Developing a RAG-Based System for Natural Language Access to Linked Building Data on Construction Sites

Lukas Kirner[1,*,†], Jyrki Oraskari[1,†] and Sigrid Brell-Cokcan[1,†]

[1]Chair for Individualized Production, RWTH Aachen University, 52074 Aachen, Germany

Abstract

Digitalizing construction site workflows requires handling heterogeneous data from various sources in a user-friendly way. Linked Building Data (LBD) offers a powerful framework for this but remains largely inaccessible to on-site users unfamiliar with Semantic Web technologies. To address this, we developed a natural language interface integrated into a 3D construction site-focused LBD viewer, enabling intuitive access to structured and unstructured data without requiring SPARQL or RDF expertise. Our system employs a locally hosted large language model (LLM) and a Retrieval-Augmented Generation (RAG) pipeline to retrieve and generate responses based on both static and real-time data. We detail the architecture, implementation, and trade-offs of three RAG design approaches and present a robust, prebuilt query pipeline tailored for dynamic construction site use cases. This work demonstrates the feasibility, limitations, and potential of LLM-enhanced access to LBD for non-expert users in real-world construction environments.

Keywords

RAG, LLM, Linked Building Data, Construction Site Management, Usability

1. Introduction

The construction industry is characterized by a multitude of heterogeneous data sources, creating a fragmented digital landscape. Interoperability across these sources is essential for effective collaboration and addressing environmental, social, and economic sustainability goals—especially as construction increasingly adopts digital and robotic technologies. Linked Building Data (LBD) offers a promising approach to improve interoperability [1]. However, despite its potential and existing implementations focused on Building Information Modeling (BIM), LBD remains largely inaccessible to on-site users such as field engineers, equipment operators, and logistics coordinators.

This is mainly due to two factors: (1) on-site users predominantly lack familiarity with Semantic Web technologies like SPARQL and RDF, and (2) most existing tools require technical setups, e.g. bringing a laptop to the work area, which is deemed unsuitable for field environments. This gap has been noted across academic and industry settings, and got particularly evident in the EConoM research project, which investigates AI, edge computing, and 5G for constructions site management. One of its key use cases, autonomous intralogistics, relies on integration and user interaction of data about logistics processes, building elements, materials, and real-time telemetry from mobile robots [2, 3].

To improve accessibility for such use cases, we began developing a web-based software prototype. As a foundation, we adapted the open-source *three.js*-based FOG demo application by Mathias Bonduel [4][1]. The resulting viewer supports endpoint selection, loads geometry generated via the IFCtoLBD converter [5], and provides a SPARQL interface alongside a result viewer (see Figure 1). While this prototype demonstrated the benefits of LBD, it was hardly helpful for on-site users as many of their requirements were not met. These included support for real-time data integration (e.g., via MQTT or

LDAC 2025: 13th Linked Data in Architecture and Construction Workshop, July 09–11, 2025, Porto, Portugal

*Corresponding author.

†These authors contributed equally.

lukas.kirner@rwth-aachen.de (L. Kirner)

https://www.ip.rwth-aachen.de/ (L. Kirner)

0000-0001-9753-2422 (L. Kirner); 0000-0002-4723-3878 (J. Oraskari); 0000-0003-1463-7515 (S. Brell-Cokcan)

[1]https://github.com/mathib/fog-demo-app

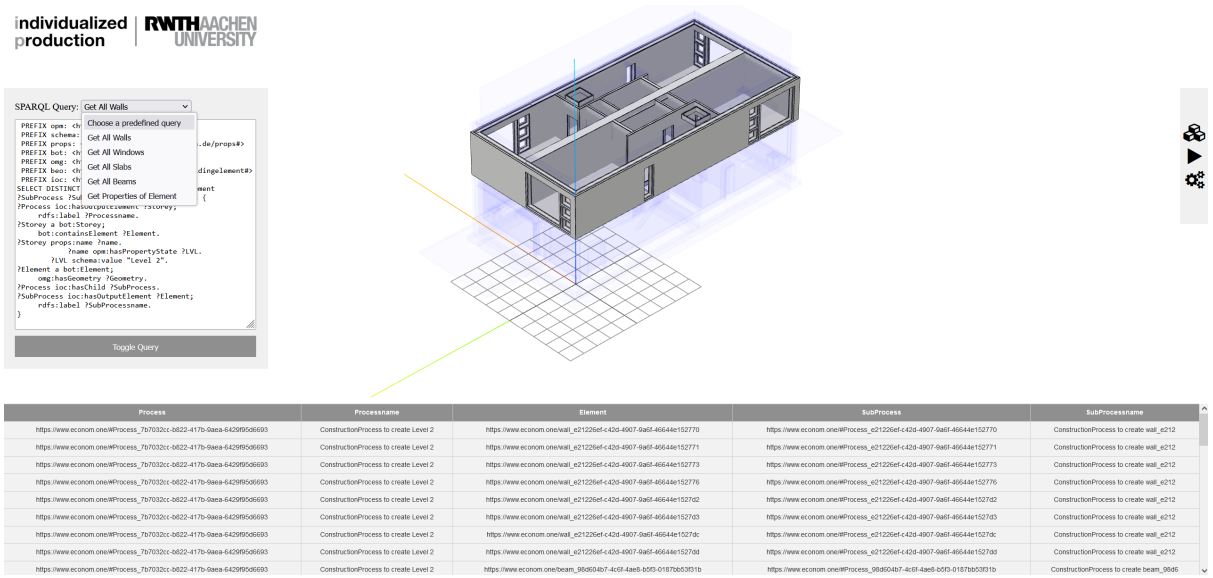

Figure 1: Simple LBD-viewer with a SPARQL interface based on the FOG demo app [4].

ROS), access to non-RDF sources like time-series databases, and reduced reliance on SPARQL. Motivated by these needs and inspired by the LDAC Summer School 2024 and its *"LLM+RAG+Ontologies"* repository[2], we explored a Retrieval-Augmented Generation (RAG) approach powered by a local large language model (LLM) [6]. RAG enables LLMs to retrieve relevant external data based on user input and combine it with their internal knowledge to generate context-aware responses [7], making it well-suited for accessing both static and dynamic construction data.

This paper presents the design and implementation of an RAG-powered LBD interface tailored for real-world construction environments. We evaluate different RAG strategies and propose a robust architecture based on prebuilt queries, assessing its feasibility, limitations, and potential for broader application.

2. Related work

Not only since the wide availability of LLMs and RAG techniques, semantic data processing with the help of natural language has sparked interest among researchers exploring new ways to retrieve, analyze, and interact with construction data. In 2016, Lin et al. [8] introduced a cloud-based natural language retrieval system for BIM, enabling non-experts to query data intuitively. Their approach used MongoDB, MapReduce, and IFC-based keyword mapping to enhance query efficiency and semantic understanding. On the other hand, LD-BIM [9] is a notable first-generation tool that enables users to query Linked Building Data (LBD) using natural language. It can understand plain English commands such as "Show windows" and focuses on queries that list elements of a specific type. In contrast, our approach extends beyond BIM models. We offer a chat interface that allows users to query and create content through conversation and interact with devices on a construction site.

Zheng et al. [10] proposed a Knowledge Graph (KG)-enhanced approach to improve LLM-based contract risk identification, demonstrating better fact recall and domain-specific reasoning. Automated Reporting and Review uses large language models (LLMs) for automated reports in construction. Pu et al. [11] introduce AutoRepo, a framework that uses unmanned vehicles and multimodal large language models to automate inspection report generation. Tested on a real-world site, it speeds up inspections, reduces resource waste, and meets regulatory standards. On the other hand, Cruz-Castro et al. [12] have examined using an LLM-based system for real-time feedback on technical reports in AEC education, addressing the challenge of time-consuming traditional evaluations.

[2] https://github.com/jakob-beetz/LDAC2024

LLM models face challenges such as inaccurate responses and made-up answers. Uhm et al. [13] evaluated the Retrieval-Augmented Generation-based Generative Pre-Trained Transformers (RAG-GPT) model to generate detailed construction safety information. The RAG-GPT model outperformed other models, with experts praising its contextual relevance and accuracy. Yi et al. [14] discuss using a generative pre-trained transformer (GPT) model to organize and retrieve data using a dataset from over 200 wells, including various reports and data types. The model allows users to access information and provides answers with references, addressing concerns about accuracy. Further, natural language processing (NLP) techniques can be used to match data within a semantic model. One approach is to create a similarity index for triple patterns, such as comments or literals. Information retrieval leverages the semantic similarity between a given sentence and the sought-after information. For that, Random Projection is utilized in GraphDB[3] as part of the Semantic Similarity Searches plugin within the program.

3. Requirements and Design Decisions

The primary use case for the system was its deployment on the local edge server of the reference construction site in Aachen, a full-scale construction testbed at RWTH Aachen University. This computational core, running a large number of dockerized services, is connected to a dedicated 5G campus network, enabling low-latency, high-throughput communication with all machines and devices on-site.

Due to the network topology and security constraints, the RAG system had to be split into at least two components: a user interface (UI) running on the user's device, and backend RAG services deployed on the edge server, where access to databases and machine APIs is provided. Several existing solutions could be integrated into the LBD viewer with minimal modification for the UI. Since the edge server is equipped with four Nvidia A40 GPUs (48 GB VRAM each), deploying the LLM locally via Ollama [15] was a natural fit. Ollama is a software tool designed to simplify hosting open-source LLMs in containerized environments.

However, multiple approaches to implementing RAG exist, each with distinct trade-offs. The following section outlines three options we investigated, along with a comparative summary of their respective strengths and limitations (see Table 1).

3.1. Tested RAG Approaches

Vector Database with Data Embeddings This method involves storing data as dense embeddings in a vector database to capture semantic relationships. Embeddings are fixed-size, high-dimensional vectors generated by the LLM to represent concepts or text. Similarity search over these vectors enables retrieving the most relevant content without explicit keyword matching. While powerful for static, document-like data, this approach performs poorly for dynamic or real-time use cases due to the need for frequent, costly re-embedding.

Fully LLM-generated Queries This approach leverages the LLM's ability to dynamically create SPARQL or SQL queries based on natural language input and an understanding of the data schema. For example, the model can convert a user query into a valid SPARQL command by referencing an embedded ontology. While this enables flexible and expressive querying, the generated queries often suffer from syntax errors or semantic mismatches, especially in heterogeneous data environments.

Pipeline with Prebuilt Queries and APIs This method employs a modular execution pipeline in which the LLM selects and parameterizes prebuilt or partially prebuilt scripts. These are connected to specific RDF stores, time-series databases, and real-time data streams. Although this approach requires more manual setup, it ensures predictable behavior, simplifies testing and debugging, and allows for fine-grained control over sensitive or safety-critical operations.

[3] https://www.ontotext.com/products/graphdb/

Table 1
Comparison of investigated approaches for RAG implementation.

Approach	Advantages	Limitations
Vector Database with Data Embeddings [16]	• Supports intuitive semantic search for unstructured knowledge [16]. • Effective for retrieving static information, e.g., from document repositories.	• Ineffective for real-time or dynamic data (e.g., machine states). • Frequent re-embedding is required for updates; it is computationally intensive [17]. • Development experience hindered by lack of repeatability and traceability.
Fully LLM-generated Queries [18]	• Highly flexible; LLM generates queries for diverse data sources. • Intermediate query evaluation simplifies development. • Reduces manual query design effort for complex datasets.	• Error-prone; generated queries may not match schema or ontology. • Complexity increases with the number of integrated sources. • Embedding ontologies requires fine-tuning and documentation. • Minor syntax issues (e.g., typos) can cause complete failures.
Pipeline with Prebuilt Queries and APIs [19]	• Predictable and robust behavior due to manually defined queries. • Suitable for real-time systems (e.g., MQTT, ROS). • Scalable and maintainable in constrained, predefined environments.	• Less flexible than dynamic approaches. • Requires upfront effort to design query library. • Limited to available, predefined queries.

3.2. Rationale for the Pipeline Approach with Prebuilt Queries

Given the dynamic nature of construction site processes, machine data, and the architectural constraints of the edge server infrastructure, the pipeline approach with prebuilt queries emerged as the most suitable option. Unlike other methods, it supports persistent RDF datasets and near-real-time data streams or machine control via dedicated APIs. This flexibility allows the system to bridge static knowledge bases and dynamic sources while maintaining architectural modularity. A major strength of this approach is its predictability: prebuilt queries and scripted logic ensure stable and repeatable behaviour. This is particularly important when user interactions do not simply retrieve data but can also create new data entries or trigger machine actions. In such contexts, reliability and system safety take precedence over flexibility. Although the pipeline approach requires more upfront development effort, we considered this trade-off essential to ensure robust performance and operational security. In contrast, while theoretically more flexible, dynamically generated queries based on embedded ontologies proved error-prone, difficult to debug, and hard to predict in production scenarios. Repetition of similar queries (e.g., highlighting specific building elements) also introduces unnecessary computational overhead without added user value.

4. Implementation

4.1. General Setup

To implement a functional prototype, we designed a layered system architecture using modular, open-source tools, specifically from Open WebUI's pipelines toolchain[4]. The architecture follows a clear separation of concerns: each layer is responsible for a specific function within the data retrieval and interaction pipeline. This modularity improves maintainability, allows distributed deployment across constrained edge hardware, and ensures safe execution of potentially sensitive operations such as machine control or data creation.

The architecture consists of five layers (see Figure 2):

[4]https://github.com/open-webui/pipelines

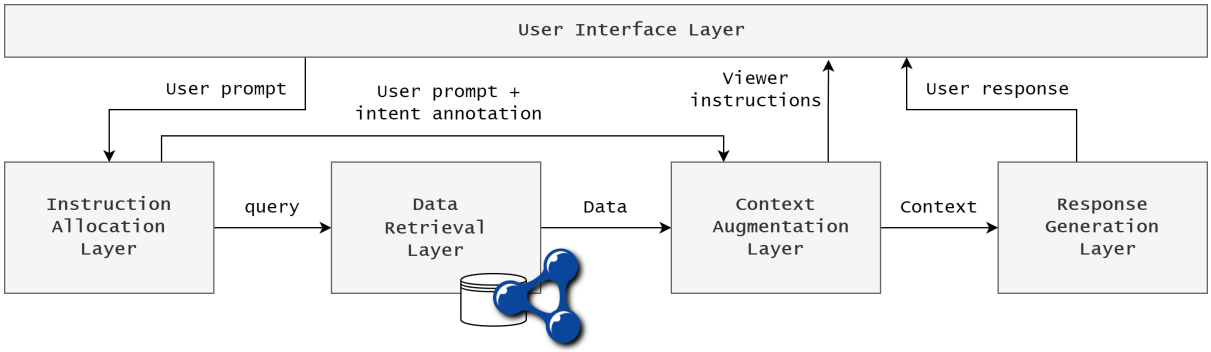

Figure 2: Simplified overview of the System architecture and its layers.

- **User Interface Layer**: Enables user interaction through natural language queries and visualization of data. Built by embedding Open WebUI's chat interface into a Three.js-based LBD viewer.
- **Instruction Allocation Layer**: Matches user intent with available functions and API calls. Implemented using custom Python code adapted from Open WebUI's pipelines toolkit.
- **Data Retrieval Layer**: Fetches information from RDF stores (via SPARQL), time-series databases, and real-time sources like MQTT or ROS.
- **Context Augmentation Layer**: Enriches the user query with retrieved data to prepare a structured prompt for the LLM.
- **Response Generation Layer**: Uses a locally deployed LLM to generate responses. This layer can also interact with the viewer via MQTT to update tables or visual highlights.

4.2. The Web Interface

The existing JavaScript implementation was updated to use a newer version of the Comunica library for more advanced SPARQL queries. Due to changes in the library's API, this required modifying the internal logic used to process and display result sets. We chose React/Node.js as the frontend framework to improve modularity and maintainability, replacing global JavaScript variables with React Context, Hooks, and Props. This change also enabled the integration of React Three Fiber for 3D rendering.

The conversational chat interface is based on Open WebUI and is embedded into the application via an HTML iframe. The backend LLM agent communicates with the frontend through MQTT channels, pushing updates such as SPARQL query results or structured messages. These are used to populate tables, highlight 3D elements, or provide feedback directly in the chat window (see Figure 3). While the modular design allows for easy extension and reconfiguration, it also introduces complexity in user session handling and authentication. For example, ensuring consistent identity across the LBD viewer and the embedded chat remains a challenge and is the subject of ongoing work.

4.3. Linked Data Retrieval

One of the first and simplest functionalities to test the system was retrieving LBD data by querying for specific building elements, using filters based on Building Element Ontology (BEO) classes or the Building Topology Ontology (BOT). For the BEO, this was achieved by passing the variable *Etype* (element type) into the Python script, which was then used in a simple SPARQL SELECT query. Since BEO aligns closely with IFC class names, mapping the user's query to the nearest IFC class and passing it as *Etype* worked well for most elements. When using language-agnostic models (we tested Llama 3.1 (8B and 70B) [20] as well as Mistral NeMo 12B [21]), this worked for many languages. To our surprise, successful queries were created in Rōmaji by a visiting Japanese delegation. For example,

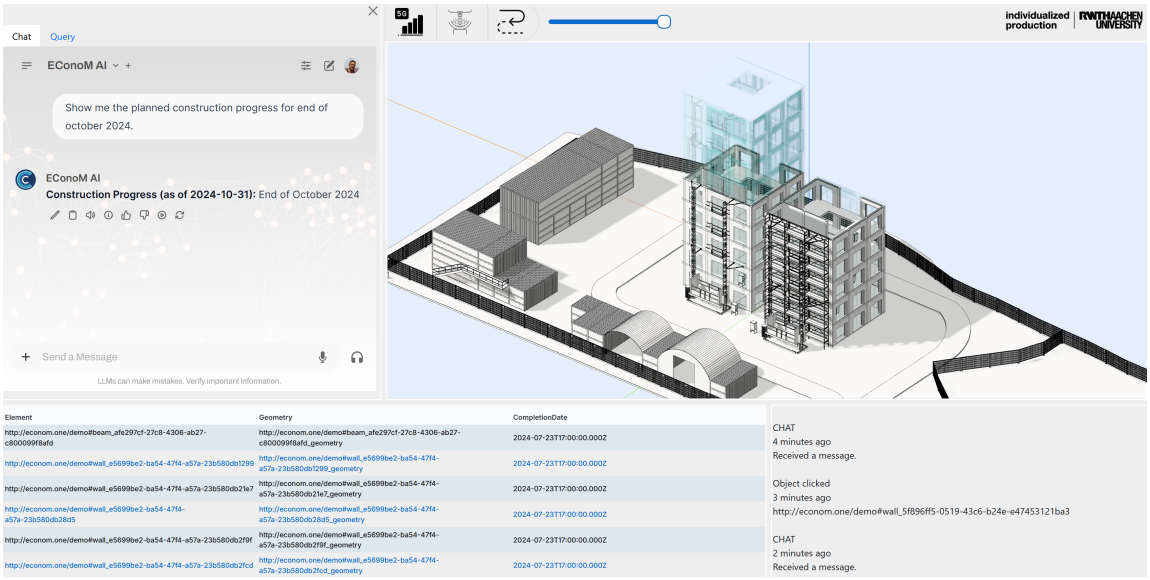

Figure 3: The RAG-enhanced web application with embedded chat (left), 3D viewer (right), and query result table (bottom).

the prompt “ウィンドウタイプのコンポーネントをすべて表示する” (”Display all components of a window type”) correctly retrieved all elements of class *beo:Window*.

To ensure functions could be discovered and invoked by the Instruction Allocation layer, they have to be clearly described. This includes what each script does, what parameters it takes, and what it returns (see Listing 1).

Listing 1: Description for the python definition retrieving the OLCC transport details.

```
"""
Gets the detail information of a OLCC transport for a specific transport ID.

:param id: The ID of the specific OLCC transport for which we want more information.
:return: The detailed information about the specific OLCC process as markdown.
"""
```

This description was used for a function retrieving logistics process data from the Online Logistics Control Center (OLCC), used by Zeppelin Rental GmbH for tracking and billing of material transports. In our observed use cases, users typically start by requesting all inbound shipments for a given day. A follow-up query, such as “Return all data for OLCC ID 187005,” triggers the function to fetch the shipment details. The SPARQL query used in the backend is shown in Listing 2.

Listing 2: Query for the logistics data which belong to a certain *OLCC* ID.

```
PREFIX olcc: <http://w3id.org/olcc#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

DESCRIBE ?resource
WHERE {?resource olcc:buchungsId "{id}"^^xsd:integer.}
```

The resulting RDF graph is pre-processed and returned in multiple ways. To keep the chat interface clean, only a short message is displayed, while full tabular data is transmitted via MQTT to the viewer and shown in the application table. An interesting feature of the response generation layer is that it can distinguish between internal “knowledge” (e.g., JSON results not shown to the user) and final user-

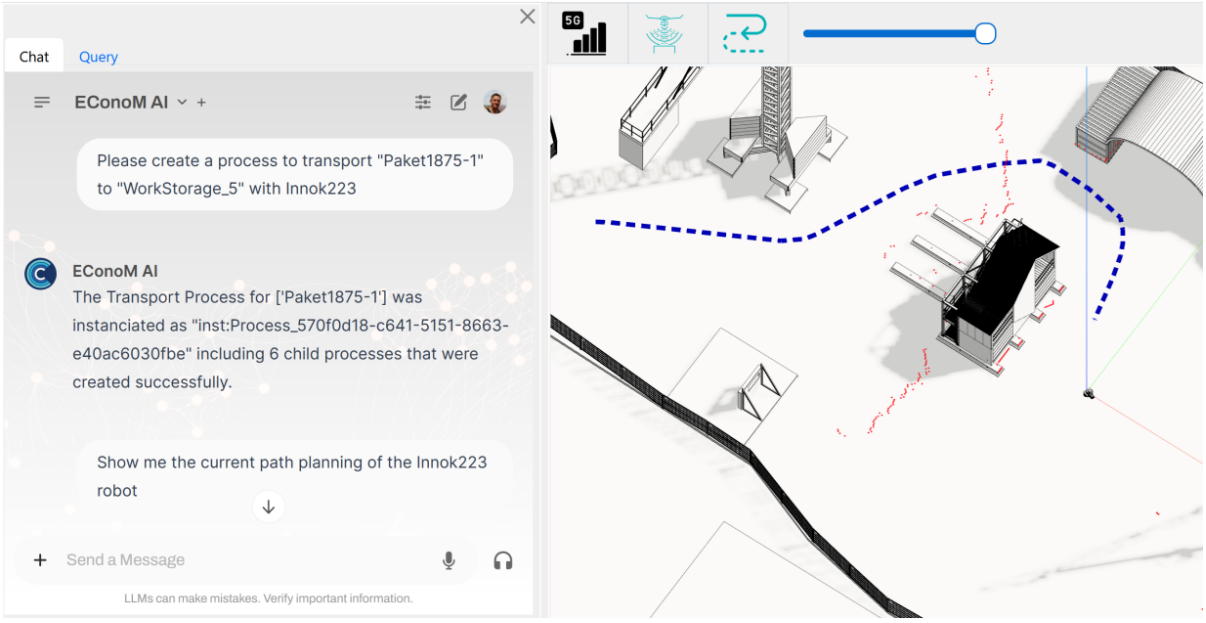

Figure 4: A chat interaction to generate a robotic transport process. The resulting robotic path planning is prompted and highlighted in the viewer.

facing output. This allows, for example, internal logistics metadata like *olcc:ladezone* (loading zone) to remain in context and be retrieved later when the user asks follow-up questions such as “Where is the last known position of the package?”

More complex queries require more attention on the data retrieval layer, such as prompting for scheduled construction progress, as shown in Figure 3. For our construction site data model, we can query planned and accomplished construction progress using the *Internet of Construction Process Ontology (IoC)*[5], specifically the concepts of *ioc:Schedule* and *ioc:Status*. However, the instances we want to filter are modelled using *xsd:dateTime* literals according to ISO 8601 [22]. Users tend not to use these descriptions and refer to time using phrases like ”calendar week 42” or relative expressions such as ”end of next week”.

To address this, we implemented a lightweight solution using an instruction-tuned language model (Mistral-Nemo-Instruct) to convert natural language date expressions into ISO 8601 format. The function can be accessed directly by the instruction allocation layer or triggered as a fallback when a regex check on a date parameter fails for ISO compliance. To ensure reliable parsing, we made efforts to prevent the model from adding annotations or altering the structure of the output. While instruct models are fine-tuned to follow explicit prompts, they also retain general world knowledge. For example, prompting the model with “the day Columbus reached the new world” correctly produces the ISO-compliant literal *”1492-10-12T02:00:00Z”* for use in filtering queries.

4.4. Generating process information

Data availability can certainly be improved with the presented RAG approach using LBD. More promising, however, are the opportunities to generate data on demand in the field, e.g., for the planning and execution of robotic processes, as there are currently no accessible methods for doing so [23]. Since the initial effort of the user, before any possible increase in efficiency, is often decisive for the successful introduction of new technologies, the concept presented here could contribute decisively to making construction robotics more attractive for construction practice. Figure 4 shows a chat interaction to instantiate a process chain using the *ioc:Process* concept.

The prompt contains the parameters of what to transport (”Paket1875-1”), where to transport it to

[5]https://internet-of-construction.github.io/IoC-Process-Ontology/

("WorkStorage_5"), and which machine to use ("Innok223"). At the current stage, these parameters must either be explicitly provided by the user or inferred from the context of previous answers. While this approach enables flexibility, it also poses challenges to usability, as users may not always know or remember the exact identifiers needed. We consider this a limitation of the current implementation and plan to integrate more robust disambiguation and auto-suggestion mechanisms in future iterations. The chat response shows that one process has been instantiated and also mentions that six child processes have been created. These processes depend on the machine requested. The *Innok Robotics Heros 223* is an autonomous ground vehicle (AGV) equipped with an automatic trailer coupling system. This means that logistics processes with this AGV follow a reference process scheme that is first queried. It consists of loading the trailer with the requested payload (1), moving the AGV to the trailer position (2), coupling the trailer (3), transporting to the target position (4), decoupling (5), and moving the AGV out of the target zone (6).

Generating the process sequence involves several validation steps. If any check fails—such as missing material location or unavailable target zone—the system responds with clarification requests or error messages. We plan to extend this validation further to include capability matching and integration with the machine scheduling system. Once the processes are instantiated, the backend services handle path planning and execution, which can be launched immediately if the schedule permits. As seen in Figure 4, the resulting path planning and current operation status are both formalized in RDF and visualized in the 3D viewer. While the system currently operates with predefined process templates, the underlying architecture could be extended to support standardized workflow definitions, such as BPMN, and executed through workflow engines (e.g., Camunda, WSO2). However, introducing such complexity would require additional semantic modeling effort and more granular user control over process instantiation and lifecycle management.

4.5. Digital Twin approach by integrating machines and their data

As the RAG pipeline has access to a Linked Data model, including machines, their network endpoints, associated APIs, and the 5G network infrastructure, it can be extended to retrieve machine data or even trigger control mechanisms on the construction site with relative ease. The web app can be used, for example, on a tablet and operated via voice using the integrated microphone and speech-to-text interface. Command success rates depend on the strictness settings of the pipeline, which can be adjusted by setting the model's temperature, and on the clarity of the user's speech. Figure 5 shows examples of how reality is shadowed in the Digital Twin.

For the autonomous transportation robot, the system can access onboard sensors and retrieve media streams, such as images or video, from its cameras. Through a time-series database, users can also query machine telemetry, such as battery status (shown at the bottom of Figure 5). In addition, live streams of data like rotational speed, location, or the robot's point cloud, used for obstacle detection, can be made available. These point cloud updates (visualized as red dots) provide temporal, real-time information not typically modeled. For example, the red line at the lower center of Figure 5 represents the open door of a construction container. Based on previous work in project [24], the system also supports access to network coverage simulations for the construction site. The increased accessibility of this information presents significant potential for improving decision-making on site (e.g., within a Last Planner System).

In addition to data retrieval, the system can also be used for actuation—controlling devices through exposed APIs. A practical example is a construction lift that can move vertically. Here, voice commands offer tangible benefits, allowing workers to control the lift remotely without dismounting from a vehicle. While this control functionality is technically feasible, it introduces critical safety concerns. Unlike passive data queries (e.g., requesting an onboard image), misinterpreted or imprecise control commands could lead to unsafe actions such as activating the wrong equipment. For this reason, safeguards must be put in place to ensure that potentially hazardous control operations are only executed with clear user intent and proper authentication.

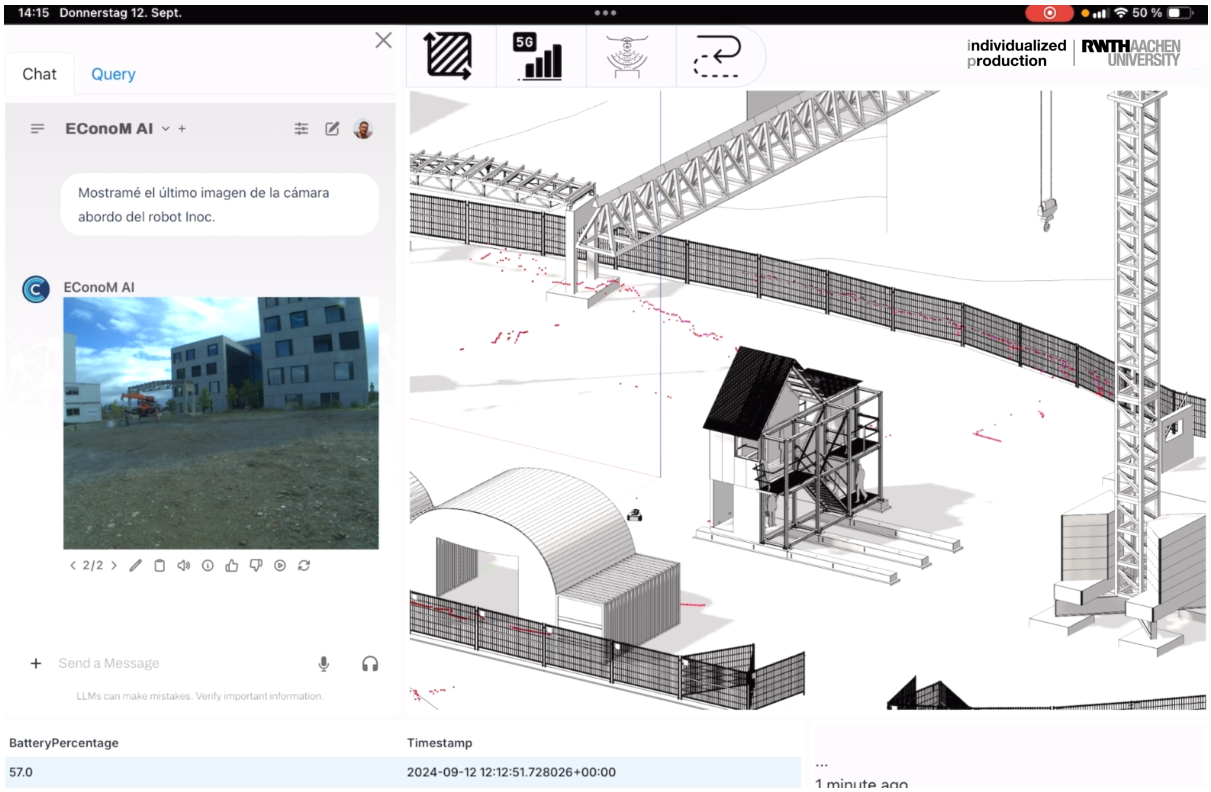

Figure 5: A chat interaction in Spanish to retrieve a current onboard image of the mobile robot and the answer as an embedded markdown image in the chat. In the 3D viewer, the robot and its point cloud stream data (small red dots) can be spotted.

5. Discussion

The presented RAG-based system for accessing Linked Building Data (LBD) and other construction site information demonstrates both significant strengths and critical limitations. To evaluate its overall impact, we outline the main strengths, challenges, opportunities, and associated risks.

Strengths and Usability Gains

A key advantage of the system is its ability to lower the entry barrier to LBD and other complex, structured data environments for non-expert users on construction sites. Users unfamiliar with SPARQL, RDF, or system-specific APIs can now interact with LBD models and digital twins using natural language. This enhances usability and accessibility compared to traditional interfaces. Beyond data retrieval, the system enables advanced functionalities such as initiating autonomous robotic processes or accessing real-time sensor streams without requiring technical expertise. The conversational interface contributes to user confidence by guiding interactions, requesting missing parameters, and providing feedback instead of cryptic errors.

Challenges and Limitations

While the use of prebuilt queries ensures predictable and safe behavior, it also limits system flexibility. The current approach restricts interactions to predefined templates and cannot yet accommodate arbitrary queries or dynamically composed logic. Error handling remains a challenge, particularly when users input vague or malformed parameters. This is especially noticeable in speech-based interaction, where identifiers like *WorkStorage_5* are prone to misinterpretation. As the system scales to include more data types and interaction patterns, retrieval accuracy and validation become increasingly important to ensure reliable operation and user trust.

Opportunities and Future Directions

Future improvements should address scalability and adaptability. They could integrate specialized LLMs optimized for structured retrieval, improving accuracy while reducing manual setup. Agent-based frameworks capable of refining user prompts, selecting appropriate data sources, and validating retrieved content could further increase robustness. The ability to instantiate and monitor robotic processes opens pathways toward adaptive scheduling, predictive diagnostics, and seamless human-machine collaboration. These extensions would support real-time decision-making and broaden the accessibility of robotic workflows in construction.

Risks and Safety Considerations

Integrating AI-based interaction into operational construction systems introduces serious safety and security concerns. While the current implementation enforces strict limitations on executable actions, the potential for voice or text-based commands to trigger real-world machine operations necessitates rigorous safeguards. Misinterpretations, unauthorized access, or malicious prompts could result in hazardous scenarios. Future iterations must implement multi-layer authentication, role-based permissions, and validation checkpoints before executing any critical command.

6. Conclusion

This work presents an approach for improving access to Linked Building Data through a Retrieval-Augmented Generation (RAG) architecture powered by a locally hosted large language model (LLM). The system enables natural language interaction with complex structured and real-time data, lowering the barrier to entry for non-expert users in construction. By embedding a conversational interface into a 3D viewer and integrating prebuilt queries, the system supports both robust information retrieval and the generation of autonomous workflows, such as robotic material transport. While the pipeline approach ensures stability and safety, challenges remain regarding query flexibility, input robustness, and system scalability. Future work will focus on improving query interpretation, expanding support for dynamic instruction allocation, and refining safety mechanisms for interacting with physical systems. Recent developments in agent-based architectures [25] and domain-specific language modeling offer promising directions. This study provides a foundation for more intuitive access to LBD, setting the stage for AI-augmented digital twins, real-time construction data augmentation, and semi-autonomous construction site management systems.

Acknowledgments

This work is part of the EConoM research project funded by the Federal Ministry for Digital and Transport of Germany within the initiative InnoNT (funding number 19OI22009F). It was supported within the TARGET-X framework, a project funded by the Smart Networks and Services Joint Undertaking (SNS JU) under Horizon Europe (funding number 101096614). The authors are responsible for the content.

Declaration on Generative AI

During the preparation of this work, the authors used a local chatbot based on Mistral Nemo 12b and Deepl for spelling checks, translations and paraphrasing for shortening the text. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] P. Pauwels, Supporting decision-making in the building life-cycle using linked building data, Buildings 4 (2014) 549–579.

[2] J. Fottner, D. Clauer, F. Hormes, M. Freitag, T. Beinke, L. Overmeyer, S. Gottwald, R. Elbert, T. Sarnow, T. Schmidt, K.-B. Reith, H. Zadek, F. Thomas, Autonomous systems in intralogistics – state of the art and future research challenges, Logistics Research 14 (2021). doi:10.23773/2021_2.

[3] J. Oraskari, L. Kirner, M. Zöcklein, S. Brell-Cokcan, Towards human-machine collaboration in autonomous material handling on construction sites, Human-Machine Communication 9 (2024) 189–213. doi:10.30658/hmc.9.11.

[4] M. Bonduel, A. Wagner, P. Pauwels, M. Vergauwen, R. Klein, Including widespread geometry formats in semantic graphs using rdf literals, in: Proceedings of the 2019 European Conference on Computing in Construction, Chania, Greece, 2019, pp. 341–350. URL: https://ec-3.org/publications/conference/paper/?id=EC32019_166. doi:10.35490/EC3.2019.166.

[5] J. Oraskari, M. Bonduel, K. McGlinn, P. Pauwels, F. Priyatna, A. Wagner, V. Kukkonen, S. Steyskaland, J. Lehtonen, M. Lefrançois, Ifctolbd: Ifctolbd v 2.44.0, 2024. URL: https://github.com/jyrkioraskari/IFCtoLBD.

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al., Retrieval-augmented generation for knowledge-intensive nlp tasks, Advances in Neural Information Processing Systems 33 (2020) 9459–9474.

[7] V. Trimborn, Enhancing Project Knowledge Management in Construction: Integrating Generative AI and Large Language Models, Master's thesis, TUM School of Engineering and Design, Technical University of Munich, 2023. URL: https://mediatum.ub.tum.de/doc/1755185/document.pdf.

[8] J.-R. Lin, Z.-Z. Hu, J.-P. Zhang, F.-Q. Yu, A natural-language-based approach to intelligent data retrieval and representation for cloud bim, Computer-Aided Civil and Infrastructure Engineering 31 (2016) 18–33. doi:10.1111/mice.12151.

[9] M. MHolten Rasmussen, A. Schlachter, Ld-bim, https://ld-bim.web.app/, 2023. Accessed: 07 April 2025.

[10] C. M. Zheng, Y. Q. Tang, X. Su, A knowledge graph modeling approach for augmenting language model-based contract risk identification, in: Proceedings of the 2024 European Conference on Computing in Construction, European Council on Computing in Construction, Chania, Greece, 2024. URL: https://ec-3.org/publications/conference/paper/?id=EC32024_178. doi:10.35490/EC3.2024.178.

[11] H. Pu, X. Yang, J. Li, R. Guo, Autorepo: A general framework for multimodal llm-based automated construction reporting, Expert Systems with Applications 255 (2024) 124601.

[12] L. Cruz-Castro, G. Castelblanco, P. Antonenko, Llm-based system for technical writing real-time review in urban construction and technology, Proceedings of 60th Annual Associated Schools 5 (2024) 130–138.
[13] M. Uhm, J. Kim, S. Ahn, H. Jeong, H. Kim, Efficacy of retrieval augmented generation-based large language models for generating construction safety information, SSRN Electronic Journal (2024). URL: https://ssrn.com/abstract=4819837. doi:10.2139/ssrn.4819837.
[14] M. Yi, K. Ceglinski, P. Ashok, M. Behounek, S. White, T. Peroyea, T. Thetford, Applications of large language models in well construction planning and real-time operation, in: SPE/IADC Drilling Conference and Exhibition, SPE, 2024, p. D021S014R003.
[15] O. Team, Ollama: Open-source large language model serving platform, https://ollama.com, 2024. Accessed: 2024-02-11.
[16] J. Wang, E. Hanson, G. Li, Y. Papakonstantinou, H. Simhadri, C. Xie, Vector databases: What's really new and what's next?(vldb 2024 panel), Proceedings of the VLDB Endowment 17 (2024) 4505–4506.
[17] E. Krippner, Rethinking vector embeddings search for analytical database systems, 2024. URL: https://homepages.cwi.nl/~boncz/msc/2024-ElenaKrippner.pdf, master's thesis, Institut für Software & Systems Engineering, Augsburg.
[18] L. Kovriguina, R. Teucher, D. Radyush, D. Mouromtsev, Sparqlgen: One-shot prompt-based approach for sparql query generation., in: SEMANTiCS (Posters & Demos), 2023.
[19] L. Contributors, Langchain: Building applications with llms, 2025. URL: https://github.com/hwchase17/langchain, accessed: 2025-04-07.
[20] Grattafiori et al., The llama 3 herd of models, arXiv preprint arXiv:2407.21783 (2024). doi:10.48550/arXiv.2407.21783.
[21] NVIDIA and Mistral AI, Mistral nemo 12b base model, Hugging Face, 2024. URL: https://huggingface.co/nvidia/Mistral-NeMo-12B-Base, accessed 2025-02-11.
[22] International Organization for Standardization, Data elements and interchange formats – information interchange – representation of dates and times, 2019. ISO 8601-1:2019, Geneva, Switzerland.
[23] J. Owen, Ai and robotics: Transforming automation and labor productivity in the construction industry, EasyChair Preprint 15086, EasyChair, 2024.
[24] L. Kirner, J. Oraskari, M. Zöcklein, A. Möller, S. Brell-Cokcan, An ontology for signal strength estimation of nomadic 5g networks on construction sites, in: Proceedings of the 2024 European Conference on Computing in Construction, European Council on Computing in Construction, Chania, Greece, 2024. URL: https://ec-3.org/publications/conference/paper/?id=EC32024_202. doi:10.35490/EC3.2024.202.
[25] A. Singh, A. Ehtesham, S. Kumar, T. T. Khoei, Agentic retrieval-augmented generation: A survey on agentic rag, arXiv preprint arXiv:2501.09136 (2025).