buildings

MDPI

Article

A GPT-Powered Assistant for Real-Time Interaction with Building Information Models

David Fernandes [1], Sahej Garg [2], Matthew Nikkel [1] and Gursans Guven [1,*]

[1] Department of Civil Engineering, Price Faculty of Engineering, University of Manitoba, Winnipeg, MB R3T 5V6, Canada; fernan68@myumanitoba.ca (D.F.); nikkelm1@myumanitoba.ca (M.N.)
[2] Department of Electrical and Computer Engineering, Price Faculty of Engineering, University of Manitoba, Winnipeg, MB R3T 5V6, Canada; gargs4@myumanitoba.ca
* Correspondence: gursans.guvenisin@umanitoba.ca

Abstract: This study introduces DAVE (Digital Assistant for Virtual Engineering), a Generative Pre-trained Transformer (GPT)-powered digital assistant prototype, designed to enable real-time, multimodal interactions within Building Information Modeling (BIM) environments for updating and querying BIM models using text or voice commands. DAVE integrates directly with Autodesk Revit through Python scripts, the Revit API, and the OpenAI API and utilizes Natural Language Processing (NLP). This study presents (1) the development of a practical AI chatbot application that leverages conversational AI and BIM for dynamic actions within BIM models (e.g., updates and queries) at any stage of a construction project and (2) the demonstration of real-time, multimodal BIM model management through voice or text, which aims to reduce the complexity and technical barriers typically associated with BIM processes. The details of DAVE's development and system architecture are outlined in this paper. Additionally, the comprehensive process of prototype testing and evaluation including the response time analysis and error analysis, which investigated the issues encountered during system validation, are detailed. The prototype demonstrated 94% success in accurately processing and executing single-function user queries. By enabling conversational interactions with BIM models, DAVE represents a significant contribution to the current body of knowledge.

Keywords: Building Information Modeling; BIM; Artificial Intelligence; conversational AI; natural language processing; Generative Pre-trained Transformer; GPT; virtual assistant

Citation: Fernandes, D.; Garg, S.; Nikkel, M.; Guven, G. A GPT-Powered Assistant for Real-Time Interaction with Building Information Models. *Buildings* **2024**, *14*, 2499. https://doi.org/10.3390/buildings14082499

Academic Editors: Zhen Liu and Mohamed Osmani

Received: 19 July 2024
Revised: 9 August 2024
Accepted: 10 August 2024
Published: 13 August 2024

1. Introduction

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 [1]. 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, significant barriers to BIM adoption remain worldwide. Among those are the lack of knowledge and training in BIM, software complexity and steep learning curves, and the complexity of BIM models [2–4].

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) and various areas of application, such as Natural Language Processing (NLP), Computer Vision (CV), and robotics, has been instrumental in revolutionizing industries by automating operations, enhancing data analysis, and supporting decision-making processes [5]. The introduction of AI in the AEC industry, particularly in the field of construction engineering and management, can enhance decision making and increase efficiency and bring digitalization and automation [6,7].

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 [8,9]. By enabling natural language interfaces, digital assistants can potentially simplify the interaction with BIM models and make them more intuitive and accessible [10,11]. Large Language Models (LLMs), such as the OpenAI Generative Pre-trained Transformer (GPT) series [12] and Google Gemini [13], 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 [14]. LLMs can understand and produce natural language at a level that is often indistinguishable from human output. This makes 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. In the context of the AEC industry and particularly in its integration with BIM, LLMs have introduced novel opportunities for interaction within BIM models for various purposes, enabling even those without extensive technical expertise to engage with BIM through natural language interfaces. Current studies have focused on, for instance, information retrieval [10,15,16], scheduling [17,18], automated compliance-checking [19,20], safety rule checking [21], and interactive design [22,23].

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 text or voice commands. This dynamic interaction aims to transform the typical use of BIM models and BIM workflows, making them more responsive and adaptable to users' needs. The architecture of the developed system combines Python scripting with 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. It also addresses the practical needs of AEC professionals who often struggle with the complexity of interacting with BIM models by making it a more user-friendly and efficient experience.

This study contributes to the ongoing digital transformation of the AEC industry by demonstrating the feasibility of integrating conversational AI with BIM. The main contributions of this research include (1) a practical tool for BIM users developed for daily use by leveraging conversational AI for dynamic updates and queries within BIM models, and (2) real-time BIM model management through voice or text to reduce the complexity and technical barriers traditionally associated with BIM. This research contributes to the efforts in making the future built environment more connected, intelligent, and user-centric.

The paper is structured as follows: Section 2 presents a review of the related literature and comparison of DAVE with relevant studies, Section 3 outlines the research objectives and scope, Section 4 describes the proposed prototype, including its architecture and workflow, Section 5 details the testing and validation processes, Section 6 discusses the results and implications, and finally, Section 7 concludes the paper with a summary of findings and future research directions.

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. First, an overview of the general applications of AI in BIM (Section 2.1) is presented, followed by a discussion on the specific roles of chatbots and NLP (Section 2.2), and the innovative uses of LLMs (Section 2.3) in facilitating real-time interactions and management within BIM

environments. Finally, the novelty and originality of DAVE are presented in the form of a comparison with some of the highly relevant studies in the literature (Section 2.4).

2.1. AI Applications in BIM

The evolution of AI technologies has opened new avenues for enhancing BIM functionalities. For instance, 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 [24]. 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 [25].

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 [26]. 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 [27]. Another application includes the use of AI for automatic material selection, aligning choices with the current progress of the project to optimize resource allocation [28]. Additionally, AI's role extends to schedule generation and optimization, where it streamlines project timelines and improves efficiency [29].

Furthermore, the integration of the Internet of Things (IoT) with AI and BIM technologies has introduced a new dimension to real-time data collection and analysis in the AEC industry. IoT devices, such as sensors and actuators embedded within buildings, facilitate the continuous monitoring of structural health, environmental conditions, and energy consumption [30]. These 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 [31].

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 [32]. 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 material usage [22]. In the construction phase, AI has proven instrumental in progress tracking and quality control [5]. Finally, in facility management, AI applications extend to energy management, predictive maintenance models, and automatic as-built classification [31]. Future research is set to focus 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 [32].

2.2. Chatbots and NLP Applications in 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 [33]. 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 [34] and aiming to make BIM more user-oriented [35].

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 information retrieval [10,15,36–38] and question and answer systems [39], construction contract summarization [40], compliance

checking [19,41], and live instructor for safety training [42], showcasing the versatility and potential of chatbots in streamlining BIM processes.

The integration of natural language 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 [35,43]. 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 [1,3,4]. Lastly, the use of chatbots for routine tasks and queries can free up human resources for the more complex and creative aspects of project management and design [44].

Despite the promising advancements, the application of NLP and chatbots in BIM has its challenges [45]. 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 [33]. This also highlights the importance of prompt engineering, a process of strategically crafting inputs 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 [46]. Section 6.2.3 demonstrates how prompt engineering is an essential part of the deployment of DAVE.

2.3. The Use of LLMs in BIM

LLMs like GPT and Gemini (formerly known as Bard) have been at the forefront of AI research due to their ability to understand, generate, and interpret human language with remarkable accuracy [14]. 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 [47].

The integration of LLMs into BIM is still in its early stages, but significant advancements have already been noted such as generating innovative design alternatives, streamlining communication among stakeholders, and optimizing construction schedules [17,47]. The capacities of these models to aid in education and training are also being investigated [48]. 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 [23] while demonstrating significant improvements in efficiency and quality of deliverables.

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 [23,47,49]. 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 [22]. The application of LLMs in BIM holds great promise for transforming the AEC industry, and 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.

2.4. Novelty and Originality of the Work

To summarize the key differences with existing studies and highlight the novelty and originality of DAVE, Table 1 provides a comparison of DAVE with three other highly

relevant studies in the literature [9–11] that also developed prototypes for BIM interaction via human language (by using GPT models or Amazon Alexa). Firstly, DAVE is designed as a comprehensive virtual assistant for BIM interaction and management. It is applicable to all phases of a construction project and is tested with multidisciplinary BIM models. This is in contrast to the other studies, which have more limited scopes where one study developed a dynamic prompt-based virtual assistant for BIM information search specifically for a hospital building [10], another was developed for the design phase of a simple BIM model [9], and the third was used for developing room schedules as a proof of concept of an AI voice assistant integrated with Dynamo [11]. Additionally, DAVE can perform 12 different actions within BIM to manage and interact with the model and more can be added as needed, whereas other studies are limited to information retrieval [10] and developing room schedules [11] or focused solely on material selection and recommendations [9]. Furthermore, DAVE stands out by supporting both voice and text commands for user input, enhancing accessibility and user interaction. The use of voice commands in DAVE also makes it suitable for use within Virtual Reality (VR) environments, as demonstrated in [50]. Finally, DAVE utilizes both GPT-4 Turbo and GPT-3.5 Turbo for managing BIMs, while [9,10] used the GPT-3.5 model and [11] used Amazon Alexa. Overall, DAVE represents a significant contribution to the current body of knowledge by enabling BIM information retrieval, updates, and querying through voice or text commands. DAVE's multifunctional capabilities and multimodal user interaction (i.e., text, voice, and 3D visualization) collectively position it as a versatile and powerful tool among the existing tools.

Table 1. Comparison of DAVE to relevant studies in the literature.

	DAVE	Zheng and Fischer (2023) [10]	Saka et al. (2024) [9]	Elghaish et al. (2022) [11]
Focus	A comprehensive virtual assistant for BIM interaction and management	Dynamic prompt-based virtual assistant prototype for BIM information search	A material selection and optimization prototype for BIMs	AI voice assistant integrated with Dynamo to interact with BIMs
Scope	Prototype developed with multidisciplinary BIM models, applicable to all phases of a project	Prototype developed for a hospital building	Prototype validated in the design phase of a simple BIM model	A room schedule created from a complex BIM model as a proof of concept
Method	Manage BIMs (information retrieval and update) by running GPT Assistants	Retrieve information from a BIM using natural language	Combining BIM data with GPT models to select material for construction projects	Retrieve information from BIM models by reading CSV data from Dynamo
User input type	Voice or text commands	Text commands	Text commands	Voice commands
Function	Information retrieval, updating/modifying and querying BIMs	Querying and information search from BIMs	Selecting the best material for a specific component in a BIM	Information retrieval and interaction with BIMs
Actions	Performs 12 different actions in BIMs, supports addition of more actions as needed	Only information retrieval from BIMs	Only material selection or recommendations in BIMs	Limited to developing room schedules in BIMs
LLM	GPT-3.5 Turbo and GPT-4 Turbo	GPT-3.5 Turbo	GPT-3.5 Turbo	Amazon Alexa

3. Research Objectives and Scope

The primary goal of this study is to develop and evaluate a prototype of an AI chatbot for dynamic updates and multimodal interactions within BIM environments. This approach aims to enhance the efficiency and effectiveness of BIM workflows by leveraging the capabilities of an LLM, namely GPT-4. It extends beyond the typical use of chatbots

in the literature for BIM information retrieval to include real-time model updates and interactions via text or speech.

The developed prototype integrates with a specific, yet widely adopted, BIM-authoring software (i.e., Autodesk Revit), showcasing the practical application of AI, more specifically NLP, in real-world BIM tasks. The prototype can perform 12 critical and commonly used actions within the BIM environment. Each one of them is selected based on the daily tasks performed by a typical Virtual Design and Construction department in a construction company and the mainstream uses of Revit. They range from simple tasks like undoing the last action to more complex tasks such as changing the family type of a selected element and updating room names, numbers, and occupants. The prototype architecture can be adapted to work with other BIM platforms and other LLMs with a few adjustments to the current function implementation to adapt to the new syntax as well as input/output formats.

The novelty and originality of this study further lie in the prototype's applicability to all phases of a construction project from pre-design and design to construction and demolition. Additionally, its capabilities have been demonstrated on a basic sample building model as well as a multidisciplinary real-world office building model proving the prototype's versatility across different project requirements. The current limitations of the prototype and some specific challenges, such as scalability and system efficiency, are discussed in Section 6, and areas for future improvements and research are identified.

4. Proposed Prototype

4.1. System Architecture Overview

Figure 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, Autodesk Revit. The system possesses a preliminary stage called Data Extraction, further explained in Section 4.1.1. During this stage, metadata of the building elements are extracted from the Revit project file (*.rvt*) and converted into a JSON model derivative. These JSON data are then transformed into CSV format for efficient information retrieval. This conversion from JSON to CSV helps prevent potential data loss that might occur during a direct conversion of the Revit file into CSV. Additionally, the CSV format makes it simpler and faster to query specific data points and facilitates smoother integration into AI systems because of the flat, tabular structure it provides.

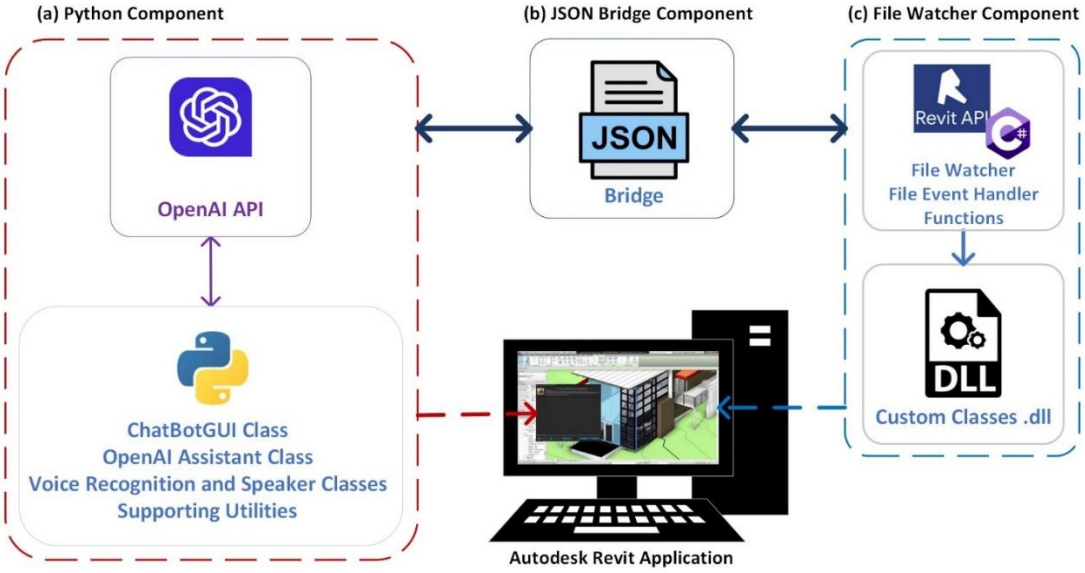


Figure 1. DAVE's system architecture overview.

The developed system consists of three main components as illustrated in Figure 1. The core of the prototype is the Python component (Figure 1a), which is empowered by the 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 4.1.2. The JSON Bridge Component (Figure 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 4.1.3. Finally, the File Watcher component (Figure 1c) is crafted in C# and dynamically monitors changes in 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 4.1.4. All the system components and their roles are summarized in Table 2.

Table 2. 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 speech 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.

4.1.1. Data Extraction Component

As depicted in Figure 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 is initiated with the extraction of data from the Revit files using the Autodesk Platform Services (APS) API. These data that are stored initially in a cloud-based bucket consist of model derivative and tree information and are downloaded as JSON files. These files are then cleaned and manipulated into a singular CSV file, which is integrated into the AI assistant. This approach ensures that each assistant is closely linked to a specific project database, enabling customized interactions and responses.

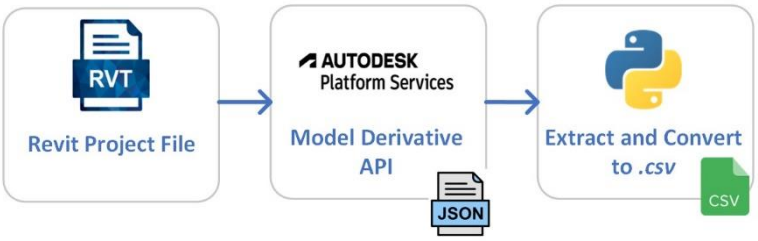


Figure 2. Data extraction component of the prototype.

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 multidisciplinary model of a real-world office building in Canada including architecture, structural, mechanical, and electrical projects.

The main part of 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 *tree.json* file outlines the hierarchical structure of objects within a Revit model and contains nested objects, each with its own *objectid* and potentially a *name*. The *properties.json* file provides the detailed properties for specific objects referenced in *tree.json*. Each object has various attributes categorized under different sections. The script navigates the hierarchical tree structure, extracting pertinent object data and key pieces of information that are important for analysis. These include *objectid*, *name*, *Family Type, Family Name,* and *Category*, and attributes like *Type Name* and *Assembly Name*, construction details, and some Industry Foundation Classes (IFC) parameters, such as IFC Globally Unique Identifier (GUID) (*IfcGuid*) for unique identification. The script further refines the dataset, eliminating irrelevant or empty properties and adjusting specific element attributes. Some superfluous or redundant data that are blank, default, or irrelevant for the current analysis or application (e.g., fields with values like “”, “0.000”, or null, comments and descriptions, URLs, and images) are cleaned and filtered in this step. The data are organized with the *IfcGuid* 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 processed data are 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 resulting structured data are then saved as a CSV file.

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 the assistant API context, executes Python code by itself in a secure, sandboxed environment, allowing for data processing [51]. 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 precision and relevance.

4.1.2. Python Component

The Python component (Figure 1a) constitutes 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 speech 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 presents how the different modules (i.e., Python files) and classes interact with each other within the DAVE system.

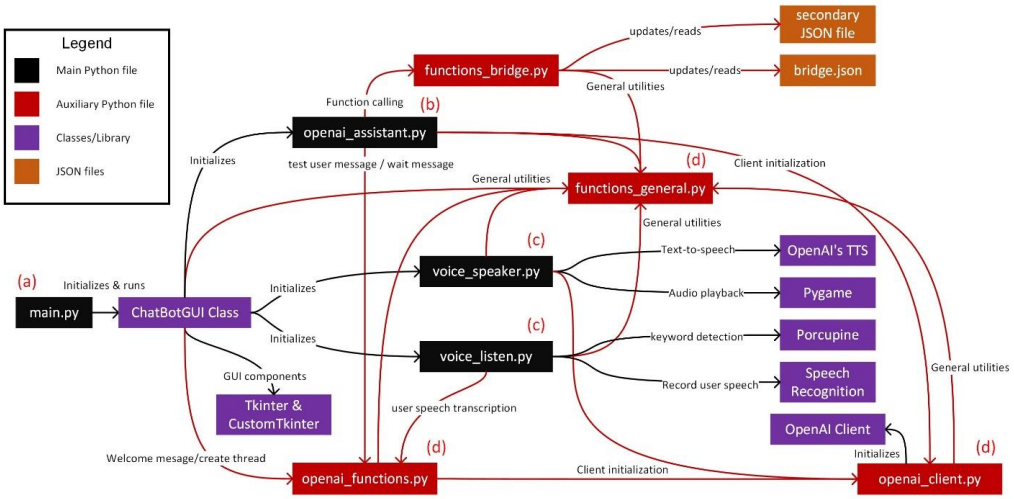


Figure 3. Dependency diagram of DAVE's Python component.

a) ChatBotGUI Class (Figure 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 the GUI and the API. The class also handles the integration of audio systems, enabling speech recognition and text-to-speech functionalities by instantiating their respective classes. The GUI (Figure 4) is designed to be intuitive, allowing users to interact with Revit through both text and voice commands.

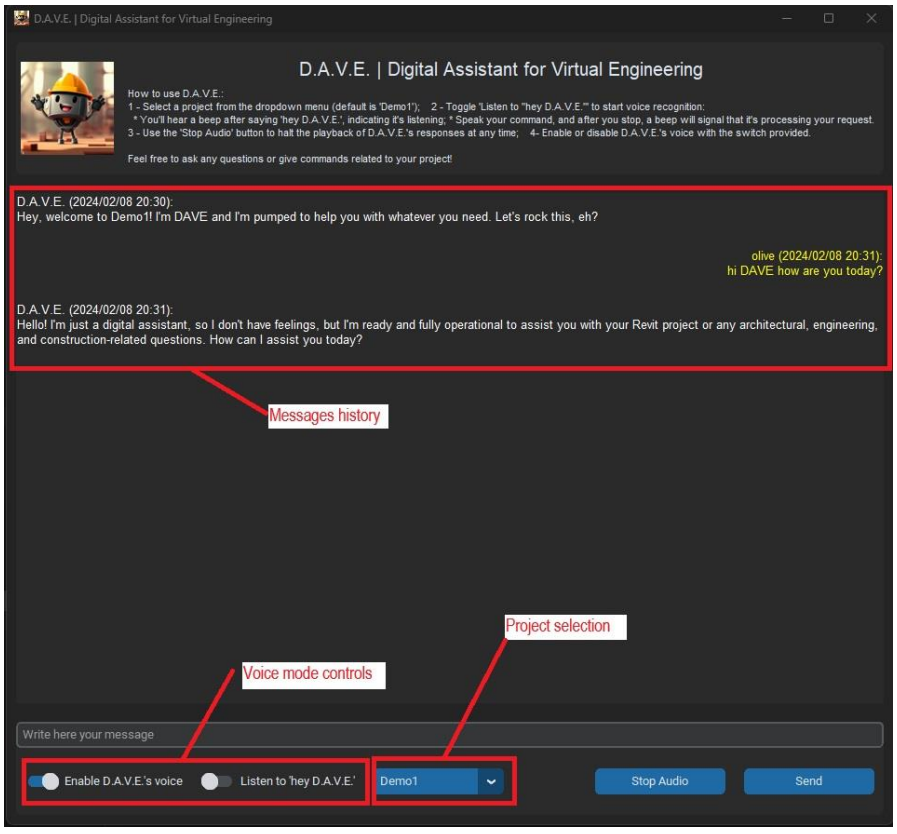


Figure 4. DAVE's GUI.

b) OpenAI Assistant Class (Figure 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. To be able to call the right function, all of them are incorporated into the GPT Assistant during its creation (Figure 5), and each function is set up using a JSON schema as illustrated in Figure 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. Python to C# functions and methods mapping.

C# Method Name	Description	Triggered by (Python Function)	Python Arguments
Transparency	Applies transparency elements or categories	change_transparency	transparency_value (int), mode (str), items (list)
Isolation	Isolates elements or categories	isolate	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)

c) Speech Recognition and Speaker Classes (Figure 3c)

The speech 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.

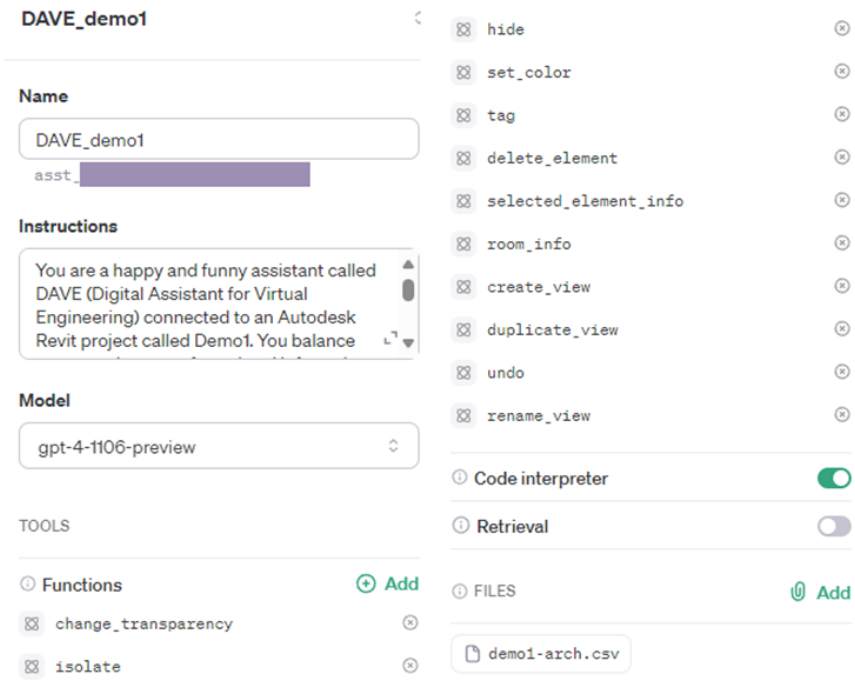


Figure 5. Assistant DAVE_demo1 setup on the OpenAI Playground website.

```
{
  "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 6. JSON Schema for the function '*change_transparency*' incorporated as a tool of the assistant.

d) Supporting Utilities (Figure 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 seamless integration with OpenAI's services for enhanced operational efficiency.

Furthermore, the *openai_functions.py* script is equipped with specialized functions tailored to interact with the OpenAI API. It includes capabilities for thread creation, test user queries, and the generation of various 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 3500 tokens to load the instructions—we employ a method known as "few-shot learning" (i.e., to learn from a limited number of examples). This approach not only generates standard responses in DAVE but also allows the prototype to pre-classify user queries by making an additional API call to OpenAI utilizing the GPT-3.5-turbo-1106 model and determine the appropriate response type (e.g., "welcome" message, "wait" message or "unrelated input" message). This way, our model avoids unnecessary generation steps and helps to optimize the token usage. Our model is trained on a few examples to recognize various query types and can handle irrelevant, out-of-scope queries by quickly identifying them and providing a standard response, 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.

4.1.3. JSON Bridge Component

The JSON Bridge Component, as illustrated in Figure 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", and "color"), and the *bridge.json* schema is designed to encompass all 12 functions presented previously in Table 3. Each key within the bridge is associated with specific parameters detailing the action's mode, execution state (run), and potential target items or values.

4.1.4. File Watcher—C# Revit API Component

The C# component, as shown in Figure 1c, was developed in Visual Studio. It operates on the .NET Framework 4.8 and consists 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. 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. The 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 the 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., changing the family type of an element or changing 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 the 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

4.2. System Workflow

Building upon the foundational architecture outlined in Section 4.1, this section delves into the detailed workflow of the system, explaining the step-by-step interactions between the user, the GPT-powered assistant, and Revit. A general flowchart (Figure 7) provides an overarching view of the process, while subsequent System Sequence Diagrams (SSDs) capture the specifics of four distinct system states.

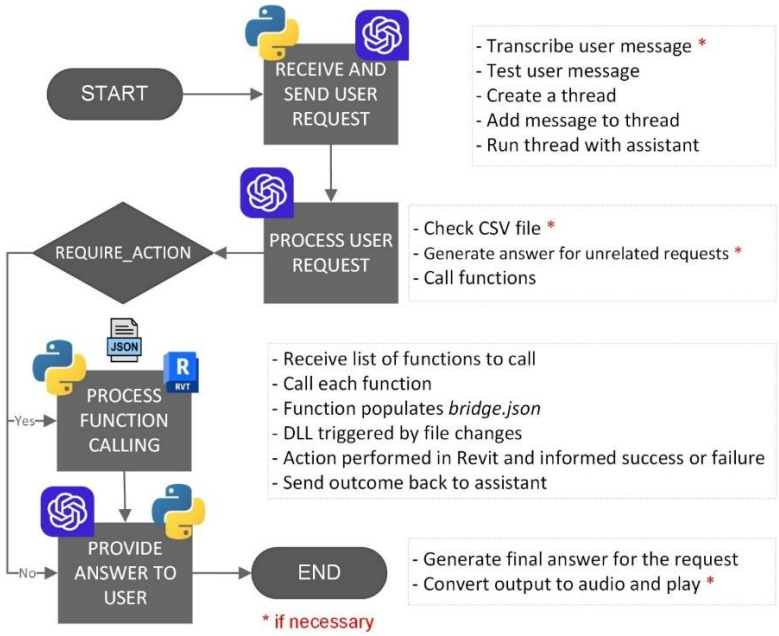


Figure 7. DAVE's workflow diagram.

4.2.1. 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 7) offers a comprehensive view of this interaction, illustrating the pathways between system components and the user.

4.2.2. 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 4.1.2. 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%".

4.2.3. Command Interpretation and Function Invocation

The Python backend creates a *thread* that 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 as part of the prompt engineering objectives is discussed later in the discussion section (Section 6).

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 8 details the process for unrelated or database inquiries and the one in Figure 9 details Revit-related actions.

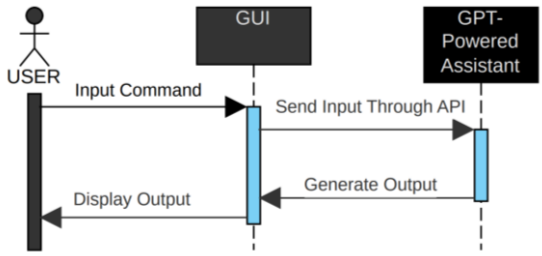


Figure 8. System Sequence Diagram of the unrelated or database inquiries process.

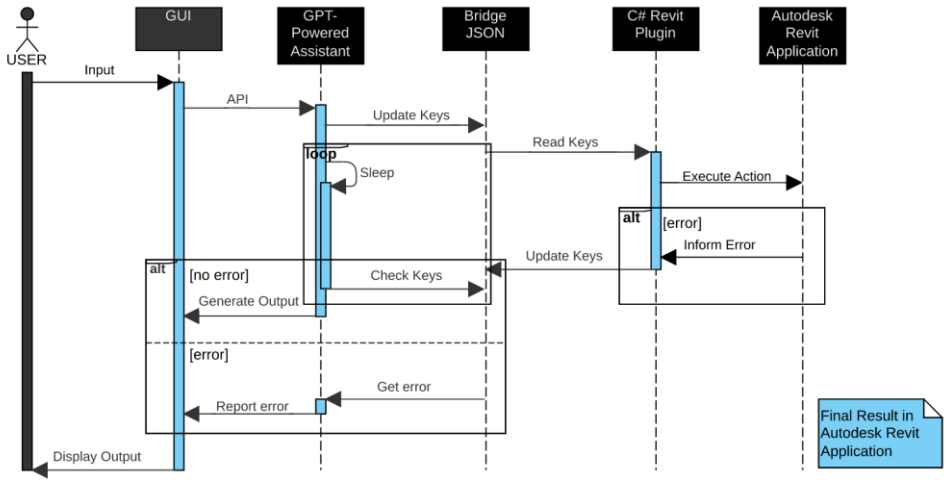


Figure 9. System Sequence Diagram of the general Revit-related actions process.

Continuing the previous scenario, after the user's message is loaded into the thread and the GPT Assistant runs on it, the OpenAI Assistant class (Section 4.1.2.b) has a method that will catch the *requires_action* status and the following function to call (Figure 10).

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

Figure 10. Example of the list of functions to be called produced by the GPT Assistant.

4.2.4. 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 11.

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

Figure 11. Example of *bridge.json* schema after a called function run.

4.2.5. 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 is triggered, it then realizes the true value of the run key of *transparency*. This would invoke the *Transparency* method 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 (Figures 12 and 13) illustrate these operations, showcasing the system's approach to handling complex requests without overburdening the *bridge.json* file.

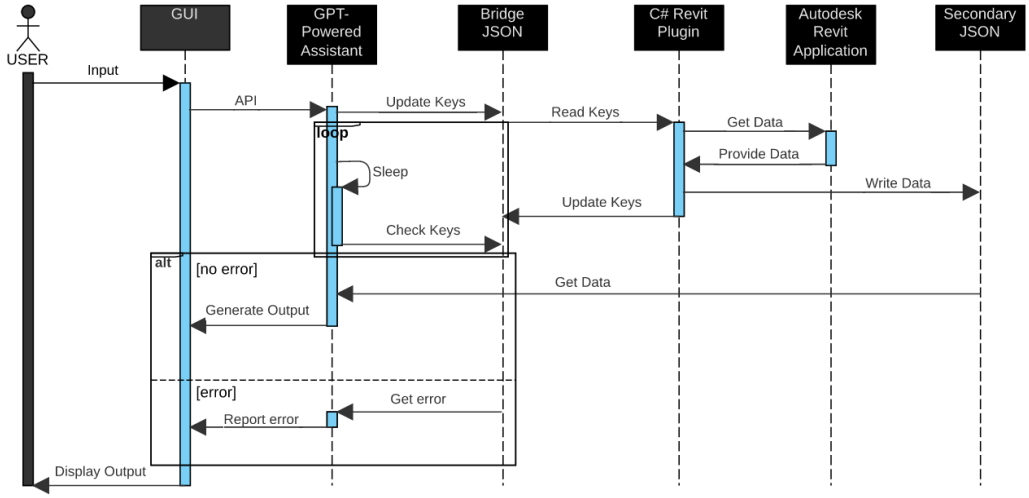


Figure 12. System Sequence Diagram of the *write* operation process.

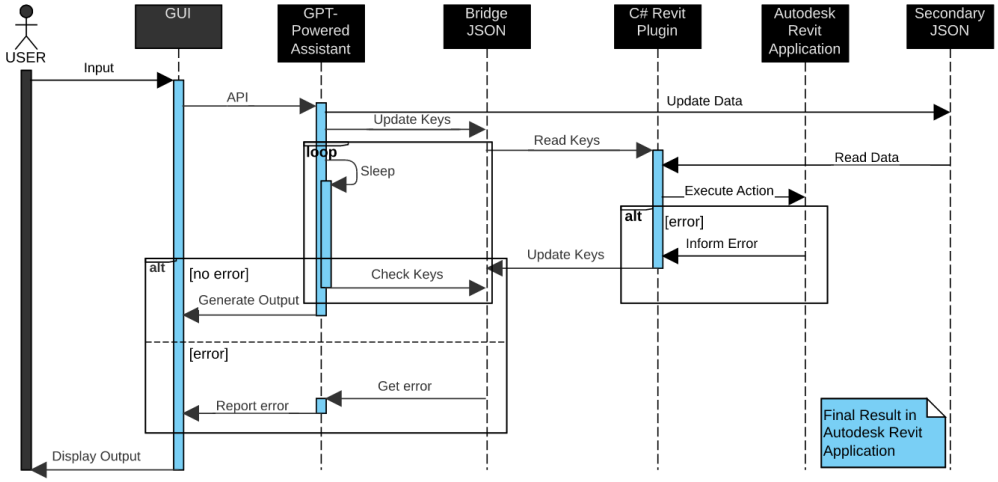


Figure 13. System Sequence Diagram of the *read* operation process.

4.2.6. 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 14 presents the system logs for the illustrative example (i.e., set wall transparency to 100%) including the final answer generated by the GPT Assistant (a) and the result in Revit (b) (i.e., walls set to 100% transparency).

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.

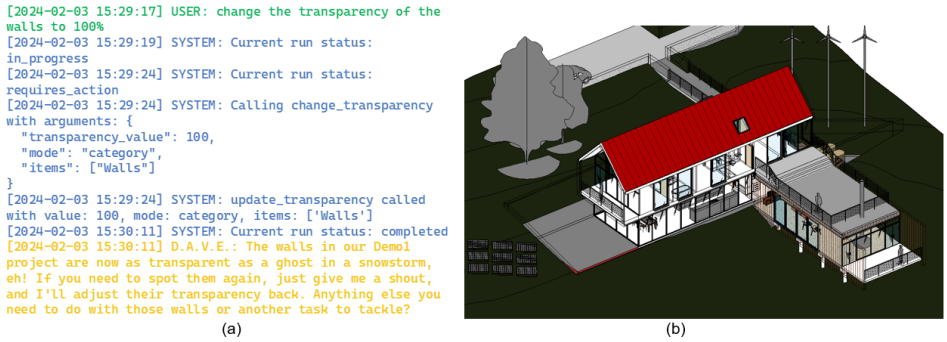


Figure 14. (**a**) System logs for the exemplified user request (set wall transparency to 100%) and (**b**) result in Revit application with all walls set to 100% transparency.

5. Validation

5.1. Validation Methodology

To ensure the robustness and reliability of DAVE, we adopted a testing and validation methodology that is tailored to assess both the system's functional accuracy and its efficiency in handling complex queries. Our approach was structured into distinct phases, each designed to evaluate the 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. Table 4 provides a list of example queries categorized by the corresponding functions. This set of queries aimed to cover a broad spectrum of the prototype's potential use cases, ensuring a comprehensive assessment of its functional accuracy.

To extend our testing to more complex scenarios, we 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 [51] 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. The collection of these data was pivotal in assessing the system's performance, particularly in terms of response time and functional accuracy.

Following data collection, we conducted a thorough analysis of the prototype's performance, focusing on the accuracy of function invocation and the efficiency of response times. We assessed accuracy 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, we closely analyzed errors 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. We examined patterns in the queries that led to incorrect function calls, using these findings to refine our approach to query formulation. This aspect of our analysis was instrumental in enhancing the system's capability to interpret and respond to user requests accurately, especially for complex queries requiring multiple functions.

Table 4. List of example queries categorized by the corresponding functions.

Function	Example Queries	
Change transparency	"Make all windows 30% transparent" "Set transparency of doors to 80%" "Apply 70% transparency to selection" "Set element 1234's transparency to 45%" "Make category 'Walls' 25% transparent"	"Adjust transparency of selection to 100%" "Set transparency of elements 5678 and 91,011 to 35%" "Change transparency for all elements in 'Furniture' category to 60%" "Make selected items fully opaque" "Apply 15% transparency to all elements in 'Lighting Fixtures'"
Isolate	"Isolate all doors in the current view" "Isolate windows on the selected floor" "Isolate walls with IDs [303, 404]" "Isolate selected furniture in the room" "Isolate floors in category 'carpet'"	"Isolate HVAC components in the selection" "Isolate lighting fixtures on level 2" "Isolate structural elements in view" "Isolate ceilings in the current floor" "Isolate plumbing elements in the bathroom"
Hide	"Hide all doors in the current view" "Hide windows on the selected floor" "Hide walls with IDs [707, 808]" "Hide selected furniture in the room" "Hide floors in category 'wood'"	"Hide HVAC components in the selection" "Hide lighting fixtures on level 3" "Hide structural elements in view" "Hide ceilings in the current floor" "Hide plumbing elements in the kitchen"
Set color	"Color all doors blue" "Set color of windows to green" "Color selected walls yellow" "Set color of wood floors to brown" "Change color of elements ID [1112, 1213] to orange"	"Color selected furniture purple" "Set all HVAC components to grey" "Color lighting fixtures black" "Set color of structural elements to white" "Color ceilings in the lobby pink"
Tag	"Tag doors as 'Fire-rated'" "Tag windows as 'Energy-efficient'" "Tag walls with 'Soundproof'" "Tag wooden floors as 'Oak'" "Tag furniture in the lounge as 'Leather'"	"Tag HVAC system as 'New Installation'" "Tag lighting in the hall as 'LED'" "Tag structural columns as 'Load-bearing'" "Tag ceilings with 'Acoustic Panel'" "Tag plumbing in restrooms as 'Water-saving'"
Delete element	"Delete selected doors in the layout" "Remove windows from the east wing" "Delete wall segments with IDs [1516, 1617]" "Remove selected chairs from the cafeteria" "Delete carpeted floors in the lobby"	"Erase HVAC units in the server room" "Remove chandeliers from the ballroom" "Delete load-bearing columns in the atrium" "Erase acoustic ceilings in the recording studio" "Delete plumbing pipes in the basement"
Selected element info	"Get info on selected doors for maintenance" "Show details of energy-efficient windows" "Retrieve info of soundproof walls" "Display data for oak wood floors" "Get info of leather furniture in the executive suite"	"Show details of the new HVAC installation" "Retrieve info of LED lighting in the corridor" "Display data for load-bearing columns" "Get info of acoustic ceilings in the auditorium" "Show details of water-saving plumbing"
Room info	"Update room info for office spaces" "Set room number for conference room ID [2122]" "Change room name for cafeteria ID [2223]" "Update occupant for office ID [2324]" "Set room data for executive suites"	"Update room details for storage areas" "Change room number for all rooms on level 4" "Set room names for meeting rooms" "Update occupant names for private cabins" "Set room data for all restrooms on ground floor"
Create view	"Create a 3D view named 'Landscape Design'" "Generate a floor plan for the third floor" "Create a ceiling plan for the main hall" "Generate a structural plan for the foundation" "Create a 3D view for the interior design"	"Generate a floor plan for the fourth floor" "Create a ceiling plan for the conference room" "Generate a structural plan for the new wing" "Create a 3D view for the entrance lobby" "Generate a floor plan for the rooftop terrace"

Table 4. *Cont.*

Function	Example Queries	
Duplicate view	"Duplicate the current 3D view as a dependent" "Create a detailed duplicate of the 'Main Lobby' view" "Duplicate the 'Level 1 Floor Plan' without detailing" "Create a dependent copy of the 'Roof Plan'" "Duplicate 'Basement Plan' with detailing included"	"Make a dependent duplicate of 'Ground Floor Plan'" "Duplicate 'East Wing Elevation' view with all details" "Create a simple duplicate of 'West Section'" "Duplicate 'Landscape Plan' as a dependent view" "Make a detailed duplicate of 'Electrical Plan Level 2'"
Rename	"Rename the '3D Building Model' view to 'Updated 3D Model'" "Change the name of 'Level 2 Floor Plan' to 'Second Floor Plan'" "Rename 'Roof Details' view to 'Updated Roof Plan'" "Change 'Basement Layout' view name to 'New Basement Plan'" "Rename 'Electrical Layout Ground Floor' to 'Ground Floor Electrical Plan'"	"Change 'Landscape Design' view name to 'Revised Landscape Layout'" "Change 'Plumbing Details Section A' to 'Section A Plumbing Updates'" "Rename 'HVAC Overview' to 'Updated HVAC Plan'" "Change 'Interior Design Plan' view name to 'Interior Decor Plan'" "Rename 'Main Entrance Elevation' to 'Front Elevation Updated'"
Undo	"Undo the last operation in Revit" "Revert the most recent change made" "Undo the last color change applied" "Roll back the last hide operation" "Undo the previous transparency setting"	"Revert the last tag applied to a category" "Undo the deletion of the last selected element" "Roll back the last update in room data" "Undo the creation of the last view" "Revert the last duplication of a view"

We also conducted manual tests to verify the results in the 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.

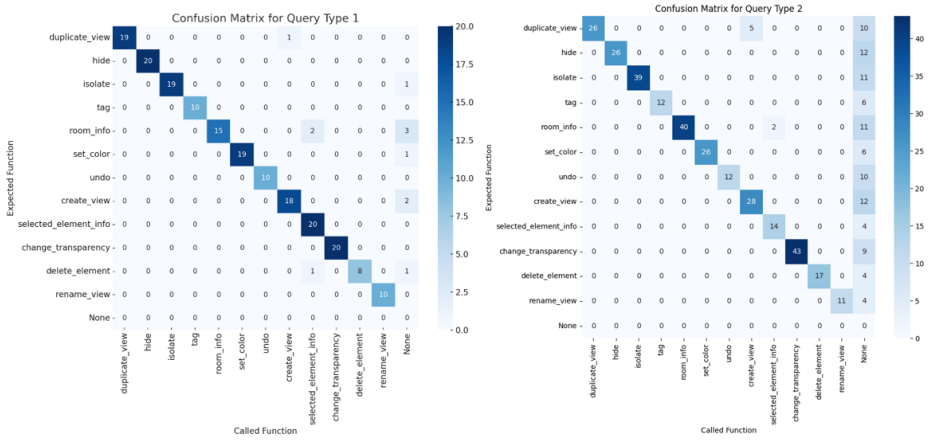
5.2. Validation Results

5.2.1. Automated Functionality Testing Results

The automated functionality testing phase subjected the 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 for token consumption efficiency.

The confusion matrices for Query Types 1 and 2 (Figure 15) reveal that while the system shows high accuracy in executing single-function queries (Figure 15a), there is a decline in the accuracy when processing compound queries (Figure 15b). This drop in performance is characterized by a higher frequency of failing to call a second required function in a single query.

Figure 16 presents a chart that shows the success and failure of functions called by Query Type and demonstrates the contrast in the success rates between the two query types. For Type 1, the success rate stood at 94% as the system leveraged 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 a feature [51]. 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.



(a) (b)

Figure 15. Confusion matrices for (**a**) Query Type 1 (single function calling queries) (left) and (**b**) Type 2 (multiple function calling queries) (right).

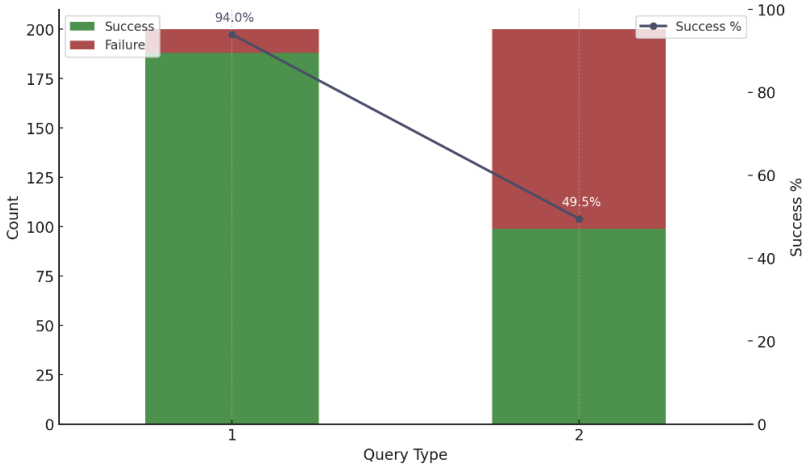


Figure 16. Success and failure of a query when considering whether the right function was called or not.

5.2.2. 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*. This indicates 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, we 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 our testing protocol only looked at the initial response by the GPT Assistant.

Furthermore, we 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, our 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 5 summarizes a range of errors encountered during our validation process, outlining their causes and proposing solutions to mitigate these issues.

Table 5. Error analysis: observations and causes of errors and proposed solutions.

Error Type	Observations	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

Table 5. *Cont.*

Error Type	Observations	Cause	Proposed Solution
Revit dialog box intervention	Occurs during operations leading to Revit warnings or errors requiring manual intervention	Certain actions (e.g., deleting a wall that defines a room) trigger Revit dialog boxes that DAVE cannot automatically close or resolve, necessitating manual user intervention	Develop a mechanism for DAVE to recognize potential actions that could trigger dialog boxes and either provide 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 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

5.2.3. Response Time Analysis

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

Our 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 17) 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 17), as well as for operations like *delete_element*. This 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.

On the other hand, 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.

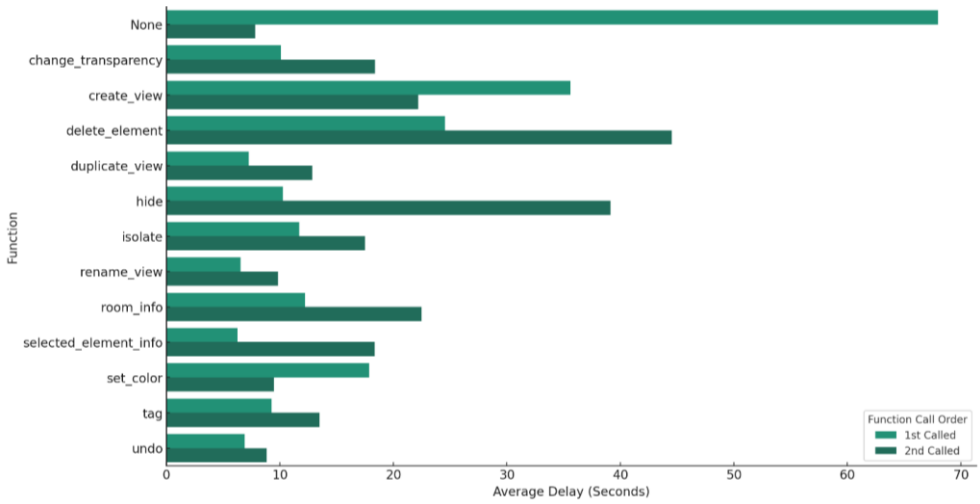


Figure 17. Average delay by function and call order.

In addition to the factors previously discussed, the utilization of voice mode introduced an additional variable to the system's response times. During testing, we observed that voice mode—either through receiving commands by voice or providing replies using voice—added an average delay of 3–5 s 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.

6. Discussion on Findings

6.1. 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, particularly NLP, to facilitate real-time updates and multimodal interactions (text, voice, and 3D visualization) with BIM models. 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. This makes DAVE a useful tool for architects, engineers, and other construction professionals and suitable for use in dynamic environments [35], such as VR [50]. This section delves into the strengths and implications of our system, highlighting its contributions to advancing conversational AI in AEC, enhancing BIM workflows, and promoting accessibility and inclusivity.

6.1.1. Advancement of Conversational AI in AEC

DAVE represents a novel advancement in the application of conversational AI in the AEC industry. 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 NLP (with text or voice commands). This positions DAVE as a 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, transforming traditional workflow approaches in architectural and engineering tasks. This also demonstrates the potential for conversational AI to transition from a supplementary tool to a central component in project management and execution.

6.1.2. Customization and Automation in the BIM Environment

The integration of AI with BIM through DAVE introduces a high level of customization and automation in the BIM environment and 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. The 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 has the potential to contribute to the overall effectiveness of BIM project management. This is a new paradigm in BIM model interaction, where users will no longer be constrained by the complexities of software interfaces or the intricacies of manual model updates.

6.1.3. Enhancement of the BIM Workflows

DAVE complements and extends the current state of research by its potential to help overcome some of the most common barriers to BIM adoption, such as BIM software complexity and the steep learning curves, as well as the complexity of BIM models [2–4]. By minimizing manual data entry and automating routine tasks, DAVE can help maintain the integrity of BIM models and reduce 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. This allows 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.

6.1.4. Accessibility and Inclusivity

DAVE's application of voice commands and natural language interaction to perform actions within BIMs represents a significant contribution to the field. While voice command technology itself is not new, DAVE is one of the systems that pioneer 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, which is another common challenge in BIM implementation [2–4]. 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.

6.2. Challenges and Limitations

Despite DAVE's innovative approach and 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.

6.2.1. Scalability, Data Integrity, Privacy, and Security

The scalability of the current version of our system is challenged by its architecture, which requires mapping each Revit action to a unique Python function. As BIM models become more complex, the volume of potential user actions would necessitate a substantial increase in the Python function library. This expansion demands ongoing development to keep pace with Revit updates and 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. This would also improve the system's flexibility in accommodating new Revit functionalities and user needs.

Given the reliance on OpenAI's API for DAVE's operation, our system inherits OpenAI's policies on privacy and security. OpenAI commits to high standards of data privacy and security, including the 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. To mitigate these risks, we propose enhancing 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 [52] 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.

6.2.2. Interoperability Challenges

DAVE's current compatibility is exclusively with Revit, and although Revit is one of the most commonly used BIM authoring tools, this can be considered 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. We will also work on expanding its compatibility to other BIM software to ensure that DAVE can serve as a versatile and comprehensive tool for BIM model management across the industry's diverse technological landscape.

6.2.3. Query Quality and Prompt Engineering

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 [46]. In our case, to obtain accurate and relevant responses in complex applications like BIM, it is essential that our model understands the user's intent and provides appropriate outputs. Since DAVE relies on NLP to interpret and execute commands within a BIM environment, the clarity and precision of user input or prompts directly impact the accuracy of the output.

DAVE is designed to handle both simple and complex queries. Prompt engineering enables the system to break down complex commands into small sub-tasks so that each component is correctly understood and executed. Since DAVE utilizes the GPT-4 model and operates on a token-based system, prompt engineering helps optimize the number of tokens used per query. Well-structured prompts can convey the necessary information concisely, minimizing the use of tokens. DAVE offers pre-defined templates for common BIM tasks, such as "changing transparency" or "selecting elements info", which provide a starting point for users to formulate their queries effectively.

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 color of the walls", a good quality query would specify the desired action in detail: "Hey DAVE, set the color of all walls on the second floor to red". In this example, the user specifies the action (set color), the target elements (all walls), the location (second floor), and the value (red). This way DAVE processes the prompt by understanding the context of "second floor" and "all walls", and identifies relevant elements in the BIM model correctly. It also helps in the execution of complex queries as the system translates the prompt into a series of actions within the BIM environment. As a result, it changes the color or other settings for the specified elements. 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 validation. Our tests revealed that 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 formulate 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.

6.2.4. System Efficiency and Economic Considerations

The efficiency of DAVE, particularly in complex or large-scale BIM projects, can be challenged by delays in information retrieval, which may extend up to 5 min as observed once during the validation. 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, these delays are planned to be addressed through the optimization of the system's architecture, such as enhancing data processing and retrieval methods or integrating more responsive AI models. Additionally, in [10], the researchers proposed an optimized dynamic framework for efficient data structuring and querying. Adopting such methodologies could enhance DAVE's ability to rapidly access information.

The operational costs associated with employing GPT-4, OpenAI's advanced language model, may present another challenge. The current general instructions and function details represent usage of around 3700 tokens per request (Figure 18), while the answer it

produces can generate as low as 97 tokens. For the current system's capabilities and current price, this represents a cost of approximately U$ 0.05 per request.

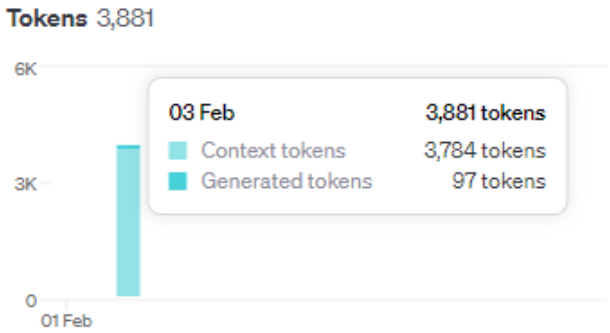


Figure 18. Approximate token consumption for a single request.

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.

7. Conclusions

In this study, we have explored the transformative potential of BIM with AI, focusing on the development of DAVE, a digital assistant that is designed to work with text or voice commands and aimed to enhance BIM model interaction and management within the AEC industry. By enabling intuitive, conversational interactions with BIMs, DAVE represents a significant contribution to the current body of knowledge and a leap toward making these technologies more accessible and efficient for professionals across the AEC industry.

By harnessing the capabilities of GPT-powered conversational AI, DAVE offers an intuitive and efficient means for users to engage with complex BIM environments in multiple modes (text or voice). This way, it aims to simplify the BIM interface for a wider audience and also exemplify the practical application of cutting-edge AI technologies in enhancing the operational efficiency and decision-making processes in construction projects. Our exploration into DAVE's development, functionality, and potential challenges is essential for future innovations in the AEC industry, encouraging a paradigm shift towards more accessible, intelligent, and user-centric model management solutions.

Future work includes extending DAVE's compatibility with other BIM software since the prototype architecture can be adapted to work with other BIM platforms and other LLMs with minor adjustments. Other future work includes refining query-processing capabilities and mitigating economic considerations associated with employing sophisticated AI models like GPT-4. Furthermore, 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.

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. Another 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.

Author Contributions: Conceptualization, D.F.; methodology, D.F.; software, D.F. and S.G.; validation, D.F. and S.G.; data curation, D.F.; visualization, D.F. and M.N.; writing—original draft, D.F. and M.N.; supervision, G.G.; project administration, G.G.; funding acquisition, G.G.; writing—review and editing, S.G. and G.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the financial support provided by the Price Faculty of Engineering, University of Manitoba; University of Manitoba Undergraduate Research Awards (URA); the Natural Sciences, Engineering Research Council of Canada (NSERC) Undergraduate Student Research Awards (USRA) and the University of Manitoba Graduate Fellowship (UMGF). The APC was funded by the University of Manitoba.

Data Availability Statement: The original contributions presented in the study are included in the article, and further inquiries can be directed to the corresponding author.

Acknowledgments: The authors gratefully appreciate the financial support provided by the Price Faculty of Engineering, University of Manitoba and NSERC.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Sacks, R.; Eastman, C.; Lee, G.; Teicholz, P. *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*, 3rd ed.; Wiley: Hoboken, NJ, USA, 2018.
2. Ullah, K.; Lill, I.; Witt, E. An Overview of BIM Adoption in the Construction Industry: Benefits and Barriers. In Proceedings of the 10th Nordic Conference on Construction Economics and Organization, Tallinn, Estonia, 7–8 May 2019; Emerald Publishing Limited: Leeds, UK, 2019; ISBN 978-1-83867-051-1.
3. Cao, Y.; Zhang, L.H.; McCabe, B.; Shahi, A. The Benefits of and Barriers to BIM Adoption in Canada. In Proceedings of the International Symposium on Automation and Robotics in Construction, ISARC, Banff, AB, Canada, 21–24 May 2019; Volume 36, pp. 152–158. [CrossRef]
4. Zahedi, F.; Sardroud, J.M.; Kazemi, S. Global BIM adoption movements and challenges: An Extensive literature Review. In Proceedings of the Creative Construction e-Conference 2022, Faculty of Architecture, Budapest University of Technology and Economics, Budapest, Hungary, 9–11 July 2022. [CrossRef]
5. Abioye, S.O.; Oyedele, L.O.; Akanbi, L.; Ajayi, A.; Delgado, J.M.D.; Bilal, M.; Akinade, O.O.; Ahmed, A. Artificial intelligence in the construction industry: A review of present status, opportunities and future challenges. *J. Build. Eng.* **2021**, *44*, 103299. [CrossRef]
6. Zhang, F.; Chan, A.P.C.; Darko, A.; Chen, Z.; Li, D. Integrated applications of building information modeling and artificial intelligence techniques in the AEC/FM industry. *Autom. Constr.* **2022**, *139*, 104289. [CrossRef]
7. Pan, Y.; Zhang, L. Roles of artificial intelligence in construction engineering and management: A critical review and future trends. *Autom. Constr.* **2021**, *122*, 10351. [CrossRef]
8. Rampini, L.; Cecconi, F.R. Artificial Intelligence in construction asset management: A review of present status, challenges and future opportunities. *J. Inf. Technol. Constr.* **2022**, *27*, 884–913. [CrossRef]
9. Saka, A.; Taiwo, R.; Saka, N.; Salami, B.A.; Ajayi, S.; Akande, K.; Kazemi, H. GPT models in construction industry: Opportunities, limitations, and a use case validation. *Dev. Built Environ.* **2024**, *17*, 100300. [CrossRef]
10. Zheng, J.; Fischer, M. Dynamic prompt-based virtual assistant framework for BIM information search. *Autom. Constr.* **2023**, *155*, 105067. [CrossRef]
11. Elghaish, F.; Chauhan, J.K.; Matarneh, S.; Rahimian, F.P.; Hosseini, M.R. Artificial intelligence-based voice assistant for BIM data management. *Autom. Constr.* **2022**, *140*, 104320. [CrossRef]
12. OpenAI, GPT-4. 2024. Available online: https://openai.com/gpt-4 (accessed on 24 February 2024).
13. Google, Gemini Apps FAQ. 2024. Available online: https://gemini.google.com/faq?gad_source=1&gclid=Cj0KCQiA5-uuBhDzARIsAAa21T-MuRuzmbLRTPE3tBPlrgkJ3zhzawColcc_bIFCdC6ZYMxtYKR3szUaAujUEALw_wcB (accessed on 24 February 2024).
14. Chang, Y.; Wang, X.; Wang, J.; Wu, Y.; Yang, L.; Zhu, K.; Chen, H.; Yi, X.; Wang, C.; Wang, Y.; et al. A Survey on Evaluation of Large Language Models. *ACM Trans. Intell. Syst. Technol.* **2024**, *15*, 1–45. [CrossRef]
15. Lin, W.Y. Prototyping a Chatbot for Site Managers Using Building Information Modeling (BIM) and Natural Language Understanding (NLU) Techniques. *Sensors* **2023**, *23*, 2942. [CrossRef]

16. Shahinmoghadam, M.; Kahou, S.E.; Motamedi, A. Neural semantic tagging for natural language-based search in building information models: Implications for practice. *Comput. Ind.* **2023**, *155*, 104063. [CrossRef]
17. Prieto, S.A.; Mengiste, E.T.; de Soto, B.G. Investigating the Use of ChatGPT for the Scheduling of Construction Projects. *Buildings* **2023**, *13*, 857. [CrossRef]
18. Singh, A.K.; Pal, A.; Kumar, P.; Lin, J.J.; Hsieh, S.-H. Prospects of Integrating BIM and NLP for Automatic Construction Schedule Management. In Proceedings of the 40th ISARC, Chennai, India, 5–7 July 2023; pp. 238–245. [CrossRef]
19. Zhou, Y.; She, J.; Huang, Y.; Li, L.; Zhang, L.; Zhang, J. A Design for Safety (DFS) Semantic Framework Development Based on Natural Language Processing (NLP) for Automated Compliance Checking Using BIM: The Case of China. *Buildings* **2022**, *12*, 780. [CrossRef]
20. Chen, N.; Lin, X.; Jiang, H.; An, Y. Automated Building Information Modeling Compliance Check through a Large Language Model Combined with Deep Learning and Ontology. *Buildings* **2024**, *14*, 1983. [CrossRef]
21. Shen, Q.; Wu, S.; Deng, Y.; Deng, H.; Cheng, J.C.P. BIM-Based Dynamic Construction Safety Rule Checking Using Ontology and Natural Language Processing. *Buildings* **2022**, *12*, 564. [CrossRef]
22. Jang, S.; Lee, G.; Oh, J.; Lee, J.; Koo, B. Automated detailing of exterior walls using NADIA: Natural-language-based architectural detailing through interaction with AI. *Adv. Eng. Inform.* **2024**, *61*, 102532. [CrossRef]
23. Jang, S.; Lee, G. Interactive Design by Integrating a Large Pre-Trained Language Model and Building Information Modeling. In *Computing in Civil Engineering 2023*; American Society of Civil Engineers: Corvallis, OR, USA, 25–28 June 2023; pp. 291–299. [CrossRef]
24. Zabin, A.; González, V.A.; Zou, Y.; Amor, R. Applications of machine learning to BIM: A systematic literature review. *Adv. Eng. Inform.* **2022**, *51*, 101474. [CrossRef]
25. Xu, S.; Wang, J.; Shou, W.; Ngo, T.; Sadick, A.M.; Wang, X. Computer Vision Techniques in Construction: A Critical Review. *Arch. Comput. Methods Eng.* **2021**, *28*, 3383–3397. [CrossRef]
26. Ahmadpanah, H.; Haidar, A.; Latifi, S.M. BIM and Machine Learning (ML) Integration in Design Coordination Using ML to automate object classification for clash detection. In Proceedings of the 41st Education and Research in Computer Aided Architectural Design in Europe (eCAADe) Conference (eCAADe 2023), Graz, Austria, 20–22 September 2023; pp. 619–628.
27. Wang, W.; Guo, H.; Li, X.; Tang, S.; Xia, J.; Lv, Z. Deep learning for assessment of environmental satisfaction using BIM big data in energy efficient building digital twins. *Sustain. Energy Technol. Assess.* **2022**, *50*, 101897. [CrossRef]
28. Stephans, T.; McClymonds, A.; Leicht, R.; Wagner, A.R. Automated material selection based on detected construction progress. In Proceedings of the 2022 39th ISARC, Bogotá, Colombia, 13–15 July 2022. [CrossRef]
29. Singh, J.; Anumba, C.J. Real-Time Pipe System Installation Schedule Generation and Optimization Using Artificial Intelligence and Heuristic Techniques. *J. Inf. Technol. Constr.* **2022**, *27*, 173–190. [CrossRef]
30. Orooje, M.S.; Latifi, M.M. A Review of Embedding Artificial Intelligence in Internet of Things and Building Information Modelling for Healthcare Facility Maintenance Management. *Energy Environ. Res.* **2021**, *11*, 31. [CrossRef]
31. Sampaio, R.P.; Costa, A.A.; Flores-Colen, I. A Systematic Review of Artificial Intelligence Applied to Facility Management in the Building Information Modeling Context and Future Research Directions. *Buildings* **2022**, *12*, 1939. [CrossRef]
32. Darko, A.; Chan, A.P.C.; Adabre, M.A.; Edwards, D.J.; Hosseini, M.R.; Ameyaw, E.E. Artificial intelligence in the AEC industry: Scientometric analysis and visualization of research activities. *Autom. Constr.* **2020**, *112*, 103081. [CrossRef]
33. Ding, Y.; Ma, J.; Luo, X. Applications of natural language processing in construction. *Autom. Constr.* **2022**, *136*, 104169. [CrossRef]
34. Erfani, A.; Cui, Q. Natural Language Processing Application in Construction Domain: An Integrative Review and Algorithms Comparison. In *Computing in Civil Engineering 2021*; American Society of Civil Engineers (ASCE): Orlando, FL, USA, 2021; pp. 26–33. [CrossRef]
35. Shin, S.; Issa, R.R.A. BIMASR: Framework for Voice-Based BIM Information Retrieval. *J. Constr. Eng. Manag.* **2021**, *147*, 10. [CrossRef]
36. Nabavi, A.; Ramaji, I.; Sadeghi, N.; Anderson, A. Leveraging Natural Language Processing for Automated Information Inquiry from Building Information Models. *J. Inf. Technol. Constr.* **2023**, *28*, 266–285. [CrossRef]
37. Wang, N.; Issa, R.R.A.; Anumba, C.J. NLP-Based Query-Answering System for Information Extraction from Building Information Models. *J. Comput. Civ. Eng.* **2022**, *36*, 04022004. [CrossRef]
38. Wang, N.; Issa, R.R.A.; Anumba, C.J. Transfer learning-based query classification for intelligent building information spoken dialogue. *Autom. Constr.* **2022**, *141*, 104403. [CrossRef]
39. Lin, T.H.; Huang, Y.H.; Putranto, A. Intelligent question and answer system for building information modeling and artificial intelligence of things based on the bidirectional encoder representations from transformers model. *Autom. Constr.* **2022**, *142*, 104483. [CrossRef]
40. Xue, X.; Hou, Y.; Zhang, J. Automated Construction Contract Summarization Using Natural Language Processing and Deep Learning. In Proceedings of the 39th ISARC, Bogotá, Colombia, 13–15 July 2022. [CrossRef]
41. Wu, J.; Xue, X.; Zhang, J. Invariant Signature, Logic Reasoning, and Semantic Natural Language Processing (NLP)-Based Automated Building Code Compliance Checking (I-SNACC) Framework. *J. Inf. Technol. Constr.* **2023**, *28*, 1–18. [CrossRef]
42. Hussain, R.; Sabir, A.; Lee, D.Y.; Zaidi, S.F.A.; Pedro, A.; Abbas, M.S.; Park, C. Conversational AI-based VR system to improve construction safety training of migrant workers. *Autom. Constr.* **2024**, *160*, 105315. [CrossRef]

43. Locatelli, M.; Seghezzi, E.; Pellegrini, L.; Tagliabue, L.C.; Di Giuda, G.M. Exploring natural language processing in construction and integration with building information modeling: A scientometric analysis. *Buildings* **2021**, *11*, 583. [CrossRef]
44. Saka, A.B.; Oyedele, L.O.; Akanbi, L.A.; Ganiyu, S.A.; Chan, D.W.M.; Bello, S.A. Conversational artificial intelligence in the AEC industry: A review of present status, challenges and opportunities. *Adv. Eng. Inform.* **2023**, *55*, 101869. [CrossRef]
45. Chung, S.; Moon, S.; Kim, J.; Kim, J.; Lim, S.; Chi, S. Comparing natural language processing (NLP) applications in construction and computer science using preferred reporting items for systematic reviews (PRISMA). *Autom. Constr.* **2023**, *154*, 105020. [CrossRef]
46. OpenAI, Prompt Engineering. 2024. Available online: https://platform.openai.com/docs/guides/prompt-engineering (accessed on 7 February 2024).
47. Rane, N.; Choudhary, S.; Rane, J. 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. Available online: https://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID4658066_code6202721.pdf?abstractid=4645601&mirid=1&type=2 (accessed on 8 August 2024). [CrossRef]
48. Uddin, S.M.J.; Albert, A.; Ovid, A.; Alsharef, A. Leveraging ChatGPT to Aid Construction Hazard Recognition and Support Safety Education and Training. *Sustainability* **2023**, *15*, 7121. [CrossRef]
49. Ghimire, P.; Kim, K.; Acharya, M. Opportunities and Challenges of Generative AI in Construction Industry: Focusing on Adoption of Text-Based Models. *Buildings* **2024**, *14*, 220. [CrossRef]
50. Fernandes, D.; Nikkel, M.; Guven, G. BIM-AI-VR Integration for Real-Time Model Update and Visualization. In Proceedings of the Canadian Society for Civil Engineering (CSCE) Annual Conference, Niagara Falls, ON, Canada, 5–7 June 2024.
51. OpenAI, Assistants Tools. 2024. Available online: https://platform.openai.com/docs/assistants/tools (accessed on 7 February 2024).
52. Autodesk, Authentication (OAuth) Scopes. 2023. Available online: https://aps.autodesk.com/en/docs/oauth/v2/developers_guide/scopes/ (accessed on 1 March 2024).