A Virtual Assistant for Architectural Design in a VR Environment

Ander Salaberria[1], Oier Ijurco[1], Markel Ferro[1], Jiayuan Wang[1], Iñigo Vilá Muñoz[1], Roberto de Ioris[2], Jeremy Barnes[1], Oier Lopez de Lacalle[1]

[1]HiTZ Center, University of the Basque Country (UPV/EHU), [2]Vection Technologies
Correspondence: ander.salaberria@ehu.eus

Abstract

Architectural design relies on 3D modeling procedures, generally carried out in Building Information Modeling (BIM) formats. In this setting, architects and designers collaborate on building designs, iterating over many possible versions until a final design is agreed upon. However, this iteration is complicated by the fact that any changes need to be made by manually introducing changes to the complex BIM files, which lengthens the design process and makes it difficult to quickly prototype changes.

To speed up prototyping, we propose VR-ARCH, a virtual assistant that allows users to interact with the BIM file in a virtual reality (VR) environment. This framework enables users to 1) make changes directly in the VR environment, 2) make complex queries about the BIM, and 3) combine these to perform more complex actions. All of this is done via voice commands and processed through a ReAct-based agentic system that selects appropriate tools depending on the query context.

This multi-tool approach enables real-time, contextualized interaction through natural language, allowing for a faster and more natural prototyping experience.

Our demo's video is available at this link, whereas the code and data are publicly available here.

1 Introduction

Architectural design review has traditionally relied on manual inspection of 2D drawings, a process that is often time-consuming and prone to oversight. The adoption of Building Information Modeling (BIM) has significantly improved this workflow by providing integrated 3D representations of buildings, leading to better coordination, fewer disputes, and increased stakeholder satisfaction (Leite, 2019; Fischer, 2006). BIM models contain rich, networked information that supports informed decision-making across architectural, structural, and engineering teams (Sacks et al., 2019).

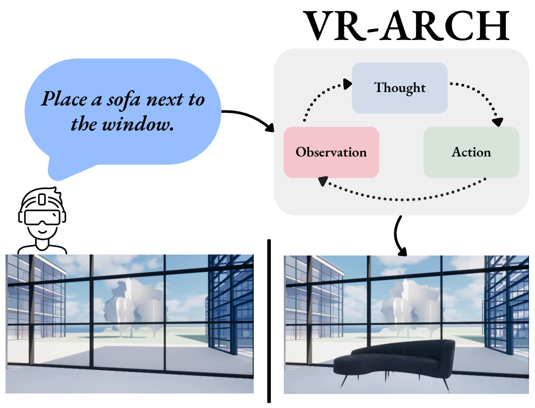


Figure 1: VR-ARCH interacts with the building and retrieves information from its BIM to assist the user in a VR environment.

Despite these benefits, interacting with BIM environments still requires specialized knowledge and proficiency with complex software, which can hinder real-time collaboration, particularly for non-expert users (Dong et al., 2025; Leite, 2019). Existing interfaces often lack the intuitive, context-aware capabilities necessary for easily navigating or adjusting intricate architectural components in 3D spaces (Bagasi et al., 2025). In this context, natural language interaction emerges as a promising solution: an LLM-based assistant could enable users to explore, query, and modify BIM models directly through natural language, simplifying design review and enhancing collaboration.

The integration of natural language capabilities into Virtual Reality (VR) systems has long been an active area of research (Everett et al., 1997, 1999; Giunchi et al., 2024). With recent advances in LLMs (Radford and Narasimhan, 2018; Brown et al., 2020), capable of sophisticated reasoning and language understanding, there is growing interest in deploying these models within immersive environments, such as collaborative design assistants (Wu et al., 2023) or as interactive virtual tutors (Ward et al., 2025).

We propose **VR-ARCH**, an LLM-based assistant that enables users to interact with and modify BIM environments through natural voice commands in a dynamic and incremental manner. Unlike previous works, VR-ARCH leverages spatial reasoning and the user's position to interpret commands more accurately, such as referencing nearby elements or adjusting components relative to the user's viewpoint. Through an iterative, tool-based agent architecture, the assistant can retrieve building information, execute multi-step operations, and continuously refine the environment. Figure 1 shows a user request for VR-ARCH to place a sofa next to a window in the BIM environment.

With the goal of allowing the LLM assistant to interact directly with the VR environment and BIM data, we develop a custom Unreal Engine sandbox combined with a Python API. This API provides functions to interact with assets within the building, such as hiding/revealing doors, changing a wall's color, or rotating stairs. Similarly, to enable queries, we use a Neo4j graph database that stores BIM data and allows complex queries. To combine these two abilities, we use a ReAct (Yao et al., 2022) inspired agentic approach where a Router LLM iteratively chooses from three specialized tools: Modifier, Querier and ID Retriever. This approach can then use these subroutines to fulfill more complex user queries. Finally, we use a speech-to-text system to capture voice commands and a text-to-speech module to return spoken responses, enabling hands-free interaction with the system.

In order to approximate the usefulness of our system in for a real-world user, we perform human evaluation and furthermore develop an automatic evaluation via LLMs-as-a-judge. This evaluation finds that models are capable of accurately answering user questions and executing successful modifications within a BIM environment, especially as model size increases.

2 Related Work

NLP approaches in VR are often dedicated to enabling VR tutors (García-Méndez et al., 2024; Konenkov et al., 2024). Ward et al. (2025), for example, develop a VR tutor for learning Irish, while Aguirre et al. (2025) develop a VR tutor for VR health and safety training.

Another common application of NLP in VR is the development of VR assistants. Wu et al. (2023) introduce a video-grounded task-oriented dialogue dataset that captures real-world AI-assisted user scenarios in VR, while Prange et al. (2017) develop a multimodal dialogue system to help doctors make decisions about patient therapy.

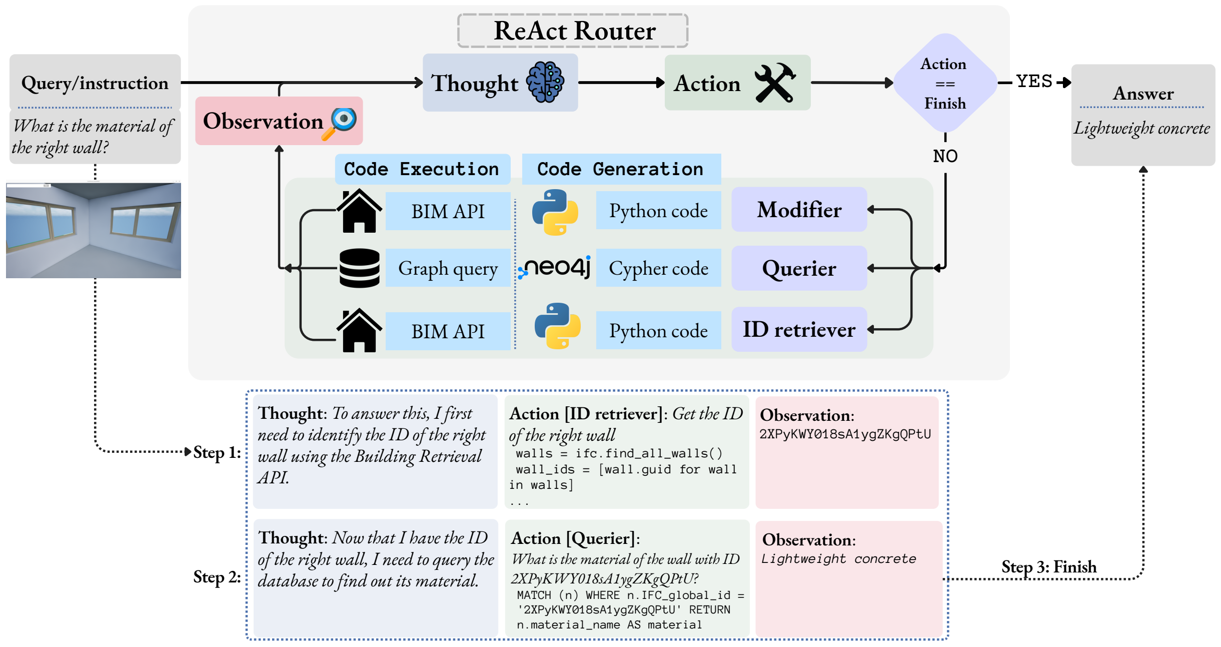
Finally, there are also a few efforts to develop avatar-based VR chatbots, either via spoken dialogue (Yamazaki et al., 2023) or sign language (Quandt et al., 2022).

While these VR tutors and assistants represent an interesting step in integrating natural language in VR, they are not actually able to make any changes to the VR environment itself and generally deal with hard-coded queries, rather than performing them on the fly.

BIM Manipulation with LLM Integration. Most previous work on manipulating BIM files with LLMs has focused on querying the BIM for information (Zheng and Fischer, 2023; Li et al., 2025). Recent work has also explored ways of generating 3D building models from natural language (Du et al., 2024). Editing or prototyping changes to the BIM file, however, has not been widely explored. While Jang and Lee (2024) propose a pipeline that converts BIM to XML to facilitate LLM changes and Fernandes et al. (2024) explores conversational manipulation of building information, these approaches do not allow a user to make changes that are directly visible in a VR environment, grounded by the user's spatial relationships. Furthermore, they currently struggle with composite queries, where several changes must be implemented.

Knowledge Graphs for BIM. Knowledge graphs have emerged as a structured approach to representing BIM data, enabling complex querying capabilities. Some prior work demonstrate the use of graph databases, particularly Neo4j and the Cypher query language, to represent and query building information (Ozsoy et al., 2025; Zhu et al., 2024). As these approaches allow for complex relational queries across elements in the building, we make use of a similar approach in VR-ARCH.

Instruction-Following Multimodal Agents. Recent advances have demonstrated that modern agents can effectively follow instructions in complex multimodal scenarios. A prevalent paradigm in recent years involves leveraging external APIs to



Thought: *To answer this, I first need to identify the ID of the right wall using the Building Retrieval API.*

Action [ID retriever]: *Get the ID of the right wall*

```
 walls = ifc.find_all_walls()
 wall_ids = [wall.guid for wall
in walls]
...
```

Observation:
`2XPyKWY018sA1ygZKgQPtU`

Thought: *Now that I have the ID of the right wall, I need to query the database to find out its material.*

Action [Querier]:
What is the material of the wall with ID 2XPyKWY018sA1ygZKgQPtU?

```
 MATCH (n) WHERE n.IFC_global_id =
 '2XPyKWY018sA1ygZKgQPtU' RETURN
 n.material_name AS material
```

Observation:
`Lightweight concrete`

Figure 2: VR-ARCH pipeline. Given the scene's configuration and an instruction, the ReAct router will decide which tool to use in order to modify, query or retrieve information from the building. The code for the chosen tool is then created and executed, and the ReAct Router will decide what to do in the next iteration until the initial query is fulfilled. Finally, it will generate a response to the initial user query.

extend the agent's action space (Shen et al., 2023). These APIs are often integrated with external models to handle more sophisticated tasks (Yang et al., 2023). Our work is closely related to systems such as ViperGPT (Surís et al., 2023) and VisProg (Gupta and Kembhavi, 2023), which frame visual question answering as a complex process requiring perception, reasoning and structured tool use. These systems typically utilize LLMs to generate executable procedures that coordinate multiple components. Our approach employs a ReAct-style routing mechanism (Yao et al., 2022) that selects among a variety of tools.

3 System Architecture

VR-ARCH contains **six main components** that together form the pipeline shown in Figure 2. The scene is rendered in a custom Unreal Engine sandbox that loads a BIM file. This sandbox communicates with a Python API that is able to apply changes to the sandbox via predefined functions. These functions allow us to identify objects, interact with them (change color, move, hide, show, etc.), handle the camera, and add/transform/delete props. Additionally, the BIM data is stored in a Neo4j graph database, enabling complex queries about the building and object characteristics.

The system takes a voice command from the user, which is converted via a speech-to-text module. The resulting text is fed to the ReAct Router, which orchestrates an iterative tool usage process to fulfill the user's request. At each step, the ReAct Router selects the most appropriate tool based on previous steps and current state of the process. This continues until the request is satisfied or a maximum number of steps is reached. The selected tool executes its task, and the results are taken as observation for the next iteration of the system. Finally, the textual output is vocalized with a text-to-speech module, providing hands-free interaction in the entire pipeline.

The ReAct Router can make use of three distinct tools, each designed for specific aspects of BIM interaction:

Modifier Tool. This tool creates the necessary Python code to apply direct changes to the sandbox environment. Spatial queries are supported, as it also takes the user's position within the scene into account. The generated code uses the Python API to perform operations such as modifying objects, manipulating the camera or creating props. This tool is selected when the user's intent involves making visible changes in the 3D environment.

Querier Tool. This tool generates Cypher queries in order to get general information about the current building in the Neo4j database. It is also used when the user needs relational information about BIM elements, such as querying specific object properties or relationships between elements. The Cypher queries enable complex graph-based searches to be made, which would be impossible to perform through the Python API alone.

ID Retriever Tool. This tool serves as a connection between spatial operations and database queries. It generates Python code using the API in order to retrieve specific object IDs based on spatial criteria (e.g., objects in front of the user, the furthest objects from the user). These retrieved IDs can then be used by the Querier tool in the next iteration to obtained detailed information about said object via the Neo4j database. This two-step approach enables requests that combine both spatial awareness and information retrieval.

3.1 Adaptation of the LLM

We employ prompt engineering techniques to adapt each LLM to its specific task-oriented environment, as the ReAct Router and each tool utility serve different purposes within our system (see Appendix B). The Router determines which tool to use at each iteration based on the user query and current state of observations, while each tool is specialized for its particular function: the Modifier generates Python code for sandbox manipulation, the Querier creates Cypher queries for database calls, and the ID Retriever generates Python code for spatial object identification.

Each tool's prompt is designed with specific requirements and includes relevant information such as task definitions, API documentation, or Graph schemas, where needed, and few-shot examples demonstrating correct usage of tools and system. For code generation tools (Modifier and ID Retriever), prompts include the Python API documentation and examples of how to create correct executable code. The Querier tool's prompt includes the Neo4j schema and Cypher syntax examples in order to retrieve general information. The ReAct Router's prompt focuses on tool selection depending on the given request and examples of appropriate tool usage in different scenarios. This prompting strategy makes sure that each component operates according to its role while maintaining coherence across the iterative process.

4 Demonstrator evaluation

The following link contains a video showcasing a short summary of the VR-Arch system with a couple of running examples: `https://youtu.be/XyxrOU3CWHs`. To assess the capabilities of the proposed system, we run an extensive evaluation of the system's performance. This section is dedicated to this evaluation and its analysis.

4.1 Evaluation Settings

The evaluation of VR-Arch focuses on assessing the system's ability to accurately interpret and execute natural language commands through a multi-tool ReAct framework. The evaluation was designed to measure both the technical accuracy of the generated code and queries, as well as the steps taken by the system itself.

Models. The ReAct Router, Modifier, and ID Retriever components all use Qwen3 language models (Yang et al., 2025). We evaluate four variants (4B, 8B, 14B and 32B parameter versions) to explore the effect of model scale, as higher capacity usually increases model performance at the expense of execution-time and higher infrastructure needs. For the Querier component, we use a 9B parameter Gemma2 model (Team et al., 2024) fine-tuned to create Cypher queries (Ozsoy et al., 2025).

Dataset. The evaluation dataset consists of 120 manually-annotated instances, divided into two categories to test the agentic system's different functionalities. On the one hand, the *Modify* category includes instances that require the model to change visibility, coloring, transformation, and removal of BIM entities, as well as camera transformations and prop addition. On the other hand, the *Query* category comprises another 60 instances that require retrieving building information without making any modifications, such as counting windows. Half of them are general questions that can be answered with just the Querier Tool, while the other half requires spatial reasoning to identify the corresponding objects using the user's spatial information before making the query.

These instances are evenly divided between two different BIM environments of varying complexity: a house and a school. The former is a simple building containing around 200 entities, whereas the latter is composed of almost 6,000, allowing us to measure how the BIM can affect the completion of instructions. The rendered scenes can be seen

Model	Modify	Query		Total
		General	Spatial	
Qwen3-4B	45.3	50.0	50.0	47.7
Qwen3-8B	46.7	50.0	**70.0**	53.3
Qwen3-14B	38.3	50.0	50.0	44.1
Qwen3-32B	**60.0**	**60.0**	67.0	**61.7**

Table 1: Human evaluation of instances that were correctly completed, divided into three main categories: Modify and Query-general and Query-spatial, as well as the total accuracy of each model.

in Appendix A. We also divide instances into easy (66) and hard (54) instances to measure the effects of query difficulty on model performance. Easy instances are short and simple (e.g "Hide the left door."), whereas the hard ones are composite or require more complex reasoning paths (e.g. "Look backwards and hide the wall found there.").

Evaluation metrics. Due to the modular nature of VR-ARCH and the difficulty of evaluating new BIM files by hand, we further explore automatic evaluation via a judge LLM (Prometheus 2 (Kim et al., 2024)) which was adapted to compare with manually created gold code responses. In our *Modify* instances, we compare the generated code with the ground truth code. For *Query* instances, we compare the ground truth answer with the generated response. In order to validate whether this methodology is a viable surrogate for lengthy human validation, we also measure its correlation w.r.t. human evaluation.

4.2 Main Results

Table 1 shows the results of the evaluated models and reveals a clear trend: accuracy improves with model size. Notably, our largest model achieves a 14-point increase in accuracy compared to the smallest one, reaching up to 61.7 accuracy. Generally, the Modify category presents the greatest challenge for the models due to the necessity of generating correct, executable Python code. The Query category, in contrast, is answered correctly a greater number of times. Note that the Querier tool is the same for all experiments, so any perceived improvement is due to better reasoning of the router. Surprisingly, Qwen3-14B is the worst-performing model despite having more parameters than the 4B and 8B versions. We have noticed this tendency across all our experiments, but further experimentation is needed to shed some light on this behavior.

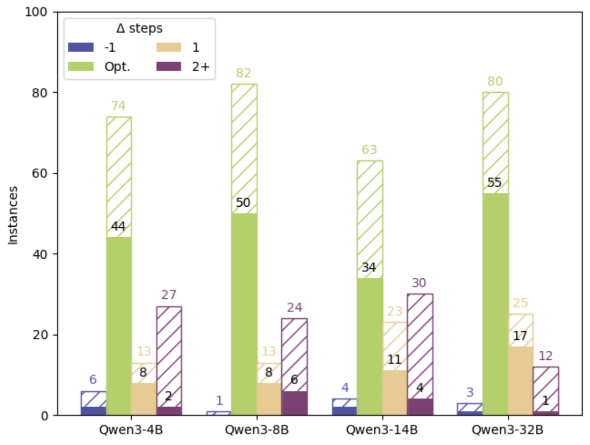


Figure 3: We measure the proportion of instances the agent answers correctly as a function of Δ steps, where Δ represents the additional steps taken beyond the optimal solution. The dashed bars indicate how many instances are solved within each Δ step, while the filled areas show the proportions that are answered correctly.

Model	Query Difficulty		BIM Environment	
	Easy	Hard	House	School
Qwen3-4B	52.5	42.9	43.2	52.8
Qwen3-8B	63.0	46.3	50.4	59.5
Qwen3-14B	49.4	41.3	43.5	47.5
Qwen3-32B	**72.3**	**51.6**	**63.3**	**60.4**

Table 2: Human evaluation of model performance by query difficulty (easy, hard) and increasing BIM complexity (House, School).

Category	Human Eval.	Prometheus Eval.
Modify	48.3	60.0
Query - General	56.7	60.0
Query - Spatial	60.0	66.7
Total	53.3	61.7

Table 3: Comparison of human evaluation and Prometheus Eval results (percentage of correct instances).

Table 2 presents the evaluation results depending on the difficulty of the instances. As expected, the model performance degrades when challenged with hard questions compared to easy ones, resulting in a 20-point drop in the 32B parameter model. Smaller models also show a similar pattern, with an average drop of about 11 points across them.

All models perform similarly on the House and School environments, despite the increased complexity. This suggests that our system is robust to

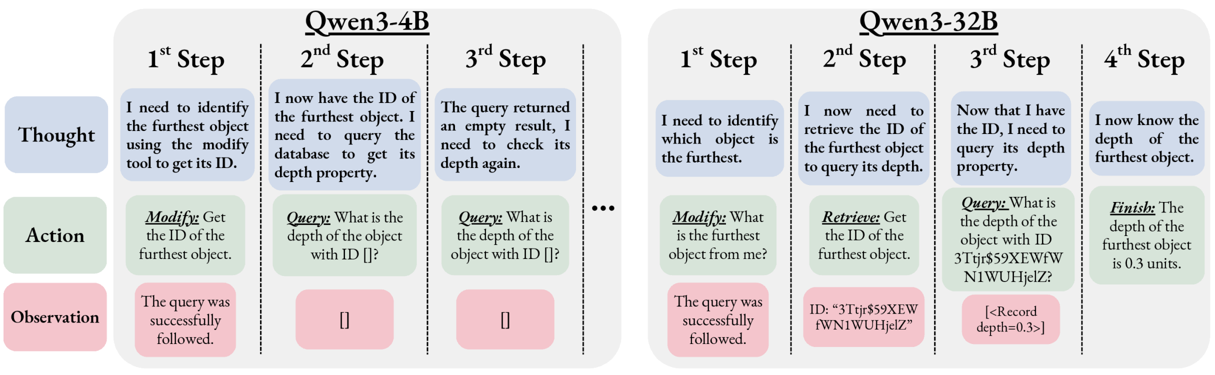

Qwen3-4B
1st Step
2nd Step
3rd Step
Thought
Action
Observation
I need to identify the furthest object using the modify tool to get its ID.
Modify: Get the ID of the furthest object.
The query was successfully followed.
I now have the ID of the furthest object. I need to query the database to get its depth property.
Query: What is the depth of the object with ID []?
[]
The query returned an empty result, I need to check its depth again.
Query: What is the depth of the object with ID []?
[]
...
Qwen3-32B
1st Step
2nd Step
3rd Step
4th Step
I need to identify which object is the furthest.
Modify: What is the furthest object from me?
The query was successfully followed.
I now need to retrieve the ID of the furthest object to query its depth.
Retrieve: Get the ID of the furthest object.
ID: "3Ttjr$59XEWfWN1WUHjelZ"
Now that I have the ID, I need to query its depth property.
Query: What is the depth of the object with ID 3Ttjr$59XEWfWN1WUHjelZ?
[<Record depth=0.3>]
I now know the depth of the furthest object.
Finish: The depth of the furthest object is 0.3 units.

Figure 4: Agent trajectories for the 4B and 32B models given the query: "Tell me the depth of the furthest object."

variations in BIM size, including a larger number of objects and relationships, and that scaling up the environment will not necessarily lead to a loss in performance.

4.3 Human Agreement

In order to determine the reliability of our data, two human annotators evaluated all outputs from the Qwen3-32B model across three task categories. Overall, considering the full evaluation set of 120 instances, the annotators produced different annotations in only 10 cases. A third annotator broke the tie for these last cases (Cohen's $\kappa = 0.83$). Table 3 shows that both human and automatic evaluations are well correlated category-wise. Moreover, humans are stricter in the evaluation, as the average accuracy drops 8 points in the case of Qwen3-32B. A more fine-grained comparison shows that both evaluations agree on 66.7% of instances. This suggests that for new BIM environments, an LLM judge serves as a slightly optimistic evaluation.

4.4 Analysis

The effects of efficient routing. Efficient routing is critical for enhancing system responsiveness and reducing user-perceived latency, as it limits the number of inferences performed. Figure 3 shows the correlation between efficient routing and model accuracy. We observe that in most of the correct responses, the model uses the optimal number of steps, without any unnecessary actions. Conversely, while a Δ of 1 step can still yield a correct response, the success rate declines sharply with additional steps, indicating that the model is capable of correcting itself, but extending the execution sequence too much is rarely a successful recovery strategy. This insight justifies future work to explore an adaptive maximum step limit, similar to the mechanism explored by Snell et al. (2024).

We observed a few cases where VR-ARCH finishes one step before the optimal number. This mainly happens in queries where the model guesses an object that, despite being unrelated to the query, provides the correct answer for the requested property. This behavior is uncommon and decreases as the model size increases.

Reasoning examples. Figure 4 demonstrates the relationship between model size and quality of the trajectories generated by 4B and 32B models in response to the query *"Tell me the depth of the furthest object"*. The 4B model fails to reason correctly, attempting to use the modify tool incorrectly and repeating calls to the query tool without resolving the task. In contrast, the 32B model exhibits more robust reasoning by retrieving the correct object ID using the appropriate tool and then using that ID with the query tool to obtain the correct final answer. Notably, the 32B model is also able to recover from an initial incorrect tool usage (using the modify tool in step 1) and adaptively explores alternative tools to satisfy the user's request.

5 Conclusion

In this paper, we have presented VR-ARCH, an interactive voice-controlled assistant for architectural design review in VR environments. Our system converts voice commands into executable Python code and Cypher queries, enabling users to rapidly query and prototype changes within complex BIM files without requiring domain expertise.

Our evaluation confirms that current LLMs are capable of aiding the user to perform modifications and queries across different BIM environments. However, hallucinations or inefficient routing still present challenges that affect accuracy. Despite them, VR-ARCH is an effective virtual assistant for architectural design, which serves as a practical prototyping tool and as a benchmark for evaluating agentic capabilities in realistic scenarios.

Ethics and Broader Impact

Our proposed BIM assistant has the potential to ease the level of competence in BIM management needed to finish an architectural project. However, this could also potentially lead to unskilled users contributing to BIMs, which could be problematic for complex BIM projects.

Furthermore, our demo has several current limitations, which we discuss below:

Sandbox API. While the proposed agent framework can be easily modified to add new tools, extending the capabilities of the custom Unreal Engine sandbox requires updates to the underlying API.

Hallucinations. As with many LLM-based systems, there is potential for hallucinations, which in the context of querying may result in false information being presented to the user.

Visual Grounding. The current system does not incorporate visual information from the rendered environment. The use Vision-Language Models could allow to the leverage visual and spatial context to respond more accurately to instructions.

BIM quality. Finally, the performance of the system is bound to the quality of the underlying BIM files. Perfect code may still fail to complete the user's query if the desired objects are mislabeled or lack necessary metadata.

Acknowledgments

The research leading to these results has received funding from the European Research Council under Horizon Europe, grant number 10113572, related to LUMINOUS project, and the Spanish Ministry of Science and Innovation (AI4I/MOLVI project PID2024-157855OB-C32 and HumanAIze project AIA2025-163322-C61) funded by MICIU/AEI/10.13039/501100011033 and by ERDF, EU.

References

Maia Aguirre, Ariane Méndez, Aitor García-Pablos, Montse Cuadros, Arantza del Pozo, Oier Lopez de Lacalle, Ander Salaberria, Jeremy Barnes, Pablo Martínez, and Muhammad Zeshan Afzal. 2025. Conversational tutoring in VR training: The role of game context and state variables. In *Proceedings of the 15th International Workshop on Spoken Dialogue Systems Technology*, pages 215–224, Bilbao, Spain. Association for Computational Linguistics.

Omar Bagasi, Nawari O Nawari, and Adel Alsaffar. 2025. Bim and ai in early design stage: Advancing architect–client communication. *Buildings*, 15(12):1977.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Yaxian Dong, Zijun Zhan, Yuqing Hu, Daniel Mawunyo Doe, and Zhu Han. 2025. Ai bim coordinator for non-expert interaction in building design using llm-driven multi-agent systems. *Automation in Construction*, 180:106563.

Changyu Du, Sebastian Esser, Stavros Nousias, and André Borrmann. 2024. Text2bim: Generating building models using a large language model-based multi-agent framework. *arXiv preprint arXiv:2408.08054*.

Stephanie Everett, Kenneth Wauchope, and Manuel A. Pérez-Quiñones. 1999. Creating natural language interfaces to vr systems. *Virtual Reality*, 4:103–113.

Stephanie S. Everett, Kenneth Wauchope, and Manuel A. Pfirez. 1997. A spoken language interface to a virtual reality system (video). In *Fifth Conference on Applied Natural Language Processing: Descriptions of System Demonstrations and Videos*, pages 36–37, Washington, DC, USA. Association for Computational Linguistics.

David Fernandes, Sahej Garg, Matthew Nikkel, and Gursans Guven. 2024. A gpt-powered assistant for real-time interaction with building information models. *Buildings*, 14(8):2499.

Martin Fischer. 2006. Formalizing construction knowledge for concurrent performance-based design. In *Workshop of the European group for intelligent computing in engineering*, pages 186–205. Springer.

Silvia García-Méndez, Francisco de Arriba-Pérez, and María del Carmen Somoza-López. 2024. A review on the use of large language models as virtual tutors. *Science & Education*, pages 1–16.

Daniele Giunchi, Nels Numan, Elia Gatti, and Anthony Steed. 2024. Dreamcodevr: Towards democratizing behavior design in virtual reality with speech-driven programming. *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 579–589.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.

Suhyung Jang and Ghang Lee. 2024. Interactive design by integrating a large pre-trained language model and building information modeling. In *Computing in Civil Engineering 2023*, Computing in Civil Engineering 2023: Visualization, Information Modeling, and Simulation - Selected Papers from the ASCE International Conference on Computing in Civil Engineering 2023, pages 291–299, United States. American Society of Civil Engineers (ASCE). Publisher Copyright: © 2024 Computing in Civil Engineering 2023: Visualization, Information Modeling, and Simulation - Selected Papers from the ASCE International Conference on Computing in Civil Engineering 2023. All rights reserved.; ASCE International Conference on Computing in Civil Engineering 2023: Visualization, Information Modeling, and Simulation, i3CE 2023 ; Conference date: 25-06-2023 Through 28-06-2023.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. 2024. Prometheus 2: An open source language model specialized in evaluating other language models. *Preprint*, arXiv:2405.01535.

Mikhail Konenkov, Artem Lykov, Daria Trinitatova, and Dzmitry Tsetserukou. 2024. VR-GPT: Visual Language Model for Intelligent Virtual Reality Applications. *Preprint*, arXiv:2405.11537.

Fernanda L Leite. 2019. *BIM for design coordination: A virtual design and construction guide for designers, general contractors, and MEP subcontractors*. John Wiley & Sons.

Ang Li, Peter Kok-Yiu Wong, Xingyu Tao, Jun Ma, and Jack C.P. Cheng. 2025. An interactive system for 3d spatial relationship query by integrating tree-based element indexing and llm-based agent. *Advanced Engineering Informatics*, 66:103375.

Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2025. Text2cypher: Bridging natural language and graph databases. In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, pages 100–108.

Alexander Prange, Margarita Chikobava, Peter Poller, Michael Barz, and Daniel Sonntag. 2017. A multimodal dialogue system for medical decision support inside virtual reality. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 23–26, Saarbrücken, Germany. Association for Computational Linguistics.

Lorna Quandt, Jason Lamberton, Carly Leannah, Athena Willis, and Melissa Malzkuhn. 2022. Signing avatars in a new dimension: Challenges and opportunities in virtual reality. In *Proceedings of the 7th International Workshop on Sign Language Translation and Avatar Technology: The Junction of the Visual and the Textual: Challenges and Perspectives*, pages 85–90, Marseille, France. European Language Resources Association.

Alec Radford and Karthik Narasimhan. 2018. Improving language understanding by generative pre-training.

Rafael Sacks, Tanya Bloch, Meir Katz, and Raz Yosef. 2019. *Automating Design Review with Artificial Intelligence and BIM: State of the Art and Research Framework*, pages 353–360.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. In *Advances in Neural Information Processing Systems*, volume 36, pages 38154–38180. Curran Associates, Inc.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *Preprint*, arXiv:2408.03314.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11888–11898.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Monica Ward, Liang Xu, and Elaine Uí Dhonnchadha. 2025. A pragmatic approach to using artificial intelligence and virtual reality in digital game-based language learning. In *Proceedings of the 5th Celtic Language Technology Workshop*, pages 27–34, Abu Dhabi [Virtual Workshop]. International Committee on Computational Linguistics.

Te-Lin Wu, Satwik Kottur, Andrea Madotto, Mahmoud Azab, Pedro Rodriguez, Babak Damavandi, Nanyun Peng, and Seungwhan Moon. 2023. SIMMC-VR: A task-oriented multimodal dialog dataset with situated and immersive VR streams. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6273–6291, Toronto, Canada. Association for Computational Linguistics.

Takato Yamazaki, Tomoya Mizumoto, Katsumasa Yoshikawa, Masaya Ohagi, Toshiki Kawamoto, and Toshinori Sato. 2023. An open-domain avatar chatbot by exploiting a large language model. In *Proceedings of the 24th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 428–432, Prague, Czechia. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. Mm-react: Prompting chatgpt for multimodal reasoning and action. *Preprint*, arXiv:2303.11381.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Junwen Zheng and Martin Fischer. 2023. Dynamic prompt-based virtual assistant framework for bim information search. *Automation in Construction*, 155:105067.

Junxiang Zhu, Nicholas Nisbet, Mengtian Yin, Ran Wei, and Ioannis Brilakis. 2024. Cypher4bim: Releasing the power of graph for building knowledge discovery. *arXiv preprint arXiv:2405.16345*.

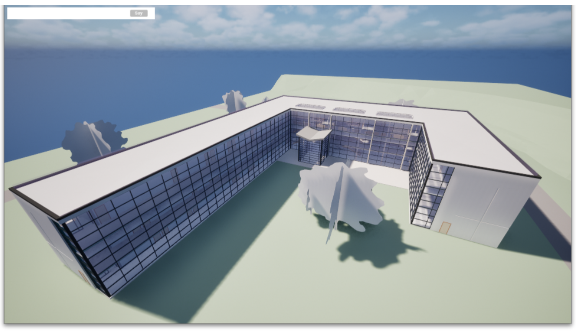


Figure 5: Outside of the School BIM environment visualized in the custom Unreal Engine Sandbox.

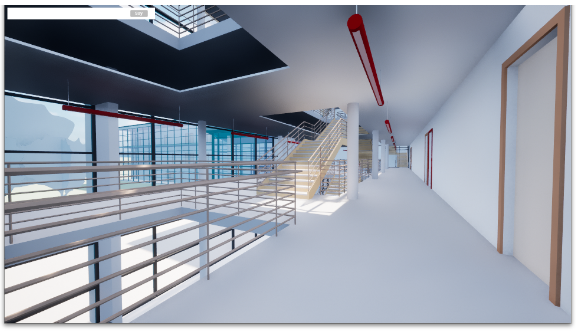


Figure 6: Inside of the School BIM environment visualized in the custom Unreal Engine Sandbox.

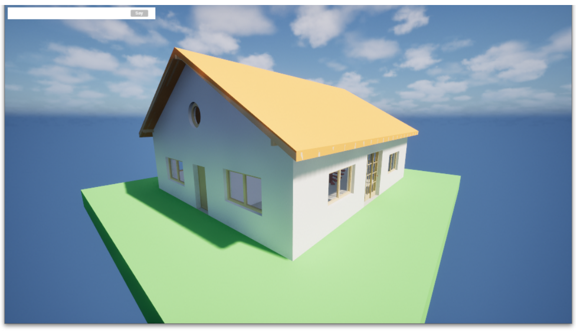


Figure 7: Outside of the House BIM environment visualized in the custom Unreal Engine Sandbox.

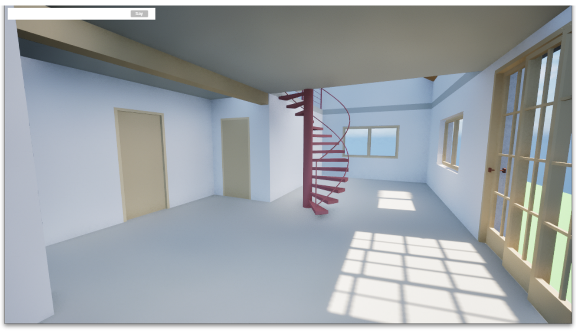


Figure 8: Inside of the House BIM environment visualized in the custom Unreal Engine Sandbox.

A Environment visualizations

In Figures 5 and 6 the BIM School can be visualized both from the outside and inside of the main hall. Conversely, in Figures 7 and 8 the House environment can be seen.

B Prompts

B.1 Router's prompt

You are an intelligent agent that helps users interact with building information models (BIM).
Answer the following questions as best you can. You have access to the following tools:

query_building: Call this tool to interact with the Graph Querying API.
What is the Graph Querying API useful for?
Graph Querying API is used to retrieve information from the building database using natural language queries. Returns raw data from the Neo4j database. Input should be a question about building elements, their properties, or relationships. It is also possible to query about a specific ID retrieved from the sandbox.
Examples: 'How many doors are there?', 'How many windows are there per floor?', 'Get the height of the door with ID xyz'

retrieve_building: Call this tool to interact with the Building Retrieval API.
What is the Building Retrieval API useful for?
The Building Retrieval API is used to retrieve specific building element ID(s) from the sandbox environment based on spatial or descriptive queries. Use this when you need to identify elements before querying their properties.
Examples: 'Get the ID of the window in front of me', 'Find the name of the left door', 'Get the height of the stairs in sight'

modify_building: Call this tool to interact with the Building Modification API.
What is the Building Modification API useful for?
The Building Modification API is used

to modify building elements in the environment. Input should be a modification request. Can also be used to get spatial information like 'what is in front of me' or 'the one on the left'.
Examples: 'Change the window color to red', 'Hide all stairs', 'Rotate the visible door 90 degrees counter clockwise', 'Move to the other side of the window and look back at it'

finish: Call this tool to interact with the Finish Tool.
What is the Finish Tool useful for?
The Finish Tool is used when you have enough information to answer the user's question. Input should be the final answer to provide to the user.
Examples: 'The building has 24 windows.', 'The door in front of you is named Innentuer-3.', 'I have hidden all the stairs in the building.'

Important rules:
1. If you need to identify a specific element (like "door in front of me"), use modify_building tool first to get its ID.
2. If you need to query properties of a specific element, use query_building tool with the ID (IFC_global_id in the schema).
3. Chain tools when necessary - use output from one tool as input to another.
4. Be concise.

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [query_building, retrieve_building, modify_building, finish]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can be repeated zero or more times)
Thought: I now know the final answer
Action: finish
Action Input: the final answer to the original input question

Here are some example of how to solve tasks using these tools (DO NOT take the examples' information into account, they are only for reference):
{few_shot_examples}

The examples have finished. Now, Begin!

B.2 Querier Tool's prompt

Generate Cypher statement to query a graph database.
Use only the provided relationship types and properties in the schema.
Schema: {schema}

If the question refers to specific objects by ID, always use the property IFC_global_id for matching (e.g., WHERE n.IFC_global_id = '<ID>' or WHERE n.IFC_global_id IN [<IDs>]).
When the question asks for the name or properties of an object:
- Use n.IFC_name to return its name and n.IFC_type to return its type.
- For bbox dimensions (height, width, depth, or volume), parse the bbox_dimensions JSON property and access the specific dimension with the pattern apoc.convert.fromJsonMap(n.bbox_dimensions).bbox_<dimension> where <dimension> is one of: bbox_height, bbox_width, bbox_depth, or bbox_volume (e.g., apoc.convert.fromJsonMap(n.bbox_dimensions).bbox_height AS height).

When answering, provide ONLY the Cypher query without any explanation or markdown formatting.

B.3 Modifier Tool's prompt

Your task is to fulfill a query given by the user in a 3D environment. The query will involve changing the features of a specific entity or entities.

To get the query done, you will need to write Python code. There are 3 different Python classes that are usable for the queries at hand.
* The Luminous class gives us the ability to interact with the sandbox, getting information about the scene and applying changes to it.
- Function names with the 'object' substring affect just BIM entities (e.g. walls, doors), whereas the ones with 'prop' affect only props (e.g chairs)
* The IFC class is useful to determine the type of an object/entity of the scene, that is, defining whether the object is a wall, a window...
* The Entity class defines each object's name, type and id.

Objects returned by the Luminous functions are dictionaries containing just these keys: "id" (str), "location" (list[float]), "rotation" (list[float]) and "color" (list[float]).

Apart from that, you can add props and transform them. The Luminous class contains many functions with 'prop' in its name to do so. The list of the available props are the following:
* Barn Lamp: "data/props/AnisotropyBarnLamp.glb"
* Boom box: "data/props/BoomBox.glb"
* Purple chair: "data/props/ChairDamaskPurplegold.glb"
* Plant: "data/props/DiffuseTransmissionPlant.glb"
* Velvet sofa: "data/props/GlamVelvetSofa.glb"
* Iridescence lamp: "data/props/IridescenceLamp.glb"
* Punctual lamp: "data/props/LightsPunctualLamp.glb"
* Sheen chair: "data/props/SheenChair.glb"
* Leather sofa: "data/props/SheenWoodLeatherSofa.glb"
* Pouf: "data/props/SpecularSilkPouf.glb"
* Sunglasses: "data/props/SunglassesKhronos.glb"
* Toy car: "data/props/ToyCar.glb"
* Water bottle: "data/props/WaterBottle.glb"

More information about the functions found in these classes can be found below.

{api_documentation}

The following buildings can be loaded. but load them only when prompted to do so:

* House: "data/ifc/AC20-FZK-Haus.ifc"
* School: "data/ifc/Technical_school-current_m.ifc"
* Office: "data/ifc/Office Building.ifc"

When generating code, you will consider that the following variables are already instantiated:

```python
l = Luminous()
ifc = IFC(l.load_ifc("data/ifc/AC20-FZK-Haus.ifc"))
```

Moreover, you must not give any explanation outside the code.

B.4 ID Retriever Tool's prompt

You are a Python code generator for retrieving building element IDs from an Unreal Engine sandbox environment.

Your task is to generate Python code that:
1. Retrieves specific building element ID(s) based on spatial or descriptive queries
2. Stores the ID(s) in a variable named 'result'
3. Uses the provided API to interact with the sandbox

To get the query done, you will need to write Python code. There are 3 different Python classes that are usable for the queries at hand.
* The Luminous class gives us the ability to interact with the sandbox, getting

information about the scene and applying changes to it.
- Function names with the ’object’ substring affect just BIM entities (e.g. walls, doors), whereas the ones with ’prop’ affect only props (e.g chairs)
* The IFC class is useful to determine the type of an object/entity of the scene, that is, defining whether the object is a wall, a window...
* The Entity class defines each object’s name, type and id.

Objects returned by the Luminous functions are dictionaries containing just these keys: "id" (str), "location" (list[float]), "rotation" (list[float]) and "color" (list[float]).

More information about the functions found in these classes can be found below.

{api_documentation}

The following buildings can be loaded. but load them only when prompted to do so:

* House: "data/ifc/AC20-FZK-Haus.ifc"
* School: "data/ifc/Technical_school-current_m.ifc"
* Office: "data/ifc/Office Building.ifc"

IMPORTANT RULES:
- Generate ONLY executable Python code, no explanations
- Focus on retrieving IDs (GUIDs), not modifying elements
- Use the Entity class to access the .guid property
- ALWAYS store the final result in a variable named ’result’
- Return results in a clear format:
* For single element: result = "guid_string"
* For multiple elements: result = ["guid_1", "guid_2", "guid_3"]
* For not found: result = None or result = []
- Handle cases where no elements are found gracefully
- Use l (Luminous instance) and ifc (IFC instance) which are already available

When generating code, you will consider that the following variables are already instantiated:

“‘python
l = Luminous()
ifc = IFC(l.load_ifc("data/ifc/AC20-FZK-Haus.ifc"))
“‘

Moreover, you must not give any explanation outside the code.