str), items (list), hide_mode (str), reset (bool)
Color	Applies color to elements or categories	set_color	mode (str), items (list), color (str), reset (bool)
Tag	Tags elements in specified categories	tag	category (list)
DeleteElement	Deletes specified elements	delete_element	mode (str), items (list)
ElementData	Reads/writes element data	selected_element_info	operation (str), key (str), user_choice (str)
RoomData	Reads/writes room data	room_info	operation (str), key (str), room_number (str), room_name (str), occupant (str)
ViewCreation	Creates various types of views	create_view	type (str), level (str), view_name (str)
ViewDuplication	Duplicates views	duplicate_view	dependent (bool), detailing (bool), view_name (str)
RenameView	Renames the current view	rename_view	new_name (str)
Undo	Executes an undo action	undo	None (No arguments)

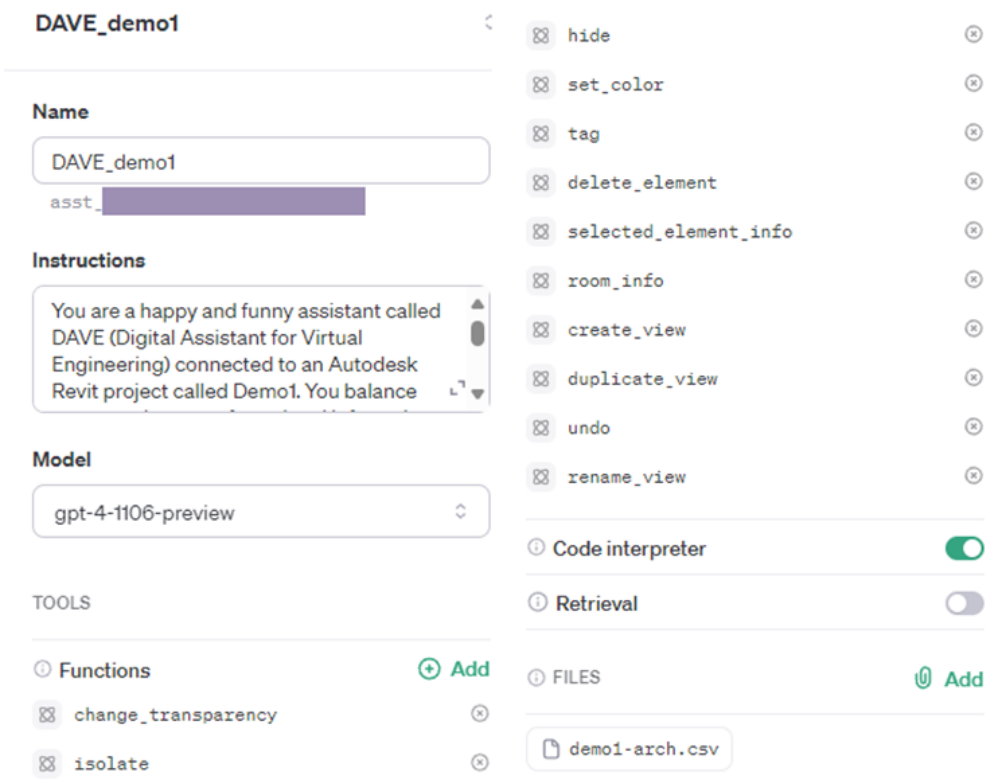


Figure 3.5: Assistant DAVE_demo1 setup on OpenAI Playground website.

```
{
  "name": "change_transparency",
  "description": "change the transparency of one or more Revit categories, or elements (by current selection or providing element IDs) in the current view",
  "parameters": {
   "type": "object",
   "properties": {
    "transparency_value": {
     "type": "integer",
     "description": "The transparency value to set"
    },
    "mode": {
     "type": "string",
     "enum": ["selection", "element", "category"],
     "description": "The mode of application for transparency"
    },
    "items": {
     "type": "array",
     "items": {"type": "string"},
     "description": "A list with one or more element IDs or one or more Revit categories to apply transparency. (e.g., ['Walls', 'Windows', 'Doors',...])"
    }
   },
   "required": ["transparency_value", "mode"]
  }
 }
```

Figure 3.6: JSON Schema for the function ‘change_transpancey’ incorporated as a tool.

3.1.2.3. Voice Recognition and Speaker Classes (Figure 3.3c)

The voice recognition and speaker aspects of the prototype are handled by two distinct classes: the *ListenManager* and the *SpeakerManager*. The *ListenManager* class, using libraries such as *pvporcupine* and *speech_recognition*, is designed to detect the activation keyword "*hey Dave*", record user speech, and transcribe it into text using OpenAI's model *whisper-1*. This class is a testament to the system's accessibility, offering an alternative mode of interaction for users. The *SpeakerManager* class, on the other hand, employs OpenAI's *tts-1* model to convert text responses into speech, facilitating an engaging and interactive user experience. The integration of these classes with the main GUI and the OpenAI Assistant underscores the system's commitment to providing a comprehensive and user-friendly interface.

3.1.2.4. Supporting Utilities (Figure 3.3d)

In addition to the main classes, the prototype incorporates several supporting utilities, enhancing the functionality and user experience of the Python scripts. Among these, the *functions_general.py* script offers a comprehensive suite of utility functions, such as JSON data management and system alerts, facilitating efficient data handling and user communication. The *openai_client.py* script plays an essential role in initializing the main OpenAI class from its module, ensuring a seamless integration with OpenAI's services for enhanced operational efficiency.

Furthermore, the *openai_functions.py* script is equipped with specialized functions tailored for interacting with the OpenAI API. It includes capabilities for thread creation, test user queries, and the generation of various messages (e.g., welcome messages, answers to unrelated queries, and "please wait" messages). This script is particularly noteworthy for its implementation of test

user queries. To optimize the use of tokens—a crucial resource since each interaction with the assistant consumes a number of tokens, for instance, around 3,500 tokens to load the instructions—it was employed a method known as "few-shot learning" (to learn from a limited number of examples). This approach allows the prototype to pre-classify user queries by making an additional API call to OpenAI, utilizing the GPT-3.5-turbo-1106 model. By doing so, it can be efficiently determined whether a query is related or unrelated to the Assistant's scope, while significantly conserving token usage as well as maintaining high accuracy and relevance in responses. These components, from message templates to context-aware algorithms, ensure that the assistant's communication is engaging, less robotic, and more in tune with the user's needs.

3.1.3. JSON Bridge Component

The JSON Bridge Component, as illustrated in **Figure 3.1b**, serves as a crucial mechanism for real-time, bidirectional communication between the GPT Assistant and Revit (through their encapsulating components: Python and File Watcher C#). This component is structured within a JSON file, termed *bridge.json*, which offers key benefits: it ensures readability and ease of editing, promotes efficient processing with minimal resource use, and supports flexible and scalable configurations. Its cross-platform compatibility between Python and C# enhances seamless integration, making it an ideal choice for facilitating dynamic, real-time communication within BIM workflows.

The file is structured as a collection of keys representing different actions (e.g., "transparency", "isolation", "hide"), the *bridge.json* schema is designed to encompass all 12 functions presented previously on **Table 3.2**. Each key within the bridge is associated with specific parameters detailing the action's mode, execution state (run), and potential target items or values.

Figure 3.7 illustrates how the JSON file interacts with the Python and C# components, presenting a number of examples of parameters keys values for *transparency*.

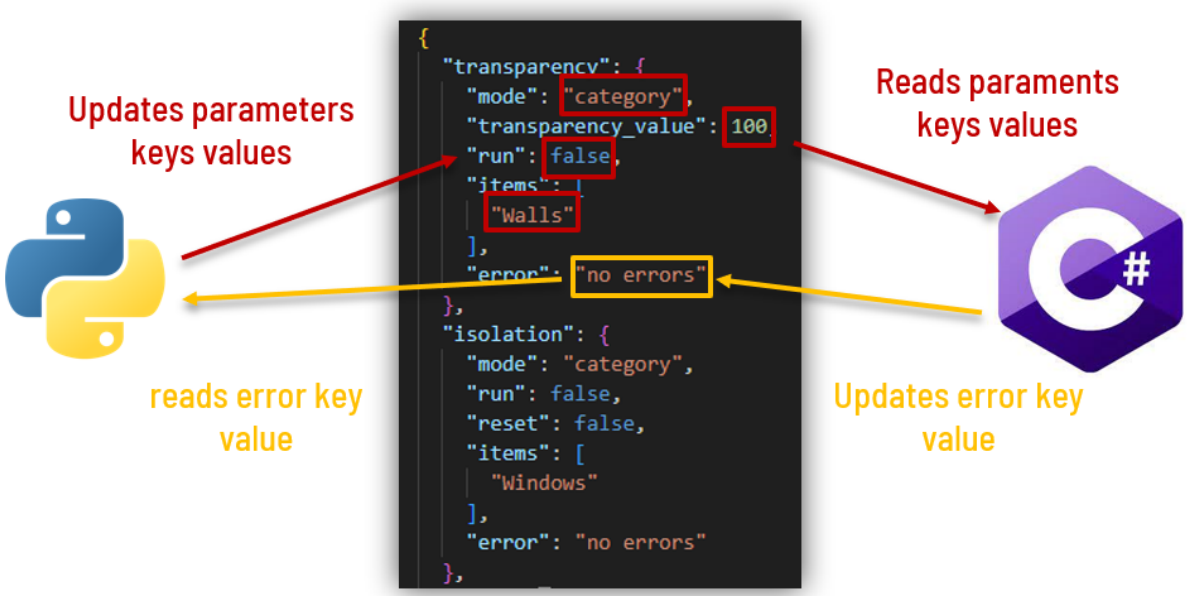


Figure 3.7: *bridge.json* schema and how it interacts with Python and C# components.

To further illustrate the principles of communication between the Python, JSON Bridge, and C# components, specifically through the actions *transparency* and *isolation*, detailed code snippets are provided: Appendix A for the Python component, Appendix B for the JSON bridge component, and, for an example of the C# component implementation, see Appendix C. The entire source code is available upon request.

3.1.4. File Watcher – C# Revit API Component

The C# component, as shown in **Figure 3.1c**, was developed in Visual Studio. It operates on the .NET Framework 4.8 and is constituted of two principal classes: *RevitFileWatcherApplication* and *FileChangeEventHandler*. This component employs libraries such as *Autodesk.Revit.DB* and *Autodesk.Revit.UI*, essential for interfacing with Revit's

architecture and user interface (explained more in Section 3.1.4.1). Additionally, *Newtonsoft.Json* is integrated for efficient JSON processing.

At the outset, the *RevitFileWatcherApplication* initializes the *FileSystemWatcher*, a class from the .NET Framework. The *FileSystemWatcher* class monitors the file system changes, triggering events when a directory or a file within it is modified. This functionality is vital for tracking real-time updates to the *bridge.json* file. Additionally, a debounce timer, set to one second, is implemented to filter out minor, frequent file changes. This timer is key in ensuring that the system processes only substantial changes, thus enhancing precision and efficiency while reducing computational load.

It is required for users to configure the environment variable *BRIDGE_JSON_PATH* correctly for the system to accurately locate and monitor the bridge file. Absence of this configuration prompts a message box within Revit, guiding the user to set the necessary path.

The Revit's API is a standout feature of this component, utilizing classes such as *Document*, *View*, *ElementId*, and *Transaction* for BIM interaction and manipulation. Central to this is the *FileChangeEventHandler* class, which decodes instructions from *bridge.json*. This file acts as a dynamic command center, mapping each BIM-related method to a main key in the JSON structure, with the method's arguments represented as sub-keys. These main keys directly correspond to specific actions within the Revit application.

The sub-keys serve as the arguments for the methods in the *FileChangeEventHandler*. For example, under the transparency key, one can specify the transparency mode, the *transparency_value*, and the specific items affected. Each action in bridge is executed by setting the run sub-key to *true*. Functions of greater complexity, like *selected_element_info* or *room_info*, involve additional JSON files for thorough data management. This enables intricate and detailed

manipulation of the BIM model (e.g., change the family type of an element or change the room name or occupant).

Robust error handling mechanisms are a critical component of the system's architecture, designed to effectively manage and mitigate potential operational issues. Each function within the C# component includes a *try-catch* block, ensuring handling of exceptions. Should an error occur during a command's execution, the exception is captured, and the details are logged into the corresponding error key within *bridge.json*. This allows DAVE to either autonomously resolve the issue or inform the user about the error, depending on its nature and severity. This system ensures that users are promptly informed of any issues, allowing for timely responses and interventions.

3.1.4.1. Autodesk API – Autodesk.Revit.UI & Autodesk.Revit.DB

In the context of Autodesk Revit API, "DB" stands for "Database." The Autodesk.Revit.DB namespace in the Revit API contains classes and methods that are used to access and manipulate the underlying database of a Revit project (Talarico, 2024). This database stores all the information about the project's elements, such as walls, windows, floors, parameters, and more.

The Autodesk.Revit.DB namespace provides the tools needed for reading from and writing to the Revit project file's data. Developers use this namespace to create custom add-ins and applications that can interact with Revit projects, enhancing functionality, automating repetitive tasks, or integrating with other software applications.

On the other hand, "UI" stands for "User Interface." The Autodesk.Revit.UI namespace provides classes and methods that allow developers to create and manage the user interface within Revit, such as dialogs, ribbon panels, buttons, and other UI elements (Talarico, 2024). This namespace is essential for developing Revit add-ins that require interaction with the user.

Using the Autodesk.Revit.UI namespace, developers can create custom ribbon tabs and panels to host their add-in command, develop windows and forms for inputting data or displaying information, interact with Revit's built-in UI components, such as the task dialogues, message boxes, and the status bar, and handle events related to the UI, like button clicks or changes in the UI state.

In essence, while Autodesk.Revit.DB deals with the data and elements within a Revit project (the "back-end"), Autodesk.Revit.UI focuses on how users interact with those elements and the overall Revit environment through the user interface (the "front-end"). Together, these two namespaces provide a comprehensive toolkit for developing powerful and user-friendly Revit add-ins and applications.

3.2. System Workflow

Building upon the foundational architecture outlined in Section 3.1, this section delves into the detailed workflow of the system, elucidating the step-by-step interactions between the user, the GPT-powered assistant, and Revit. A general flowchart (**Figure 3.8**) provides an overarching view of the process, while subsequent System Sequence Diagrams (SSD) (**Figure 3.9**, **Figure 3.10**, **Error! Reference source not found.** and **Error! Reference source not found.**) capture the specifics of four distinct system states.

3.2.1. System Interaction Overview

The system's basic interaction commences with user input and concludes with Revit executing the requested actions or providing a response. The general flowchart (**Figure 3.8**) offers a comprehensive view of this interaction, illustrating the pathways between system components

and the user.

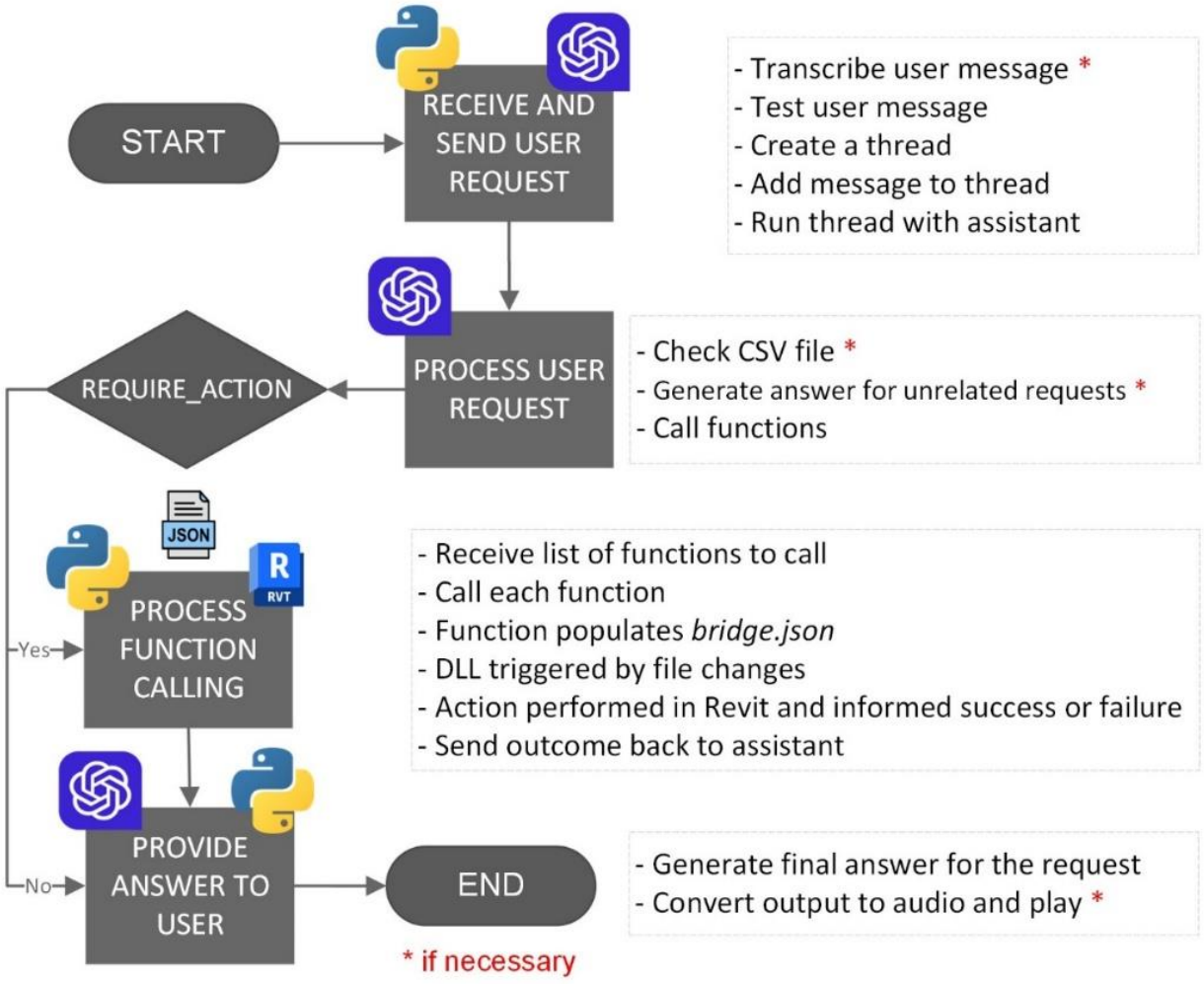


Figure 3.8: DAVE's workflow diagram.

3.2.2. User Input and Initial Processing

Inputs to the system can be in two forms: voice or text. In voice mode, the system listens for the activation phrase, "*hey Dave*", as explained in Section 3.1.2. Upon detection, auditory signals indicate the start and end of recording, and the audio is then transcribed into text. In text mode, users input their commands directly. Both input methods lead to the creation of a thread where the input is processed by the OpenAI API using the custom GPT-4 assistant. For example,

a user might say or type “[Hey Dave] change the transparency of the walls to 100%”.

3.2.3. Command Interpretation and Function Invocation

The Python backend creates a *thread* which then receives the user's message. The system proceeds to create a *run* request. In a *run*, all the GPT Assistant’s capabilities and instructions, including the functions it can invoke, are loaded to the thread. The complexity of these capabilities directly correlates with token consumption, which may impact the cost of the system if several runs are required to fulfill a request. The formulation of “good quality” queries by the user are discussed later in the Discussion (Chapter 5).

The Python script intermittently polls the status of the thread, with *in_progress* and *requires_action* being pivotal states. Informational queries are self-contained within the OpenAI environment, retaining an *in_progress* status until they reach a *completed* or *failed* state. Conversely, requests implicating Revit actions prompt a *requires_action* response, detailing the functions and arguments deduced as necessary by the Assistant. The SSD in **Figure 3.9** details the process for unrelated or database inquiries, and the one in **Figure 3.10** for Revit-related actions.

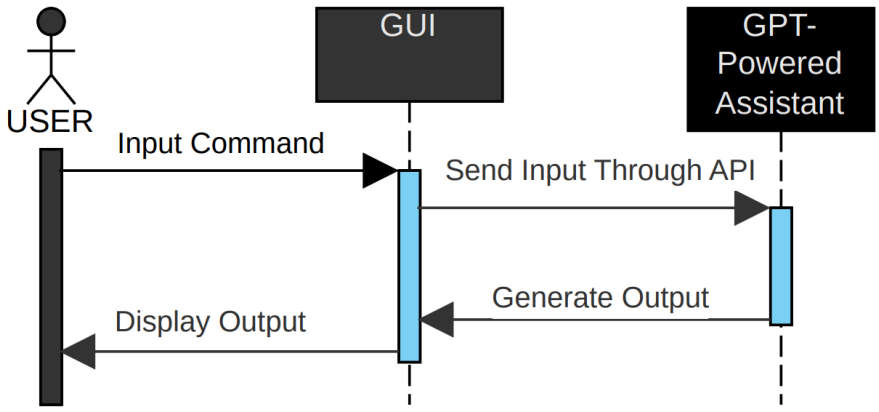


Figure 3.9: SSD of the unrelated or database inquiries process.

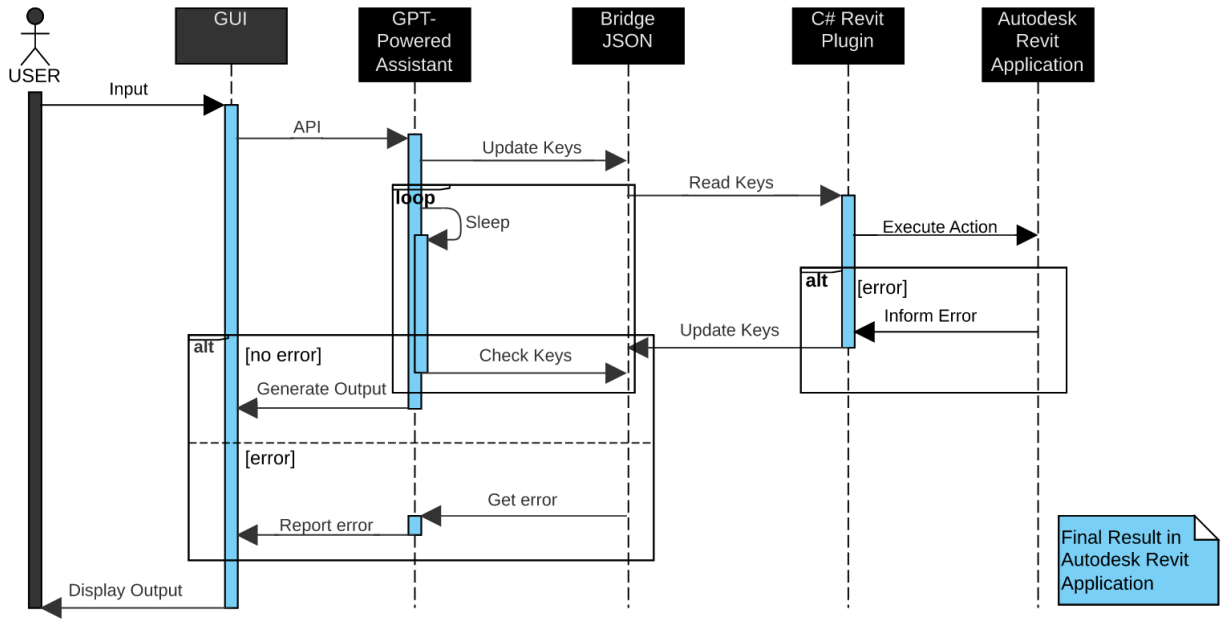


Figure 3.10: SSD of the general Revit-action process.

Continuing the previous scenario, after the user's message was loaded into the thread, and the GPT Assistant ran on it, the OpenAI Assistant class (Section 3.1.2.2) has a method that will catch the *requires_action* status and the following function to call (**Figure 3.11**):

```
{
  "id": "call_abc123",
  "type": "function",
  "function": {
    "name": "change_transparency",
    "arguments": "{\"transparency_value\": 100, \"mode\": \"category\", \"items\": [\"Walls\"]}"
  }
},
```

Figure 3.11: Example of the list of functions to be called produced by the GPT Assistant.

3.2.4. Synchronous Execution and Error Handling

Upon encountering a *requires_action* status, the Python script sequentially invokes the specified functions, updating the bridge JSON with the outcomes and monitoring for errors. This

serialized execution ensures methodical processing and error tracking. For the scenario discussed, the appropriate part of *bridge.json* would be updated as illustrated in **Figure 3.12**.


```
{
"transparency": {
"mode": "category",
"transparency_value": 100,
"run": true,
"items": ["Walls"],
"error": ""
},
```


Figure 3.12: Example of *bridge.json* schema after a called function run.

3.2.5. Interaction with Autodesk Revit and Specialized Function Handling

The File Watcher reacts to changes in the bridge JSON, executing corresponding actions within Revit and updating the keys *run* to *false* and *error* to *no error* (or the caught error during the execution). For the illustrative example, after the file watcher being triggered, it would then realize the true value on the run key of *transparency*. This would invoke the method Transparency and proceed to apply 100% transparency to the Walls category.

For the specialized functions *selected_element_info* and *room_info*, the system interacts with a secondary JSON file to manage data intricacies. The decision to utilize a secondary JSON file for *write* and *read* operations is rooted in the necessity to maintain a clean and efficient communication pathway within the system. By segregating detailed data handling from the primary operational flow, DAVE avoids overloading the *bridge.json* file and minimizes the risk of communication errors or delays. This structural choice not only enhances the system's reliability but also optimizes performance by ensuring that operations are executed with minimal latency and

maximum accuracy.

These auxiliary files serve as an intermediary for managing subsequent user-driven changes. For instance, if the user requires information about selected elements or rooms, the *write* operation will populate the secondary JSON. Upon receiving the command to perform the limited changes on the element (e.g., change family type) or room (e.g., change room name, number, or occupant), the *read* operation will retrieve the changes written on the secondary JSON. The corresponding SSDs (**Error! Reference source not found.** and **Error! Reference source not found.**) illustrate these operations, showcasing the system's approach to handling complex requests without overburdening the *bridge.json* file.

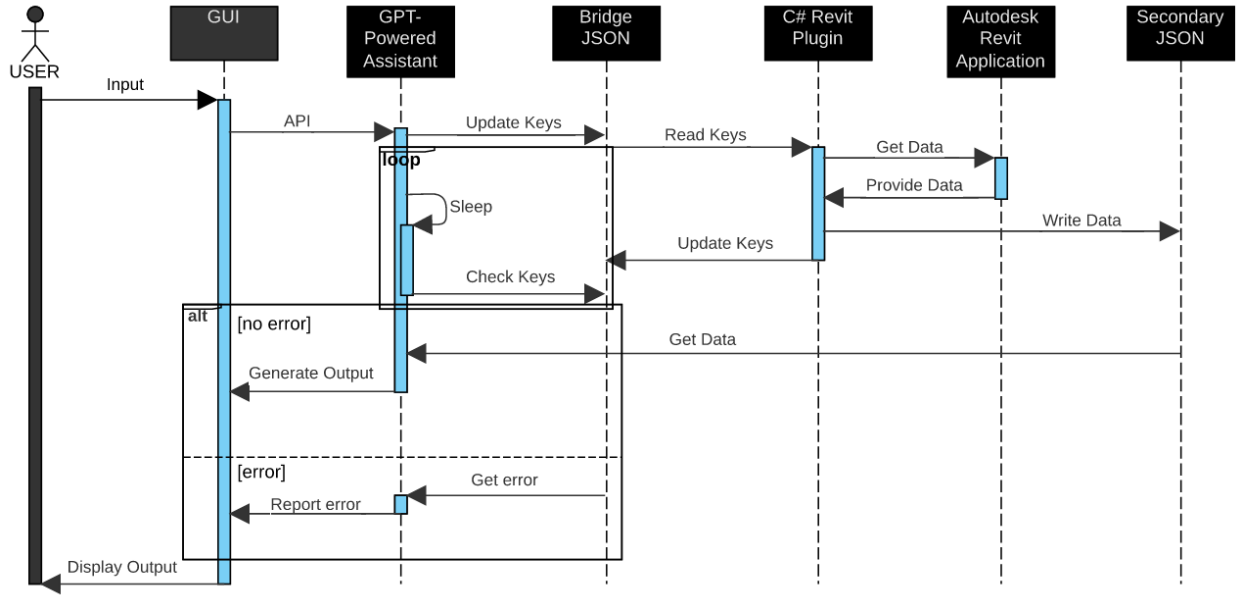


Figure 3.13: SSD of the *write* operation process.

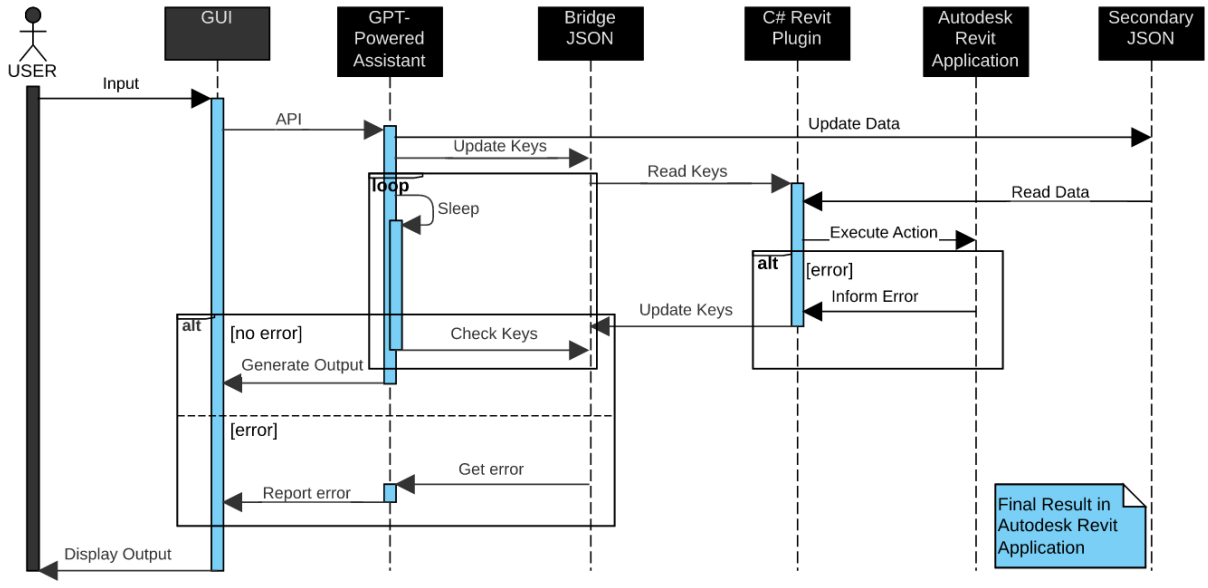


Figure 3.14: SSD of the *read* operation process.

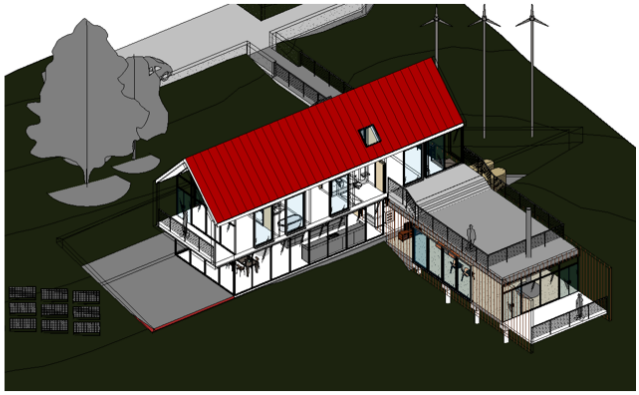
3.2.6. Completion and Response Formulation

When the error keys are no longer empty, and once all functions are executed and their results compiled, the Python script submits the outcomes back through a new API request, which then transitions the status of the run to *in_progress*. The GPT Assistant then can either formulate a response, invoke additional functions, or consult the database for argument values to refine the user's request.

The system's robust architecture ensures that, in the event of errors, the GPT Assistant will attempt autonomous resolution. Should these efforts be insufficient, the GPT Assistant communicates the error to the user, completing the feedback loop. **Figure 3.15** presents the system logs for the illustrative example including the final answer generated by the GPT Assistant (a) and the result in Revit (b).

```
[2024-02-03 15:29:17] USER: change the transparency of the
walls to 100%
[2024-02-03 15:29:19] SYSTEM: Current run status:
in_progress
[2024-02-03 15:29:24] SYSTEM: Current run status:
requires_action
[2024-02-03 15:29:24] SYSTEM: Calling change_transparency
with arguments: {
  "transparency_value": 100,
  "mode": "category",
  "items": ["Walls"]
}
[2024-02-03 15:29:24] SYSTEM: update_transparency called
with value: 100, mode: category, items: ['Walls']
[2024-02-03 15:30:11] SYSTEM: Current run status: completed
[2024-02-03 15:30:11] D.A.V.E.: The walls in our Demo1
project are now as transparent as a ghost in a snowstorm,
eh! If you need to spot them again, just give me a shout,
and I'll adjust their transparency back. Anything else you
need to do with those walls or another task to tackle?
```

(a)



(b)

Figure 3.15: (a) System logs for the exemplified user request, and (b) result in Revit application.

In the scenario where auditory feedback is desired, the system's capabilities extend to converting text responses into speech. Upon finalizing a response, should the user have specified or pre-set a preference for audio output, the system submits the text to the OpenAI API once more. This time the *tts-1* model is invoked to generate the spoken version of the GPT Assistant's reply. The resultant audio is then played back to the user, ensuring an interactive and accessible user experience.

4. Validation

4.1. Validation Methodology

To ensure the robustness and reliability of DAVE, a revised testing and validation methodology tailored to assess both the system's functional accuracy and its efficiency in handling complex queries is proposed. The validation approach was structured into distinct phases, each designed to evaluate different capabilities of the AI Assistant.

The initial phase involved the development of an automated testing script, specifically designed to validate the prototype’s capability to correctly invoke system functions in response to user queries. A set of 200 unique queries was prepared, each intended to trigger a single specific function within the system. This set aimed to cover a broad spectrum of the prototype’s potential use cases, ensuring a comprehensive assessment of its functional accuracy.

To extend the testing to more complex scenarios, it was generated an additional set of 200 queries by randomly combining pairs of the initial queries using a variety of linking phrases (e.g., "and", "as well as", ",", etc.). These compound queries were designed to simulate more complex user requests, requiring the prototype to call multiple functions sequentially within a single query. This step was critical for evaluating the system's ability to handle multi-functional requests, a capability touted by the OpenAI API (OpenAI, 2024d) for its efficiency and token economy.

Each query, simple or complex, was executed through the testing script, which recorded the time it took for the prototype to process the query (delay) and the specific functions (and their arguments) it invoked. This data collection was pivotal in assessing the system's performance, particularly in terms of response time and functional accuracy.

Following data collection, it was conducted a thorough analysis of the prototype’s performance, focusing on the accuracy of function invocation and the efficiency of response times.

The accuracy was assessed by determining whether the prototype correctly identified and called the appropriate function(s) for each query. Scores were assigned based on the number of correct functions invoked and their relevance to the queries posed.

In addition to accuracy and efficiency, errors are closely analyzed in function invocation to understand the underlying causes of inaccuracies. This error analysis provided critical insights into how queries could be better structured to increase the system's accuracy. Patterns in the queries that led to incorrect function calls were examined, using these findings to refine the approach to query formulation. This aspect of the analysis was instrumental in enhancing the system's capability to interpret and respond to user requests accurately, especially for complex queries requiring multiple functions.

Manual tests were also conducted to verify the final results on Revit environment to ensure real-time accuracy and execution. This manual testing phase was essential for confirming the system's practical applicability in architectural and engineering contexts, where the precision of function invocation is crucial.

4.2. Validation Results

4.2.1. Automated Functionality Testing Results

The automated functionality testing phase subjected prototype to 200 singular function queries (Type 1) and an additional 200 compound queries that combined pairs of initial queries (Type 2) using Demo2 CSV files. These were aimed at evaluating the system's capability to handle (1) single function calls and (2) multiple function calls within a single query as an effort to handle the maximum of user requests with a single run, aiming token consumption efficiency.

The confusion matrices for Query Types 1 and 2 (**Figure 4.1**) reveal that while the system

shows high accuracy in executing single-function queries, there is a decline in the accuracy when processing compound queries. This drop in performance is characterized by a higher frequency of failing to call a second required function in a single query.

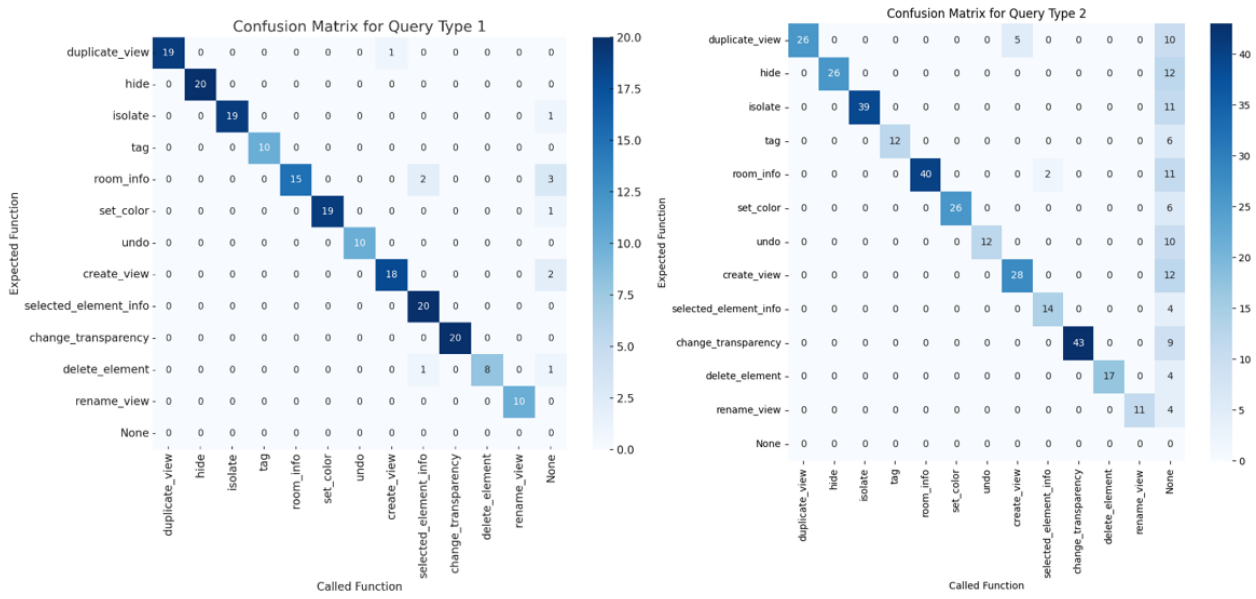


Figure 4.1: Confusion matrices for Query Type 1 (left) and Type 2 (right).

Figure 4.2 is the chart that shows the success and failure of functions called by Query Type and it elucidates a stark contrast in the success rates between the two query types. For Type 1, the success rate stood at 94% as the system leverages the robustness of the GPT-4 model in accurately processing and executing clear, single-action commands. However, when handling Type 2 queries, the success rate dipped to 49.5%. This significant variance accentuates the system's current limitations when processing queries that demand a sequence of actions, even though the AI model is capable of such feature (OpenAI, 2024d). The visualization of these results highlights areas where the system can be refined to better accommodate complex, multi-step operations without compromising the high success rate.

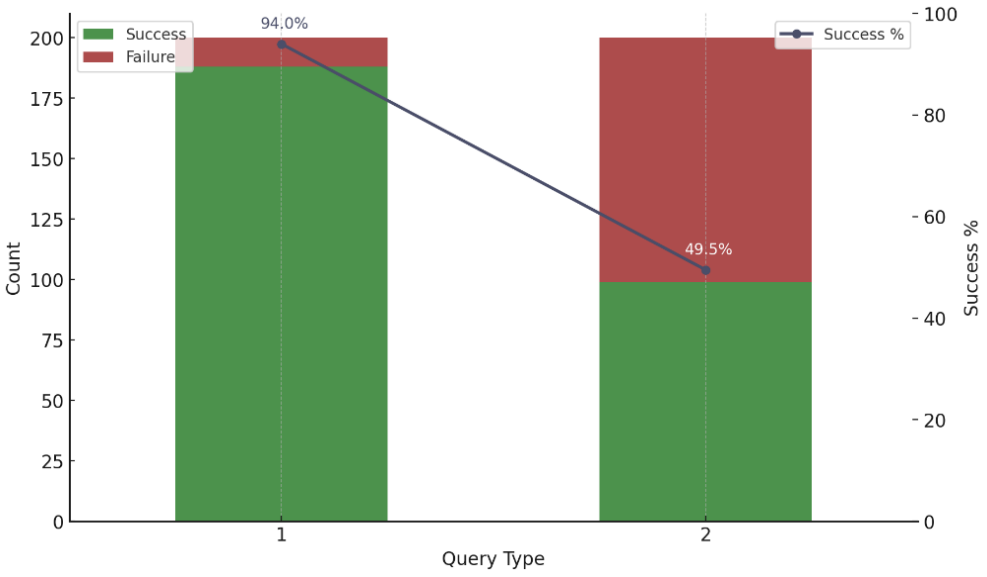


Figure 4.2: Evaluating query success and failure by function accuracy.

4.2.2. Error Analysis

In addition to the challenges in understanding and executing compound instructions, mislabeling of functions occurred frequently with similar instances, such as *selected_element_info* versus *room_info*, indicating that the user's function instructions could benefit from further clarification during the GPT Assistant creation.

When dealing with single-function queries that resulted in a failure status, it was noted that the GPT Assistant often attempted to request additional information from the user after not finding the suitable argument values directly from the dataset. This approach led to such interactions being marked as failures, as the testing protocol only looked at the initial response by the GPT Assistant.

Furthermore, it was identified minor inaccuracies within the argument values themselves. These included errors such as the improper capitalization of view names, discrepancies in the expected number of categories for open-ended requests (e.g., requests for "all devices in the project" or "all HVAC related categories"), and the inclusion of unnecessary (or optional) parameters.

Interestingly, when it was necessary, the GPT Assistant demonstrated a proactive approach by searching within CSV files for argument values, such as floor names and categories. This capability was particularly evident in Query Type 1 interactions, where the GPT Assistant utilized the extracted information with notable accuracy. This behavior underscores the prototype's potential to effectively navigate and utilize dataset content when provided with clear and precise instructions.

During manual testing within Revit, common issues arose when actions were not supported by the current view, such as attempting to hide elements in a schedule view. In such instances, errors were communicated back to the GPT Assistant, which then informed the user of the specific issue. These problems were resolved by guiding the user to switch to an appropriate view or state within Revit where the function could be successfully executed. This process ensures that the user is aware of and can rectify any limitations in command execution, thereby maintaining the system's usability and effectiveness.

DAVE's interaction with Revit presents operational limitations when actions prompt dialog boxes requiring manual intervention. For instance, deleting structural elements that affect room definitions triggers Revit warnings, halting automated processes until manually addressed by the user. This limitation underscores the necessity for a more intelligent system capable of predicting such scenarios or providing users with preemptive guidance to avoid workflow disruptions.

Finally, the exploration extended to operational issues related to system architecture and external dependencies, such as API accessibility, rate limit exceedances, and data synchronization errors. These additional challenges emphasize the complexity of creating a seamless and robust AI-powered BIM tool. **Table 4.1** summarizes the diverse range of errors encountered during the validation process, outlining their causes, and proposing solutions to mitigate these issues.

Table 4.1: Error analysis.

Error Type	Frequency	Cause	Proposed Solution
Function Mislabeling	Occurred in complex queries	Ambiguity in user queries leading to incorrect function mapping	Enhance NLP capabilities to improve intent recognition and query classification. Implement detailed prompt engineering guidelines for users
Delay in Response	Noted in operations requiring external data consultation	Lengthy data retrieval processes from CSV files and processing time by GPT-4	Optimize data retrieval algorithms and consider local caching of frequently accessed data to reduce response times
Incorrect Argument Values	Several instances during testing	Errors in parsing or interpreting user inputs, leading to mismatched or incorrect command parameters	Refine data parsing algorithms and implement additional validation checks to ensure accuracy of interpreted arguments. Improve the instruction clarity of functions during the GPT Assistant creation
Revit View Compatibility Issues	Frequent in specific operations and conditions	Attempting to execute an action in Revit that is not supported by the current view or context	Enhance DAVE's ability to recognize the context and limitations of the current Revit view. Implement a feature that informs the user of the incompatibility and suggests switching to an appropriate view where the action is feasible
Linked Model Functionality Limitation	Encountered during operations involving linked models (Demo2)	DAVE's current architecture primarily operates on the main Revit model and may not directly interact with elements in the linked Revit models	Develop and integrate a module or extend the system's capabilities to recognize and manipulate elements within linked Revit models, ensuring comprehensive model management
Revit Dialog Box	Occurs during operations leading	Certain actions (e.g., deleting a wall that defines a room) trigger Revit dialog	Develop a mechanism for DAVE to recognize potential actions that could trigger dialog boxes and either provide

Intervention	to Revit warnings or errors requiring manual intervention	boxes that DAVE cannot automatically close or resolve, necessitating manual user intervention	a preemptive warning to the user or incorporate strategies for automated handling of common dialog scenarios. Further, explore the integration of ML techniques to predict and mitigate actions leading to disruptive dialogs
API Accessibility Issues	Occasional, depends on OpenAI server availability	Temporary unavailability or slowdowns of OpenAI's servers can impact DAVE's response times and functionality	Implement retry mechanisms and provide user feedback on server status. Consider local fallback processes for limited offline functionality.
API Rate Limit Exceedance	Temporary, dependent on user account limits	Exceeding OpenAI API call limits under the initial account setup that leads to reduced functionality	Expand account limits to accommodate higher usage. Monitor API usage closely and adjust plan or optimize queries to manage costs and maintain continuous service
Data Synchronization Errors	Could occur after external model updates	Changes made directly in Revit that are not captured in real-time by DAVE can lead to outdated information in the system's database (CSV file)	Revise the data retrieval process to utilize a more dynamic and efficient method or implement a live data synchronization mechanism

4.2.3. Response Time Analysis

The system's response times were generally swift, with an average of 16.25 seconds for Type 1 and 14.63 seconds for Type 2 queries. However, the standard deviation was notably high for both types (25.46 and 29.74 seconds for Type 1 and 2 respectively), attributed to instances where the system consulted an external CSV file which caused delays. Consultations typically lasted 1 to 2 minutes, but some cases exceeded this, with one instance taking nearly 5 minutes. This variability in response times underscores the need for optimized data retrieval processes aimed for the future versions of the prototype.

Analysis of response times across different functions unveiled considerable variability in delays. The chart that shows the average delay by function and call order (**Figure 4.3**) illustrates that those functions necessitating database consultations, such as *create_view* and *hide*, incurred longer average delays. This pattern underscores a direct correlation between the complexity of data retrieval—for instance, verifying floor names for *create_view* or handling a vast array of queries for *hide* when faced with open-ended requests—and the subsequent processing time required.

Additionally, a notable increase in average delay was observed for cases where no specific function was called (described as "*None"* cases in **Figure 4.3**), as well as for operations like *delete_element*. This phenomenon could be attributed to the GPT Assistant's misinterpretation of user requests, leading to extensive efforts to autonomously resolve ambiguities, often culminating in unsuccessful outcomes. These findings highlight the impact of function-specific demands and the GPT Assistant's interpretative challenges on response times, suggesting areas for further optimization to enhance efficiency and user experience.

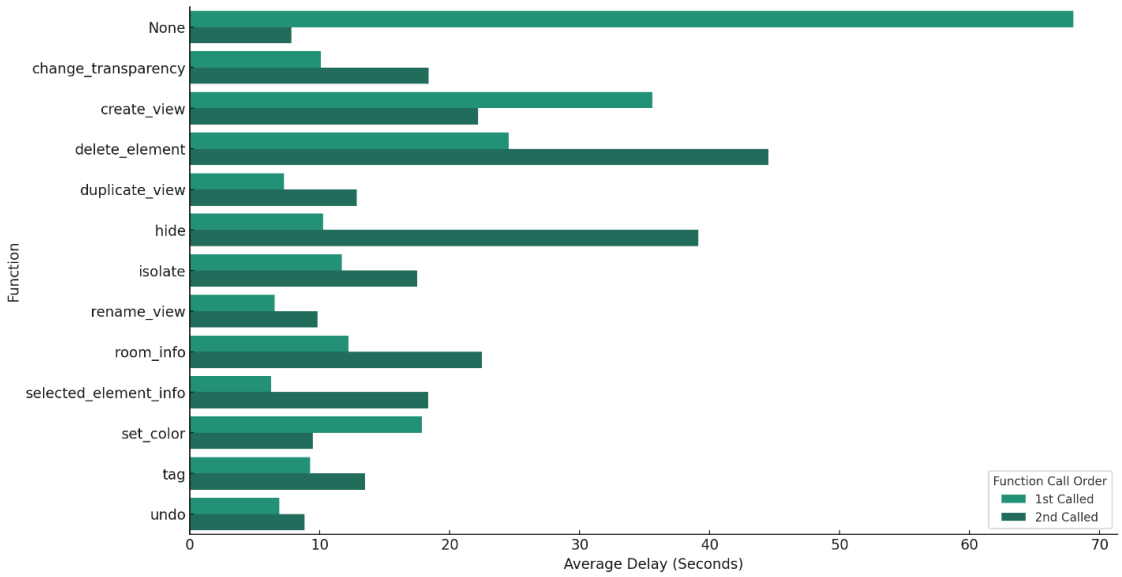


Figure 4.3: Average delay by function and call order.

Manual testing within the Revit environment confirmed that, when the system correctly interpreted functions and their arguments, the implementation of changes was virtually instantaneous. This rapid response is attributed to the prototype's efficient handling of JSON files and the efficacy of the Revit API.

In addition to the factors previously discussed, the utilization of voice mode introduced an additional variable to the system's response times. During testing, it was observed that voice mode—either through receiving commands by voice or providing replies using voice—added an average delay of 3-5 seconds to the overall response time. It is important to note that this delay can vary significantly depending on external factors, including the user's computer capacity, Internet connection speed, and the current load on OpenAI's servers. The system was tested on high-end computers (equipped with at least an Intel i7 processor, a minimum of 32 GB RAM, SSD storage, and powerful GPUs) to minimize potential bottlenecks related to hardware performance. Despite these high specifications, the variability in response times highlights the complexity of delivering a real-time AI-powered BIM tool and underscores the need for continued optimization.

5. Discussion

5.1. Contributions

The development and implementation of DAVE represents a significant advancement in the integration of Conversational AI within the AEC industry, particularly for BIM model manipulation. The main contribution of this study is a system that leverages the power of AI to facilitate real-time updates and interactions with BIM models while offering a novel approach to digital model management that is both intuitive and efficient. The use of voice commands and NLP allows users to interact with BIM models in a way that reduces cognitive load and streamlines the modification process, making DAVE a useful tool for architects, engineers, and other construction professionals. This section delves into the strengths and implications of the system, highlighting its contributions to advancing conversational AI in AEC, enhancing BIM workflows, and promoting accessibility and inclusivity.

5.1.1. Advancement of Conversational AI in AEC

DAVE represents a novel advancement in the application of conversational AI in the AEC sector. Unlike existing research, which primarily focuses on leveraging AI for information retrieval from BIM models, DAVE extends this capability to include dynamic updates, queries, and real-time model interactions through natural language processing and voice commands. This broader application scope marks a significant departure from the status quo, positioning DAVE as a pioneering solution that facilitates a more interactive and engaging user experience. For example, DAVE enables on-the-fly adjustments to model elements using simple voice commands, revolutionizing traditional workflow approaches in architectural and engineering tasks. This system demonstrates the potential for conversational AI to transition from a supplementary tool to

a central component in project management and execution, challenging traditional practices and encouraging a reevaluation of how tasks are approached and completed in the AEC industry.

5.1.2. Customization and Automation in BIM-Authoring Tool

The integration of AI with BIM through DAVE introduces a high level of customization and automation in BIM-authoring tools. This leap in technology not only fills a crucial gap in BIM tool functionality but also sets a new benchmark for software adaptability and user-directed automation in the AEC industry. Leveraging NLP, DAVE provides a dynamic and user-friendly interface that adapts to the specific needs and preferences of its users. This adaptability not only enhances the user experience but also streamlines BIM tasks through automation, reducing the need for manual inputs and enabling more efficient project workflows. The real-time interaction capability ensures that updates and queries can be processed in a timely manner, further enhancing the system's utility in fast-paced project environments. Central to DAVE's strengths is its accuracy, particularly highlighted in executing single commands. Validation results demonstrate a remarkable success rate for simple queries, underscoring the system's ability to understand and process user inputs with precision. This accuracy not only improves the user experience by providing reliable and timely responses but also contributes to the overall effectiveness of BIM project management. This shift introduces a new paradigm in BIM model interaction, where users are no longer constrained by the complexities of software interfaces or the intricacies of manual model updates.

5.1.3. Enhancement of BIM Workflow

DAVE enhances the BIM workflow by complementing and extending the current state of

research. This innovation directly tackles previously identified inefficiencies, such as complex interface navigation and ineffective communication between disciplines, and accessibility issues within BIM. As such, DAVE constitutes a significant addition, propelling the industry toward more streamlined and intuitive practices. For instance, the reduction of errors and revisions is a direct outcome of the system's ability to accurately interpret and execute user commands. By minimizing manual data entry and automating routine tasks, DAVE helps to maintain the integrity of BIM models and reduces the likelihood of mistakes. The system also facilitates remote collaboration by allowing team members to communicate updates and changes to the BIM model in real-time, irrespective of their physical location. This capability is particularly valuable in the context of increasing remote work trends, enabling more efficient and cohesive project management. The system's support for voice commands further enables hands-free operation, allowing users to remain engaged with their tasks while interacting with the BIM model, such as during design review meetings or client presentations, thus significantly reducing the time and effort required for model manipulation.

Furthermore, the integration of Virtual Reality (VR) with DAVE brings an additional layer of innovation to the BIM workflow. By enabling users to immerse themselves within a virtual representation of the BIM model, VR technology allows for an unparalleled level of interaction and spatial understanding. Real-time changes made to the BIM model can be instantly visualized in the VR environment, offering immediate feedback and facilitating a more intuitive design and review process. This immersive experience not only enhances the ability to identify potential design issues before they become costly changes but also improves stakeholder engagement by allowing non-technical participants to visualize and understand complex spatial relationships. The combination of DAVE with VR technology represents a leap forward in making BIM workflows

more dynamic, interactive, and accessible, further bridging the gap between digital modeling and physical realization.

5.1.4. Accessibility and Inclusivity

DAVE's application of voice commands and natural language interaction to perform actions within BIM authoring tools represents a groundbreaking contribution to the field. While voice command technology itself is not new, DAVE pioneers its use for direct command execution in BIM environments, significantly enhancing accessibility and inclusivity. This innovation democratizes BIM technology, providing a unique, user-friendly interface for a broader range of AEC professionals and stakeholders and effectively filling a crucial gap in current BIM practices. By allowing users to interact with the software in a more intuitive and natural manner, the system lowers the barrier to entry for individuals who may not have extensive training in BIM. Furthermore, by leveraging the LLM's vast knowledge base, DAVE can also serve as an on-demand tutor, guiding users through the functionalities of Revit, and eventually, other BIM-authoring software. This inclusivity extends the benefits of BIM technologies to a broader audience, promoting a more diverse and collaborative industry landscape. The provision of instant, AI-driven support for software-related inquiries can significantly flatten the learning curve, making BIM technologies more accessible and less intimidating to new users.

5.2. Challenges and Limitations

Despite DAVE’s innovative approach and its benefits, it confronts several challenges that include ensuring scalability amidst complex model management, maintaining data integrity and security, and guaranteeing privacy. Furthermore, achieving seamless interoperability with diverse

BIM tools and optimizing system efficiency—particularly the speed and efficacy of information retrieval processes—are critical areas that require ongoing attention.

Additionally, DAVE's current implementation does not fully exploit the LLM's capacity for proactive design optimization and autonomous decision-making. This represents a missed opportunity to leverage AI not merely as a tool for executing user commands but as an active participant in the design process, capable of suggesting optimizations and innovative solutions.

Finally, the system's reliance on high-quality user queries and the economic implications of employing advanced AI models like GPT-4 necessitates a delicate balance to ensure the solution's long-term viability. These challenges are further explored in the next sections.

5.2.1. Scalability, Data Integrity, Privacy and Security

The scalability of the current version of the system is challenged by its architecture, which requires mapping each Revit action to a unique Python function. As BIM models become more complex, the sheer volume of potential user actions necessitates a substantial increase in the Python function library. This expansion demands an ongoing development to keep pace with Revit updates and to enhance functionality. Exploring more dynamic architectures, such as incorporating ML to automate the generation and refinement of Python functions based on user interactions, could significantly alleviate manual expansion efforts, and improve the system's flexibility in accommodating new Revit functionalities and user needs.

Given the reliance on OpenAI's API for DAVE's operation, the system inherits OpenAI's policies on privacy and security. OpenAI commits to high standards of data privacy and security, including encryption of data in transit and at rest, and adherence to leading industry practices. However, concerns may arise within the AEC industry regarding the sharing of sensitive project

data with external APIs. Additionally, converting BIM models to CSV format may also raise concerns regarding the secure handling of potentially sensitive information. To mitigate these risks, it is proposed enhancements to the system's data handling protocols by implementing additional layers of encryption for data at rest and in transit, particularly for CSV files created during the data conversion process. Furthermore, adopting a more secure method of interacting with the APS API, such as utilizing OAuth tokens (Autodesk, 2023) with limited permissions, can reduce the exposure of sensitive data. Regular security audits and adherence to Autodesk's recommended practices for API use will further ensure that the developed system remains robust against potential vulnerabilities.

5.2.2. Interoperability Challenges

DAVE's current compatibility is exclusively with Revit, although Revit is the most commonly used BIM authoring tool (especially in North America) (6sense, 2024), this can be considered as a limitation in the context of the AEC industry's diverse ecosystem of BIM tools. A specific aspect of this challenge within the Revit environment is Revit's linking functionality, which is crucial for managing large projects with multiple files or collaborating across disciplines but currently poses limitations for DAVE. The system struggles to accurately apply certain functions to linked Revit models, often defaulting to actions on the main file rather than appropriately interacting with the linked files. Addressing these limitations is essential for DAVE to realize its full potential as a transformative tool in the AEC industry, fostering a more inclusive and adaptable approach to BIM model manipulation and management. Future development efforts will focus on enhancing DAVE's ability to recognize and interact with Revit links and expanding its compatibility with other BIM software, ensuring that DAVE can serve as a versatile and

comprehensive tool for BIM model management across the industry's diverse technological landscape.

5.2.3. Query Quality

Query quality emerges as a pivotal factor that significantly impacts DAVE's performance. The effectiveness of the system in interpreting and executing user commands hinges on the clarity, specificity, and comprehensiveness of the queries posed. This reliance underscores the importance of 'prompt engineering'—a nuanced process of strategically crafting queries to effectively communicate with AI systems. Prompt engineering involves the deliberate structuring of requests to maximize the AI's understanding and response accuracy. It requires users to be adept at formulating queries that are not only clear but also structured in a manner that aligns with the AI's processing capabilities (OpenAI, 2024c).

A 'good quality' query, therefore, is one that is explicit in its intent, detailed in its requirements, and structured in a way that directly aligns with the AI model's understanding. For example, instead of a vague command like "Change the walls," a good quality query would specify the desired action in detail: "Increase the transparency of the north wall's windows to 75%." This specificity helps the AI to accurately parse the user's intent and execute the command without the need for further clarification or guesswork.

The impact of query quality on DAVE's performance became particularly evident during the system validation phase. Tests revealed that queries precisely tailored to DAVE's linguistic model and functional schema resulted in higher accuracy and efficiency in command execution. Conversely, queries that were vague, overly complex without clear direction, or misaligned with the system's expected input format often led to misinterpretations, incomplete actions, or the need

for multiple iterations to achieve the intended outcome. This underscores the critical role of prompt engineering in enhancing user interaction with DAVE, necessitating additional guidance for users to master the art of formulating effective queries.

To mitigate these challenges and improve system usability, future iterations of DAVE could incorporate more advanced natural language understanding capabilities or offer interactive tutorials and feedback mechanisms to assist users in crafting high-quality queries. Educating users on the principles of prompt engineering and providing examples of effective queries could significantly enhance the overall user experience and system performance.

5.2.4. System Efficiency and Economic Considerations

The efficiency of DAVE, particularly in complex or large-scale BIM projects, is occasionally hampered by delays in information retrieval, which can extend up to five minutes. Such delays, primarily arising from the processing time required by GPT-4 to analyze and respond to queries, may disrupt project workflows. For a seamless integration of DAVE into daily operations, it is planned to address these delays through optimization of the system's architecture, such as enhancing data processing and retrieval methods or integrating more responsive AI models.

Efforts to streamline information retrieval are crucial, as the protracted retrieval times currently experienced can significantly hinder workflow momentum. Addressing this challenge is a topic of ongoing research, as in (Zheng & Fischer, 2023), where an optimized dynamic framework is proposed for efficient data structuring and querying. Adopting such methodologies could markedly enhance DAVE's ability to rapidly access information.

Moreover, the synchronous nature of DAVE's operations, wherein tasks are processed

sequentially, may lead to bottlenecks, particularly when handling multiple or complex queries simultaneously. Future iterations of DAVE will explore the integration of asynchronous processing techniques to enhance throughput and system responsiveness.

The operational costs associated with employing GPT-4, OpenAI's advanced language model, may present another challenge. The current general instructions and functions details represent a usage of around 3,700 tokens per request (**Figure 5.1**), while the answer produces can generate as low as 97 tokens. For the current system's capabilities and current price, it can represent a cost of approximately U$ 0.05 per request.

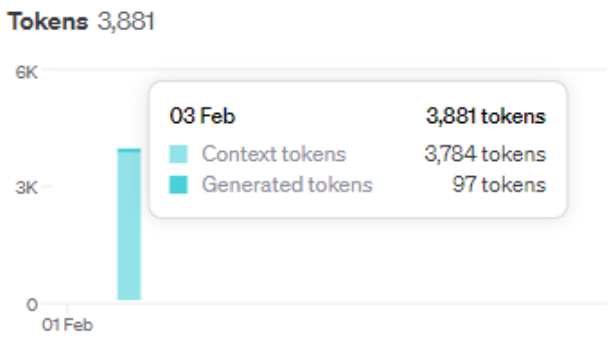


Figure 5.1: Approximate tokens consumption for a single request.

While GPT-4's capabilities enable DAVE to understand and execute complex commands with remarkable accuracy, the cost implications of leveraging such a sophisticated AI tool can be substantial. This is particularly relevant when considering the volume of queries processed daily in a typical AEC project. To ensure DAVE remains an economically viable solution, it is essential to strategize the management of query formulation and system usage. This could involve optimizing queries to reduce unnecessary token consumption and exploring cost-effective alternatives or enhancements to GPT-4.

6. Summary, Conclusions and Recommendations

6.1. Summary

In this study, it has been explored the transformative potential of BIM with AI, focusing on the development of DAVE, a digital assistant designed to enhance model interaction and management within the AEC industry. The investigation highlights how advancements in AI, particularly in NLP and LLMs like GPT, offer innovative solutions to overcome the inherent limitations of traditional BIM methodologies. By enabling intuitive, conversational interactions with BIM systems, DAVE represents a significant contribution to the current body of knowledge and a leap towards making these technologies more accessible and efficient for professionals across the AEC sector. Through a comprehensive literature review in this thesis, the key challenges in BIM and AI integration are identified, including complexity, interoperability, and the need for enhanced user interfaces. It is also discussed how AI-driven tools like DAVE can address these issues, thereby streamlining project management, decision-making, and operational efficiency in construction projects.

6.2. Conclusions

By harnessing the capabilities of GPT-powered conversational AI, DAVE transcends traditional barriers, offering an intuitive and efficient means for users to engage with complex BIM environments. This breakthrough not only signifies a leap towards simplifying the BIM interface for a wider audience but also exemplifies the practical application of cutting-edge AI technologies in enhancing the operational efficiency and decision-making processes in construction projects. The exploration into DAVE's development, functionality, and potential challenges lays the groundwork for future innovations in the AEC sector, encouraging a paradigm shift towards more

accessible, intelligent, and user-centric model management solutions.

The study sought to answer the following specific research questions:

- **Integration**: *How can an AI chatbot be integrated with a BIM authoring tool to enable real-time data exchange and updates effectively, thereby streamlining project workflows?*
    - DAVE demonstrates that an AI chatbot can be seamlessly integrated with a BIM authoring tool, such as Autodesk Revit, to allow for real-time data exchange and updates. This integration has been shown to streamline project workflows significantly by enabling dynamic updates, queries, and interactions through voice commands and natural language, effectively reducing the complexity and technical barriers traditionally associated with BIM.
- **Evaluation:** *In what ways is the used LLM effective in interpreting and executing BIM-related queries?*
    - The LLM used in this study, namely GPT-4, has proven effective in interpreting and executing BIM-related queries with a high degree of accuracy. The system's success in understanding complex commands and performing detailed actions within the Revit environment confirms the potential of LLMs to enhance BIM functionalities. The LLM's ability to process natural language inputs and translate them into precise BIM operations has been a cornerstone of DAVE's effectiveness.
- **Performance Assessment:** *How does the prototype perform when handling simple versus compound queries within the BIM context?*

- Validation results have shown that DAVE performs exceptionally well when handling simple queries, with a high success rate in executing single-function commands accurately. However, the prototype faces challenges with compound queries, highlighting an area for future optimization. The system's architecture and processing capabilities will be further refined to improve its handling of complex, multi-step operations.

❖ **Exploration:** *What potential does the prototype hold in improving project development, management, execution, collaboration, and decision-making in BIM projects?*

- DAVE holds significant potential in improving various aspects of project development and management in BIM projects. Its intuitive interface and real-time processing capabilities facilitate better decision-making, enhance collaboration among team members, and streamline execution processes. By democratizing access to BIM technologies, DAVE contributes to a more inclusive and efficient project environment.

Within the realm of Civil Engineering, DAVE's contributions are particularly noteworthy. The prototype has demonstrated the ability to significantly enhance the accuracy, efficiency, and adaptability of Civil Engineering practices. For example, DAVE's has the potential to automate material quantity takeoffs and updating cost estimates in real-time address specific challenges in Civil Engineering project management. By providing a more accessible interface for interacting with complex BIM models, DAVE reduces the cognitive load of civil engineers, enabling them to focus on critical design and analysis tasks.

Despite its innovative approach, DAVE faces some challenges such as scalability, interoperability, and the need for precise user queries. Future iterations of the developed system

aim at addressing these challenges, alongside ensuring security, privacy, and overcoming technical limitations such as handling Revit links. Future work includes extending DAVE's compatibility with other BIM software, refining query processing capabilities, and mitigating economic considerations associated with employing sophisticated AI models like GPT-4.

Moreover, engaging with the AEC community for comprehensive user testing and feedback is crucial. This collaborative effort will help tailor DAVE to meet the practical needs of professionals, ensuring its relevance and applicability in real-world settings. As we look towards a more connected, intelligent, and user-centric future in the built environment, systems like DAVE underscore the potential of conversational AI to revolutionize BIM functionalities and enhance construction efficiency. In conclusion, DAVE embodies a significant step forward in the digital transformation of the AEC industry. Its development highlights the vast potential for AI to redefine architectural and construction methodologies, setting the stage for a future where technology and human expertise converge to create more efficient, sustainable, and intelligent built environments.

6.3. Recommendations for Future Work

Addressing the limitations identified in this study is essential for the advancement and wider adoption of AI-powered BIM assistants like DAVE. Future work will focus on expanding compatibility with other BIM software, improving the system's ability to handle complex and multi-step queries, and refining the data conversion process to mitigate information loss. Additionally, developing a more sophisticated understanding of query quality and user interaction patterns will enhance the system's usability and efficiency.

A significant avenue for future research involves conducting comprehensive user testing to gather feedback on DAVE's functionality, usability, and integration within professional

workflows. This feedback will be instrumental in identifying areas for improvement and ensuring that the system meets the practical needs of AEC professionals.

A promising direction for future research involves extending DAVE's capabilities to include proactive design suggestions and optimizations. Leveraging the full potential of the LLM could transform DAVE from a reactive assistant into an intelligent design partner, capable of offering recommendations based on best practices, efficiency considerations, and sustainability criteria. This would not only streamline the design process but also foster a collaborative environment where AI contributes creatively to project development.

Moreover, exploring the potential for integrating DAVE with emerging technologies such as VR, Augmented Reality (AR), and Mixed Reality (MX) presents an exciting frontier. The combination of DAVE with these immersive technologies could revolutionize the BIM workflow by enabling more intuitive and interactive design and review processes.

Additionally, exploring the potential for developing a built-in assistant directly within the Revit environment, thereby eliminating the need for external data conversion, API calls and manual database updates, represents a promising direction. This integration could further streamline interactions and increase the system's responsiveness and accuracy.

7. References

6sense. (2024). *Autodesk Revit*. https://6sense.com/tech/bim-and-architectural-design-software/autodesk-revit-market-share

Abioye, S. O., Oyedele, L. O., Akanbi, L., Ajayi, A., Davila Delgado, J. M., Bilal, M., Akinade, O. O., & Ahmed, A. (2021). Artificial intelligence in the construction industry: A review of present status, opportunities and future challenges. In *Journal of Building Engineering* (Vol. 44). Elsevier Ltd. https://doi.org/10.1016/j.jobe.2021.103299

Adel, K., Elhakeem, A., & Marzouk, M. (2022). Chatbot for construction firms using scalable blockchain network. *Automation in Construction*, *141*. https://doi.org/10.1016/j.autcon.2022.104390

Ahmadpanah, H., Haidar, A., & Latifi, S. M. (2023). BIM and Machine Learning (ML) Integration in Design Coordination Using ML to automate object classification for clash detection. *ECAADe 41*, 619–628.

Autodesk. (2023). *Authentication (OAuth) Scopes*. https://aps.autodesk.com/en/docs/oauth/v2/developers_guide/scopes/

Autodesk. (2024). *What is BIM*. https://www.autodesk.com/solutions/aec/bim#:~:text=Building%20information%20modeling%20(BIM)%20is%20the%20holistic%20process%20of%20creating,information%20for%20a%20built%20asset.

Azhar, S. (2011). Building Information Modeling (BIM): Trends, Benefits, Risks, and Challenges for the AEC Industry. *Leadership and Management in Engineering*, *11*(3), 241–252. https://doi.org/10.1061/(ASCE)LM.1943-5630.0000127

Becerik-Gerber, B., & Rice, S. (2010). Becerik-Gerber and Rice, pg. 185 www.itcon.org-Journal

of Information Technology in Construction-ISSN 1874-4753 U.S. building industry. In *Journal of Information Technology in Construction (ITcon)* (Vol. 15). http://www.itcon.org/2010/15

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P. S., Yang, Q., & Xie, X. (2024). A Survey on Evaluation of Large Language Models. *ACM Transactions on Intelligent Systems and Technology*. https://doi.org/10.1145/3641289

Chung, S., Moon, S., Kim, J., Kim, J., Lim, S., & Chi, S. (2023). Comparing natural language processing (NLP) applications in construction and computer science using preferred reporting items for systematic reviews (PRISMA). In *Automation in Construction* (Vol. 154). Elsevier B.V. https://doi.org/10.1016/j.autcon.2023.105020

Darko, A., Chan, A. P. C., Adabre, M. A., Edwards, D. J., Hosseini, M. R., & Ameyaw, E. E. (2020). Artificial intelligence in the AEC industry: Scientometric analysis and visualization of research activities. In *Automation in Construction* (Vol. 112). Elsevier B.V. https://doi.org/10.1016/j.autcon.2020.103081

Ding, Y., Ma, J., & Luo, X. (2022). Applications of natural language processing in construction. In *Automation in Construction* (Vol. 136). Elsevier B.V. https://doi.org/10.1016/j.autcon.2022.104169

Enshassi, M. A., Hallaq, K. Al, & Tayeh, B. A. (2019). Limitation Factors of Building Information Modeling (BIM) Implementation. *The Open Construction and Building Technology Journal*, *13*(1), 189–196. https://doi.org/10.2174/1874836801913010189

Erfani, A., & Cui, Q. (2021). Natural Language Processing Application in Construction Domain: An Integrative Review and Algorithms Comparison. *Computing in Civil Engineering 2021* -

Selected Papers from the ASCE International Conference on Computing in Civil Engineering 2021, 26–33. https://doi.org/10.1061/9780784483893.004

Ghimire, P., Kim, K., & Acharya, M. (2024). Opportunities and Challenges of Generative AI in Construction Industry: Focusing on Adoption of Text-Based Models. *Buildings*, *14*(1), 220. https://doi.org/10.3390/buildings14010220

Ghosh, D. P., Soni, D., Gopikanna, S., Nambiar, N. A., Apoorva, Y. S., Kumarasamy, S., Moitheen, A., Gupta, I., Tiwari, A., Gupta, S., Singh, D., Durga, P., & Akan, M. (2023). Interactive Design Automation Framework Using Large Language Model and Past Data. *International Conference on Cement and Building Koncrete for a Sustainable and Resilient Infrastructure*.

Google. (2024). *Gemini Apps FAQ*. https://gemini.google.com/faq

Hussain, R., Sabir, A., Lee, D. Y., Zaidi, S. F. A., Pedro, A., Abbas, M. S., & Park, C. (2024). Conversational AI-based VR system to improve construction safety training of migrant workers. *Automation in Construction*, *160*. https://doi.org/10.1016/j.autcon.2024.105315

Jang, S., & Lee, G. (2023). Interactive Design by Integrating a Large Pre-Trained Language Model and Building Information Modeling. In *Computing in Civil Engineering 2023* (pp. 291–299). https://doi.org/10.1061/9780784485231.035

Lin, T. H., Huang, Y. H., & Putranto, A. (2022). Intelligent question and answer system for building information modeling and artificial intelligence of things based on the bidirectional encoder representations from transformers model. *Automation in Construction*, *142*. https://doi.org/10.1016/j.autcon.2022.104483

Lin, W. Y. (2023). Prototyping a Chatbot for Site Managers Using Building Information Modeling (BIM) and Natural Language Understanding (NLU) Techniques. *Sensors*, *23*(6).

https://doi.org/10.3390/s23062942

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. http://arxiv.org/abs/1907.11692

Locatelli, M., Seghezzi, E., Pellegrini, L., Tagliabue, L. C., & Di Giuda, G. M. (2021). Exploring natural language processing in construction and integration with building information modeling: A scientometric analysis. *Buildings*, *11*(12). https://doi.org/10.3390/buildings11120583

MongoDB. (2024). *Understanding Large Language Models (LLMs) in the World of Artificial Intelligence*. https://www.mongodb.com/basics/large-language-models

Nabavi, A., Ramaji, I., Sadeghi, N., & Anderson, A. (2023). Leveraging Natural Language Processing for Automated Information Inquiry from Building Information Models. *Journal of Information Technology in Construction*, *28*, 266–285. https://doi.org/10.36680/J.ITCON.2023.013

OpenAI. (2024a). *Fine-tuning*. https://platform.openai.com/docs/guides/fine-tuning

OpenAI. (2024b). *GPT-4*. https://openai.com/gpt-4

OpenAI. (2024c). *Prompt Engineering*. https://platform.openai.com/docs/guides/prompt-engineering

OpenAI. (2024d, February 8). *Assistants Tools*. https://platform.openai.com/docs/assistants/tools

Orooje, M. S., & Latifi, M. M. (2021). A Review of Embedding Artificial Intelligence in Internet of Things and Building Information Modelling for Healthcare Facility Maintenance Management. *Energy and Environment Research*, *11*(2), 31. https://doi.org/10.5539/eer.v11n2p31

Pan, Y., & Zhang, L. (2021). Roles of artificial intelligence in construction engineering and management: A critical review and future trends. *Automation in Construction*, *122*, 103517. https://doi.org/10.1016/j.autcon.2020.103517

Pedral Sampaio, R., Aguiar Costa, A., & Flores-Colen, I. (2022). A Systematic Review of Artificial Intelligence Applied to Facility Management in the Building Information Modeling Context and Future Research Directions. In *Buildings* (Vol. 12, Issue 11). MDPI. https://doi.org/10.3390/buildings12111939

Prieto, S. A., Mengiste, E. T., & García de Soto, B. (2023). Investigating the Use of ChatGPT for the Scheduling of Construction Projects. *Buildings*, *13*(4). https://doi.org/10.3390/buildings13040857

Rampini, L., & Re Cecconi, F. (2022). Artificial intelligence in construction asset management: a review of present status, challenges and future opportunities. *Journal of Information Technology in Construction*, *27*, 884–913. https://doi.org/10.36680/j.itcon.2022.043

Rane, N., Choudhary, S., & Rane, J. (2023). Integrating Building Information Modelling (BIM) with ChatGPT, Bard, and similar generative artificial intelligence in the architecture, engineering, and construction industry: applications, a novel framework, challenges, and future scope. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4645601

Rane, N. L., Choudhary, S. P., & Rane, J. (2023). Integrating ChatGPT, Bard, and Leading-Edge Generative Artificial Intelligence in Architectural Design and Engineering: Applications, Framework, and Challenges. *International Journal of Architecture and Planning*, *3*(2), 92–124. https://doi.org/10.51483/IJARP.3.2.2023.92-124

Sacks, R., Eastman, C., Lee, G., & Teicholz, P. (2018). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors* (3rd

ed.). Wiley.

Saka, A. B., Oyedele, L. O., Akanbi, L. A., Ganiyu, S. A., Chan, D. W. M., & Bello, S. A. (2023). Conversational artificial intelligence in the AEC industry: A review of present status, challenges and opportunities. In *Advanced Engineering Informatics* (Vol. 55). Elsevier Ltd. https://doi.org/10.1016/j.aei.2022.101869

Saka, A., Taiwo, R., Saka, N., Salami, B. A., Ajayi, S., Akande, K., & Kazemi, H. (2024). GPT models in construction industry: Opportunities, limitations, and a use case validation. In *Developments in the Built Environment* (Vol. 17). Elsevier Ltd. https://doi.org/10.1016/j.dibe.2023.100300

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism*. http://arxiv.org/abs/1909.08053

Singh, J., & Anumba, C. J. (2022). Real-time pipe system installation schedule generation and optimization using artificial intelligence and heuristic techniques. *Journal of Information Technology in Construction*, *27*, 173–190. https://doi.org/10.36680/j.itcon.2022.009

Stephans, T., McClymonds, A., Leicht, R., & Wagner, A. R. (2022, July 15). Automated material selection based on detected construction progress. *2022 Proceedings of the 39th ISARC, Bogotá, Colombia*. https://doi.org/10.22260/ISARC2022/0056

Succar, B. (2009). Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in Construction*, *18*(3), 357–375. https://doi.org/10.1016/j.autcon.2008.10.003

Talarico, G. (2024). *Revit API Docs*. https://www.revitapidocs.com/

Uddin, S. M. J., Albert, A., Ovid, A., & Alsharef, A. (2023). Leveraging ChatGPT to Aid

Construction Hazard Recognition and Support Safety Education and Training. *Sustainability (Switzerland)*, *15*(9). https://doi.org/10.3390/su15097121

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need*. http://arxiv.org/abs/1706.03762

Wang, N., Issa, R. R. A., & Anumba, C. J. (2022a). NLP-Based Query-Answering System for Information Extraction from Building Information Models. *Journal of Computing in Civil Engineering*, *36*(3). https://doi.org/10.1061/(asce)cp.1943-5487.0001019

Wang, N., Issa, R. R. A., & Anumba, C. J. (2022b). Transfer learning-based query classification for intelligent building information spoken dialogue. *Automation in Construction*, *141*. https://doi.org/10.1016/j.autcon.2022.104403

Wang, W., Guo, H., Li, X., Tang, S., Xia, J., & Lv, Z. (2022). Deep learning for assessment of environmental satisfaction using BIM big data in energy efficient building digital twins. *Sustainable Energy Technologies and Assessments*, *50*. https://doi.org/10.1016/j.seta.2021.101897

Wu, J., Xue, X., & Zhang, J. (2023). Invariant Signature, Logic Reasoning, and Semantic Natural Language Processing (NLP)-Based Automated Building Code Compliance Checking (I-SNACC) Framework. *Journal of Information Technology in Construction*, *28*, 1–18. https://doi.org/10.36680/j.itcon.2023.001

Xu, S., Wang, J., Shou, W., Ngo, T., Sadick, A. M., & Wang, X. (2021). Computer Vision Techniques in Construction: A Critical Review. *Archives of Computational Methods in Engineering*, *28*(5), 3383–3397. https://doi.org/10.1007/s11831-020-09504-3

Xue, X., Hou, Y., & Zhang, J. (2022, July 15). Automated Construction Contract Summarization Using Natural Language Processing and Deep Learning. *2022 Proceedings of the 39th*

ISARC, Bogotá, Colombia. https://doi.org/10.22260/ISARC2022/0063

Zabin, A., González, V. A., Zou, Y., & Amor, R. (2022). Applications of machine learning to BIM: A systematic literature review. *Advanced Engineering Informatics*, *51*. https://doi.org/10.1016/j.aei.2021.101474

Zhang, F., Chan, A. P. C., Darko, A., Chen, Z., & Li, D. (2022). Integrated applications of building information modeling and artificial intelligence techniques in the AEC/FM industry. In *Automation in Construction* (Vol. 139). Elsevier B.V. https://doi.org/10.1016/j.autcon.2022.104289

Zheng, J., & Fischer, M. (2023). Dynamic prompt-based virtual assistant framework for BIM information search. *Automation in Construction*, *155*. https://doi.org/10.1016/j.autcon.2023.105067

Appendix A: *Python Component Code Snippet*

A.1. Utility Functions to Manage Bridge Interaction

```python
def load_json(path):
    """
    Loads JSON data from a specified file path.

    Parameters:
    path (str): Path to the JSON file.

    Returns:
    dict or str: Parsed JSON data if successful, or an error message if an exception occurs.
    """
    try:
        with open(path, "r") as file:
            data = json.load(file)
        return data
    except Exception as e:
        print_colored("ERROR", f"An error occured in load_json: {e}")
        return f"Error loading json file: {str(e)}"

def save_json(data, path):
    """
    Saves data to a JSON file at a specified path.

    Parameters:
    data (dict): The data to save.
    path (str): Path to the JSON file where data will be saved.

    Returns:
    None or str: None if successful, or an error message if an exception occurs.
    """
    try:
        with open(path, "w") as file:
            json.dump(data, file, indent=4)
    except Exception as e:
        print_colored("ERROR", f"An error occured in save_json: {e}")
        return f"Error saving json file: {str(e)}"

def save_and_check_errors(action_name, data):
    """
    Saves the updated data to the bridge.json file, reloads it, and checks for errors.
```

```
    Parameters:
    data (dict): The updated data to be saved.
    action_name (str): The name of the action, to access the correct part of the JSON for error checking.

    Returns:
    str: A message indicating the success or failure of the operation.
    """
    # Save the updated data
    save_result = save_json(data, bridge_path)

    # Check if save_bridge returned an error message
    if isinstance(save_result, str):
        return save_result

    while True:
        time.sleep(1) # wait the dll to run

        # Re-load to check for errors
        updated_data = load_json(bridge_path)

        # Check if load_bridge returned an error message
        if isinstance(updated_data, str):
            return updated_data

        # Check for specific action errors
        error = updated_data[action_name]["error"]
        if error:
            break

    if error != "no errors":
        print_colored("SYSTEM", f"An error from bridge for {action_name}: {error}")
        return f"Error in {action_name}: {error}"
    else:
        return f"'{action_name}' action was successful."

def update_bridge(action, parameters):
    """
    Updates a specific section of the bridge.json file based on the provided action and parameters.

    Parameters:
    action (str): The name of the action section to be updated in the bridge.json file
    parameters (dict): A dictionary of parameters to be updated for the specified action.
```

```
    Returns:
    str: A message indicating the success or failure of the operation.
    """
    try:
        data = load_json(bridge_path)

        if isinstance(data, str):
            return data

        if action not in data:
            return f"Error: Action '{action}' not found in bridge.json."

        if parameters is not None:
            # Update the specified action with the provided parameters
            for key, value in parameters.items():
                if key in data[action]:
                    data[action][key] = value
                else:
                    return f"Error: Key '{key}' not found in '{action}' section of bridge.json."

        data[action]["run"] = True
        data[action]["error"] = ""

        return save_and_check_errors(action, data)

    except Exception as e:
        print_colored("ERROR", f"Failed to update {action}: {e}")
        return f"Failed to update '{action}': {str(e)}"
```

A.2 Examples of Bridge Updating Functions

```
def change_transparency(transparency_value=100, mode="selection", items=[], **kwargs):
    """
    Updates the 'transparency' section of the bridge.json file.

    Parameters:
    transparency_value (int): The transparency value to set.
    mode (str): The mode of application for transparency, which can be "selection", "element", or
"category".
    items (list): A list of element IDs (as strings) or categories, based on the specified 'mode', to
which transparency is to be applied.

    Returns:
```

```
    str: A message indicating the success or failure of the operation.
    """

    print_colored(
        "SYSTEM",
        f"update_transparency called with value: {transparency_value}, mode: {mode}, items: {items}"
    )

    # Validate that items is a list
    if not isinstance(items, list):
        return "Error: 'items' must be a list."

    if mode not in ["selection", "element", "category"]:
        return "Error: 'mode' must be 'selection', 'element' or 'category' only."

    if mode in ["element", "category"] and not items:
        return "Error: 'items' should not be empty"

    # Create a dictionary with the necessary parameters
    parameters = {
        "transparency_value": transparency_value,
        "mode": mode,
        "items": items
    }

    return update_bridge("transparency", parameters)

def isolate(mode="selection", items=[], reset=False, **kwargs):
    """
    Updates the 'isolation' section of the bridge.json file.

    Parameters:
    mode (str): The mode of application for transparency, which can be "selection", "element", or
"category".
    items (list): A list of element IDs (as strings) or categories, based on the specified 'mode', to
which isolation is to be applied.
    reset (bool): Resets the temporary isolation in the current Revit view.

    Returns:
    str: A message indicating the success or failure of the operation.
    """

    print_colored(
        "SYSTEM",
```

```
        f"update_isolation called with mode: {mode}, items: {items}, reset: {reset}"
    )

    # Validate that items is a list
    if not isinstance(items, list):
        return "Error: 'items' must be a list."

    if mode not in ["selection", "element", "category"]:
        return "Error: 'mode' must be 'selection', 'element' or 'category' only."

    if mode in ["element", "category"] and not items:
        return "Error: 'items' should not be empty"

    if reset:
        parameters = {"reset": reset}
    else:
        # Create a dictionary with the necessary parameters
        parameters = {
            "mode": mode,
            "items": items
        }

    return update_bridge("isolation", parameters)
```

Appendix B: *Bridge Component Code Snippet*

B.1. Examples of Bridge JSON Schema

```
{
  "transparency": {
    "mode": "",
    "transparency_value": 0,
    "run": false,
    "items": [],
    "error": ""
  },
  "isolation": {
    "mode": "",
    "run": false,
    "reset": false,
    "items": [],
    "error": ""
  },
```

Appendix C: *C# Component Code Snippet*

C.1. File Watcher Class

```
    public class RevitFileWatcherApplication : IExternalApplication
    {
        private FileSystemWatcher _watcher;
        private ExternalEvent _externalEvent;
        private FileChangeEventHandler _handler;
        private Timer _debounceTimer;

        public Result OnStartup(UIControlledApplication application)
        {
            // Initialize the debounce timer
            _debounceTimer = new Timer(1000); // Set to 1 second
            _debounceTimer.Elapsed += OnDebounceTimerElapsed;
            _debounceTimer.AutoReset = false; // So that it only runs once per trigger

            // Retrieve the path from environment variable
            string jsonFolderPath = Environment.GetEnvironmentVariable("BRIDGE_JSON_PATH");

            if (string.IsNullOrEmpty(jsonFolderPath))
            {
                TaskDialog.Show("Configuration Error", "The environment variable BRIDGE_JSON_PATH is not
set. Please set the variable and restart Revit.");
                return Result.Failed;
            }

            _watcher = new FileSystemWatcher
            {
                Path = jsonFolderPath,
                Filter = "bridge.json",
                NotifyFilter = NotifyFilters.LastWrite
            };
            _watcher.Changed += HandleFileChange;
            _watcher.EnableRaisingEvents = true;

            _handler = new FileChangeEventHandler();
            _externalEvent = ExternalEvent.Create(_handler);

            return Result.Succeeded;
        }
```

```
        public Result OnShutdown(UIControlledApplication application)
        {
            _watcher?.Dispose();
            _debounceTimer?.Dispose();

            return Result.Succeeded;
        }

        private void HandleFileChange(object source, FileSystemEventArgs e)
        {
            // Restart the debounce timer
            _debounceTimer.Stop();
            _debounceTimer.Start();
        }

        private void OnDebounceTimerElapsed(object sender, ElapsedEventArgs e)
        {
            // Invoke the external event
            _externalEvent.Raise();
        }
    }
```

C.2. Example of Revit Action Method

```
private void Transparency(UIApplication app, JObject jsonData)
{
    var data = jsonData["transparency"];
    string mode = (string)data["mode"];

    try
    {
        Document doc = app.ActiveUIDocument.Document;
        int transparencyValue = (int)data["transparency_value"];

        if (mode == "category")
        {
            using (Transaction t = new Transaction(doc, "Apply Category Transparency"))
            {
                t.Start();

                List<string> categories = data["items"].ToObject<List<string>>();
                foreach (string categoryName in categories)
                {
```

```
                    Category category = GetCategoryByName(doc, categoryName);
                    if (category != null)
                    {
                        OverrideGraphicSettings overrideSettings = new OverrideGraphicSettings();
                        overrideSettings.SetSurfaceTransparency(transparencyValue);
                        doc.ActiveView.SetCategoryOverrides(category.Id, overrideSettings);
                    }
                }
                t.Commit();
            }
        }
        else if (mode == "element" || mode == "selection")
        {
            List<int> elementIds = mode == "element"
                ? data["items"].ToObject<List<int>>()
                : app.ActiveUIDocument.Selection.GetElementIds().Select(id => id.IntegerValue).ToList();

            ApplyElementTransparency(doc, elementIds, transparencyValue);
        }
        jsonData["transparency"]["error"] = "no errors";
    }
    catch (Exception ex)
    {
        jsonData["transparency"]["error"] = $"Failed to apply transparency: {ex.Message}";
    }
    finally
    {
        // Update JSON to set "run" to false for transparency
        data["run"] = false;
    }
}

private void Isolation(UIApplication app, JObject jsonData)
{
    var data = jsonData["isolation"];
    bool reset = (bool)data["reset"];
    string mode = (string)data["mode"];
    try
    {
        if (reset)
        {
            ResetIsolation(app.ActiveUIDocument.Document);
        }
        else
```

```
        {
            Document doc = app.ActiveUIDocument.Document;
            View view = doc.ActiveView;

            switch (mode)
            {
                case "category":
                    List<string> categoriesToIsolate = data["items"].ToObject<List<string>>();
                    ApplyIsolation(doc, categoriesToIsolate);
                    break;
                case "element":
                    List<int> elementIds = data["items"].ToObject<List<int>>();
                    ApplyElementIsolation(doc, elementIds);
                    break;
                case "selection":
                    ICollection<ElementId> selectedIds = app.ActiveUIDocument.Selection.GetElementIds();
                    if (selectedIds.Count > 0)
                    {
                        List<int> selectedElementIds = selectedIds.Select(id => id.IntegerValue).ToList();
                        ApplyElementIsolation(doc, selectedElementIds);
                    }
                    else
                    {
                        jsonData["isolation"]["error"] = "No elements selected for isolation.";
                    }
                    break;
                default:
                    jsonData["isolation"]["error"] = $"Invalid isolation type: {mode}";
                    break;
            }
            jsonData["isolation"]["error"] = "no errors";
        }
    }
    catch (Exception ex)
    {
        jsonData["isolation"]["error"] = $"Failed to isolate: {ex.Message}";
    }
    finally
    {
        data["run"] = false;
        if (reset) data["reset"] = false;
    }
}
```