A new Graph-based Retrieval-Augmented Generation (RAG) approach for querying and instantiating large-scale industrial semantic artifacts

Nilay Tufek [a,*], Burak Yigit Uslu [a], Valentin Philipp Just [b], Tathagata Bandyopadhyay [a], Aparna Saissre Thuluva [a], Marta Sabou [c] and Allan Hanbury [d]

[a] *Technology Department, Siemens AG, Munich, Germany*
E-mails: ntufek@gmail.com, burak.uslu@siemens.com, tathagata.bandyopadhyay@siemens.com, aparna.thuluva@siemens.com
[b] *Institute of Computer Engineering, Technical University of Vienna, Vienna, Austria*
E-mail: valentin.just@tuwien.ac.at
[c] *Institute for Data, Process and Knowledge Management, Vienna University of Economics and Business, Vienna, Austria*
E-mail: marta.sabou@wu.ac.at
[d] *Institute of Informatics, Technical University of Vienna, Vienna, Austria*
E-mail: allan.hanbury@tuwien.ac.at

Abstract. Large Language Models (LLMs) have demonstrated remarkable capabilities in extracting knowledge and generating new content from a wide range of resources, particularly text-based ones. Beyond unstructured data, LLMs also show strong performance on structured yet semantically rich resources such as ontologies, schemas, and knowledge graphs. However, the direct utilization of large-scale semantic artifacts as input to LLMs is constrained by prompt size and token limits. The state-of-the-art solution to this challenge is the use of RAG systems. In this work, we propose a novel graph-based RAG approach specifically designed for large semantic artifacts. Our method integrates LLM-based Named Entity Recognition and Disambiguation (NERD) and Entity Linking (EL), forming a unified pipeline tailored for semantically complex resources. Within this framework, we implement three use cases that combine LLM reasoning with our RAG system: (i) semantic artifact validation, (ii) information retrieval, and (iii) information model generation. The first two tasks convert Natural Language Queries (NLQs) into executable SPARQL queries, whereas the third populates semantic artifacts based on NLQ-driven instructions. Across all use cases, the system demonstrates strong performance, confirming the effectiveness of the approach. Comparative experiments against two additional RAG baselines further show superior performance in both accuracy and contextual reasoning. Open Platform Communications Unified Architecture (OPC UA) serves as our primary data resource due to its breadth and semantic richness. To demonstrate generalizability, we additionally evaluate the system on the large-scale Smart Applications REFerence (SAREF) ontology, a structurally and semantically distinct artifact. Consistent performance across both resources indicates that the proposed system is not domain-specific and can be reliably applied to diverse semantic datasets.

Keywords: Graph-based-RAG, LLM, SPARQL query generation, Information model generation

*Corresponding author. E-mail: ntufek@gmail.com.

1. Introduction

Over the past two decades, the Semantic Web vision has led to the development of expressive ontologies and large-scale knowledge graphs that enable machines to represent, integrate, and reason over structured knowledge. These semantic artifacts now underpin a wide range of intelligent applications, from biomedical data analysis to recommendation systems and digital twins [1]. As knowledge graphs continue to grow in scale and semantic richness, the ability to interact with them efficiently has become increasingly important.

At the same time, the rise of modern data ecosystems—particularly in enterprise and industrial settings—has resulted in unprecedented amounts of heterogeneous data being generated and digitized. This data must be modeled, exchanged, and interpreted in machine-interpretable formats to ensure interoperability, automation, and scalability across distributed systems. To support these needs, numerous semantic standards have emerged, including the W3C Web of Things (WoT)[1], the Asset Administration Shell (AAS)[2], the Smart Applications REFerence (SAREF) ontology family[3], and the Open Platform Communications Unified Architecture (OPC UA)[4] information modeling framework. These standards provide ontologies and large-scale knowledge graphs that encode domain knowledge in a structured and semantically coherent form. However, interacting with these semantic artifacts remains challenging. Many of them are complex, multi-domain, and large in scale—for example, the ecosystem of OPC UA industrial standard comprises more than 160 domain-specific Companion Specification[5] documents, spanning hundreds to thousands of pages and developed by numerous expert groups and vendors. Moreover, the authoritative descriptions of these artifacts—such as specifications, compliance rules, and modeling constraints—are expressed in free-form natural language, whereas the underlying models are encoded in machine-readable formats such as RDF, TTL, or OWL. This mismatch creates a persistent semantic gap between natural-language domain knowledge and symbolic graph-based representations.

Large language models (LLMs) have recently emerged as powerful tools for bridging this gap. Since the early 2020s, Large Language Models (LLMs) such as Generative Pre-trained Transformer (GPT) have been increasingly used across domains [2], and prior studies have shown that they can validate information models and support knowledge retrieval [1, 3]. Yet, their application to large-scale semantic artifacts is significantly constrained by prompt-size limitations [4]. Industrial knowledge graphs frequently exceed the token limits of current models (e.g., GPT-4: 8K; GPT-4 Turbo/GPT-4o: 32K [5]), making it infeasible to supply entire graphs to an LLM [4]. Beyond this, processing large graphs in context is computationally inefficient. For natural-language documents, Retrieval-Augmented Generation (RAG) [6] addresses similar scalability issues through chunking and embedding-based retrieval [7, 8]. However, RDF/TTL files and knowledge graphs cannot be segmented like text without losing structural integrity or semantic coherence. Identifying meaningful subgraphs, preserving relational context, and aligning them with Natural Language Queries (NLQs) remain unresolved challenges—rendering standard RAG pipelines inadequate for tasks such as validation, information retrieval, question answering, or semantic artifact generation.

These limitations are particularly problematic in industrial and domain-specific contexts, where semantic artifacts such as OPC UA and SAREF play a critical role in information modeling, knowledge integration, and interoperability. To examine this challenge in diverse and realistic settings, this study considers two representative artifacts: (i) the OPC UA standard, a complex multi-domain ecosystem widely adopted in industry; and (ii) the SAREF ontology, a modular semantic framework used across smart domains such as energy, buildings, and manufacturing. To address these issues, this study investigates how NLQs can be used to interact with large-scale semantic artifacts across three representative use cases (UC) covering two fundamental task families: *querying* and *instantiation*. The first family focuses on deriving machine-executable representations from NLQs and executing them over semantic artifacts:

[1] https://www.w3.org/TR/wot-thing-description11/
[2] https://www.plattform-i40.de
[3] https://saref.etsi.org
[4] https://opcfoundation.org/about/opc-technologies/opc-ua/
[5] https://opcfoundation.org/about/opc-technologies/opc-ua/ua-companion-specifications/

UC1 Validation: verifying whether a semantic artifact satisfies natural-language rules or constraints. In the OPC UA ecosystem, validating whether a semantic artifact satisfies natural-language rules or constraints is especially challenging because Companion Specifications contain highly technical yet inherently ambiguous textual rules. Manually checking whether these rules are correctly reflected in large, complex information models is time-consuming, error-prone, and often infeasible at scale. Therefore, the validation use case is critical both for industrial reliability and for our work—since without robust rule validation, any automation, integration, or downstream RAG process is fundamentally undermined.

UC2 Information Retrieval: performing industrial question answering by returning graph-based information relevant to an NLQ. In both OPC UA and SAREF ecosystems, information retrieval is challenging because users often ask natural-language questions that must be mapped to deeply structured, graph-based knowledge. These models contain complex hierarchies, cross-references, and domain-specific semantics, making it difficult for users to directly locate the information they need. Effective NLQ-to-graph retrieval is therefore crucial to help practitioners quickly access the correct concepts, relationships, and constraints without manually navigating large industrial ontologies.

The second family focuses on the creation or extension of semantic artifacts:

UC3 Information Model Generation: populating or extending an existing semantic artifact based on natural-language requirements. Generating or extending information models is inherently complex, requiring strict adherence to specification rules, precise NodeClass usage, and consistent semantic structure. Current OPC UA tooling offers only partial assistance—typically via manual editors or schema-guided interfaces—and none support direct natural-language input for creating compliant nodes. This makes NLQ-driven model generation an unmet and technically demanding challenge in industrial settings.

Although LLMs show strong potential for these tasks [3], a fundamental limitation persists: enabling NLQ-driven reasoning over *large* semantic artifacts whose size exceeds the prompt and token constraints of current models. Existing LLMs cannot ingest entire industrial knowledge graphs end-to-end, and traditional embedding-based retrieval approaches—such as Word2Vec [9] and Node2Vec [10]—are insufficient for supporting the complex reasoning needs of large-scale semantic resources. In particular, current methods exhibit three major gaps: (i) the absence of effective mechanisms for handling large industrial semantic artifacts; (ii) the lack of comprehensive solutions that support both querying and instantiation from NLQ inputs; and (iii) the lack of generalizable NLQ-driven techniques that can operate robustly across diverse semantic artifacts.

These gaps motivate the following research questions:

RQ1 (Design) What kind of RAG architecture is required for effective NLQ–KG interaction at scale?

RQ2 (Application) How can such an architecture support validation, information retrieval, and information model generation?

RQ3 (Generalization) Can the approach generalize across heterogeneous semantic artifacts such as OPC UA Robotics, PackML, and SAREF?

Proposed Approach. To answer these questions, we introduce a novel graph-based RAG architecture specifically designed for large-scale industrial knowledge graphs. The approach integrates LLM-based named-entity and relation detection (NERD), entity linking (EL), graph modularization, targeted subgraph retrieval, and LLM-driven reasoning for query generation, validation, and model instantiation. The system is evaluated across three representative semantic artifacts: OPC UA Robotics, OPC UA PackML, and a combined large-scale SAREF knowledge graph. We further benchmark our approach against two state-of-the-art RAG baselines to demonstrate its superiority in accuracy and contextual reasoning.

Experiments. To demonstrate the broad applicability and practical value of the proposed approach, we identify three key use cases that collectively validate its versatility and effectiveness across realistic industrial scenarios. These use cases are designed to reflect critical needs in industrial settings and evaluate the system from multiple perspectives: validation, retrieval, and generation.

1. *Validation of Large Knowledge Graphs (KGs) using NLQs:* This use case examines the system's ability to verify the correctness and consistency of large-scale knowledge graphs through natural language queries. Experiments were conducted on two semantic artifacts from the OPC UA ecosystem: *Test Case 1: OPC*

UA Robotics and *Test Case 2: OPC UA PackML*. The evaluation focuses on validating the compliance rules extracted from their respective Companion Specification documents.

2. *Information Retrieval and Question Answering over Large KGs:* This use case evaluates the system's capability to retrieve relevant information and answer natural-language questions directly from large knowledge graphs, demonstrating its robustness in handling complex and diverse query structures. The evaluation consists of three test cases:
   - *Test Case 1:* Executing the proposed system alongside three baselines on the OPC UA Robotics information model using a predefined set of NLQs.
   - *Test Case 2:* Applying the system and baseline RAG algorithms to SAREF, a substantially different industrial semantic artifact, to assess generalizability using SAREF competency questions.
   - *Test Case 3:* Evaluating the system on the OPC UA Robotics semantic artifact with increasingly varied NLQs to analyze robustness when queries deviate from exact entity mentions toward more abstract or paraphrased formulations.
3. *Creation of New Semantic Artifacts from Large-Scale KGs and Natural Language Requirements:* This use case demonstrates how the proposed approach enables the generation of new semantic artifacts by integrating existing large-scale knowledge graphs with requirements expressed in natural-language form. This experiment was carried out using the OPC UA Robotics information model as the base artifact (*Test Case 1*).

Contributions. This study offers the following key contributions:

CONT1 We propose an end-to-end RAG framework tailored for large-scale knowledge graphs, integrating LLM-based NERD, EL, and subgraph extraction components, addressing **RQ1**.

CONT2 We evaluate the system across three complementary use cases—validation, information retrieval, and information model generation—addressing **RQ2**.

CONT3 We demonstrate generalizability across multiple OPC UA domains and the SAREF ontology, addressing **RQ3**.

CONT4 We empirically show that the proposed method outperforms baseline RAG approaches in accuracy and contextual reasoning, supporting **RQ2** and **RQ3**.

The remainder of this paper is organized as follows. Section 2 provides background and motivating use cases. Section 3 presents our graph-based RAG methodology. Sections 4–6 detail the pipeline components. Section 7 reports the experiments and results, followed by discussion in Section 8. Section 9 reviews related work. Section 10 concludes and outlines future directions.

2. Background and Motivation

This section provides background information on the semantic artifacts studied in this work. In addition, the use cases addressed in the study are described in detail.

2.1. Industrial Semantic Artifacts

The main industrial use case in this work relies on the OPC UA standard, which provides a semantically rich and widely adopted framework for modeling industrial systems. We additionally incorporate the SAREF ontology to represent IoT concepts using a lightweight, ontology-driven approach. Using both OPC UA and SAREF enables us to evaluate our methods across two distinct semantic modeling paradigms.

2.1.1. OPC UA Standard

The *OPC Foundation*, established in 1996, plays a central role in enabling interoperability across industrial automation systems. Its flagship technology, the *OPC Unified Architecture (OPC UA)*, is a platform-independent, service-oriented framework that unifies communication, data exchange, and information modeling. Unlike earlier OPC technologies, OPC UA integrates a robust security model with an extensible address space and a semantically consistent information representation, which makes it a foundational technology for Industry 4.0 and digitally connected manufacturing ecosystems.

At the core of OPC UA lies a comprehensive set of *Core Specifications* that define the abstract information model, reference types, communication protocols, security mechanisms, and serialization formats. These core standards provide a rigorous semantic foundation that can represent any industrial asset or process in a structured and machine-interpretable manner. Building on this foundation, the OPC Foundation and various industry consortia publish *Companion Specifications*, which extend OPC UA with domain-specific concepts for verticals such as robotics, machine tools, packaging, process automation, and energy systems. Companion Specifications define standardized object types, variables, state machines, capabilities, and constraints, ensuring semantic consistency and interoperability across vendors within a given domain.

OPC UA has become increasingly important as modern industrial systems grow in scale, complexity, and heterogeneity. Applications such as digital twins, predictive maintenance platforms, cross-vendor control solutions, and industrial knowledge graphs rely on the availability of semantically precise and interoperable models. Companion Specifications serve as authoritative references for these models, enabling consistent interpretation of device structures, operational behavior, diagnostics, and domain-level semantics. As industrial ecosystems evolve, OPC UA remains one of the few technologies capable of providing both rich semantic modeling and secure, real-time data exchange, making it indispensable to next-generation industrial automation.

Despite its significance, the rapid expansion of OPC UA Companion Specifications introduces substantial challenges. New specifications are published frequently, and existing ones evolve continuously, resulting in hundreds of pages of complex, human-readable documentation. Each specification contributes new object types, reference structures, constraints, and behavioral rules that must be carefully translated into nodeset files and validated against the OPC UA information model. This process is highly error-prone and increasingly difficult to scale as specifications grow in size and interconnectedness. Furthermore, existing validation tools primarily support syntactic checks and provide limited capabilities for verifying the semantic rules defined in Companion Specifications, creating a significant bottleneck for developers and integrators working with large-scale OPC UA information models.

2.1.2. SAREF Ontology

SAREF ontology[6] is used in this work as a secondary data resource, providing an independent and domain-agnostic benchmark for evaluating the generalizability of our approach beyond the primary industrial knowledge graphs. SAREF is an ontology developed by ETSI to enable semantic interoperability across IoT ecosystems by defining common concepts such as devices, sensors, actuators, properties, measurements, and units, together with their relationships. It offers a modular and extensible structure—including sector-specific extensions such as SAREF4ENER, SAREF4BLDG, and SAREF4INMA—which allows heterogeneous IoT systems to exchange data in a consistent, machine-interpretable manner. By incorporating SAREF as an additional data source, we are able to test whether the proposed methods remain effective outside the industrial OPC UA context and operate reliably across broader IoT-oriented semantic models. For the SAREF ontology, we rely on the specification published by ETSI.

2.2. Problem Definition

The problem addressed in this work can be formalized from an input–output perspective, comprising two complementary tasks. The first is query generation, where natural-language inputs are mapped to executable SPARQL queries, and the second is information model generation, where natural-language requirements are transformed into instantiated RDF knowledge graphs.

We consider a semantic artifact represented as an RDF graph G:

$$G = (T), \qquad T \subseteq U \times U \times (U \cup L), \tag{1}$$

where T is the set of RDF triples, with each triple (s, p, o) consisting of a subject $s \in U$, a predicate $p \in U$, and an object $o \in (U \cup L)$. Let $\mathcal{L}$ denote the space of natural-language expressions, and $\mathcal{S}$ the space of executable SPARQL queries.

[6] https://SAREF.etsi.org/

2.2.1. NLQ-to-SPARQL Generation

Given a natural-language query q:

$$q \in \mathcal{L}, \tag{2}$$

the goal is to generate a SPARQL query s:

$$s \in \mathcal{S}, \tag{3}$$

such that executing s over G returns the information requested by q. Let $\mathrm{Exec}(s, G)$ denote the evaluation of s on G, and let $\mathrm{Ans}(q)$ denote the expected answer to the NLQ. The NLQ-to-SPARQL problem and evaluation is formalized as:

$$s^*(q, G) = \arg\max_{s \in \mathcal{S}} \mathrm{Sim}\big(\mathrm{Exec}(s, G), \mathrm{Ans}(q)\big). \tag{4}$$

In our setting, s may be restricted to ASK queries (for validation) or SELECT queries (for information retrieval).

2.2.2. NLQ-to-Instantiated Knowledge Graph

Given a base RDF graph:

$$G_{\text{base}} = (T_{\text{base}}), \tag{5}$$

and a natural-language requirement:

$$q_{\text{im}} \in \mathcal{L}, \tag{6}$$

the objective is to construct an extended RDF graph:

$$G' = (T'), \qquad T_{\text{base}} \subseteq T', \tag{7}$$

that satisfies the modeling constraints expressed in the requirement. Let $\mathcal{C}(q_{\text{im}})$ denote the set of constraints extracted from the requirement. The instantiation problem is therefore defined as finding a graph G' such that:

$$G' \models \mathcal{C}(q_{\text{im}}), \qquad T_{\text{base}} \subseteq T'. \tag{8}$$

These definitions characterize the two problem spaces addressed in this work, independently of any specific solution strategy.

2.3. Use Cases

In this study, the RAG implementation was evaluated across three different use cases: validation, information retrieval, and information model generation.

2.3.1. Validation

In the validation use case, the objective is to identify the rules defined in the relevant companion specification documents and determine whether these rules are correctly implemented within a given nodeset file. The ultimate goal is to automate the entire validation workflow. To achieve this, the process begins with extracting sentences from complex, multi-modal PDF documents—which often contain a mixture of text, tables, and figures—and detecting those sentences that express modeling rules. Among these, it is crucial to distinguish information model rule sentences, i.e., those that describe structural or semantic constraints pertaining to the OPC UA information model. These processes are described in our previous studies [11][3].

As it is demonstrated in Figure 1, the rule sentences serve as NLQs that must be interpreted and mapped to their corresponding SPARQL Protocol and RDF Query Language (SPARQL) queries. Once converted, these SPARQL queries are executed over large RDF graphs, which are automatically generated by converting OPC UA nodeset files (e.g. OPC UA Robotics Noteset file [12]) into a semantic RDF representation. This pipeline enables automated validation of whether the semantic artifacts in the nodeset conform to the intended structure and constraints described in the companion documents.

The main challenge in this process is the automatic translation of NLQs into SPARQL (ASK-type) queries that can be executed over large knowledge graphs generated from OPC UA nodeset files. These graphs are semantically rich and structurally complex, making it difficult to match informal rule descriptions with formal query patterns. Bridging this gap requires accurate semantic parsing and alignment between natural language and the underlying information model.

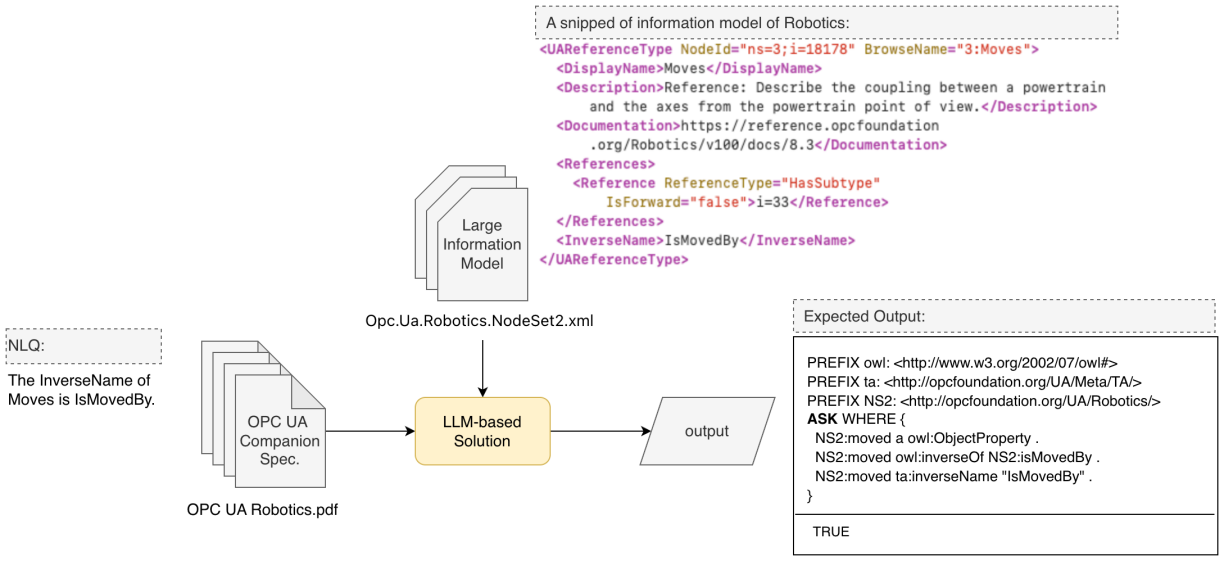

Fig. 1. Example input-output for validation use case for OPC UA Robotics

2.3.2. *Information Retrieval*

In the second use case, the focus is on the generation of SPARQL SELECT queries from user-provided NLQs as shown in Figure 2. These queries are formulated intuitively by users who seek to retrieve specific information from large OPC UA knowledge graphs. The main challenge lies in the system's ability to accurately interpret the user's intent and translate it into a correct and meaningful SPARQL query. This requires robust semantic relations between natural language expressions and the formal structure of the OPC UA information model. Additionally, the solution must be capable of performing efficient query execution over large, semantically complex knowledge graphs generated from nodeset files.

2.3.3. *Information Model Generation*

The third use case differs from the previous two in that it does not focus on generating SPARQL queries, but rather on automatically generating or modifying OPC UA information models based on user-provided requirements. Specifically, the goal is to allow users to extend existing nodeset files by adding new nodes, attributes, or relationships—or by manipulating existing model elements—according to their specific application needs. This process is driven by NLQs that intuitively express the user's modeling requirements. By leveraging large OPC UA knowledge graphs, the system interprets these NLQ s and generates corresponding structural modifications to the information model. Ultimately, this use case enables intuitive and automated information model generation, supporting flexible, user-driven model customization while ensuring semantic consistency and compliance with OPC UA standards.

2.4. *RAG*

RAG has emerged as a standard approach for leveraging large-scale resources in LLMs. This approach is necessary because current LLM solutions are neither efficient nor feasible for handling extensive text-based documents in tasks such as information extraction and question answering. One primary limitation is that as the length of the text increases, the model struggles to maintain contextual coherence between the beginning and later sections. Furthermore, despite recent increases in prompt size, LLMs still have inherent constraints on the amount of input they can process at once. Additionally, utilizing LLMs for such large-scale tasks is computationally expensive, requiring significant time, processing power, and financial resources.

Principles of RAG Mechanism: As demonstrated in Figure 3, output generation using big data resources and textual input (NLQ) consists of three phases: the retrieval phase, the augmentation phase, and the generation phase.

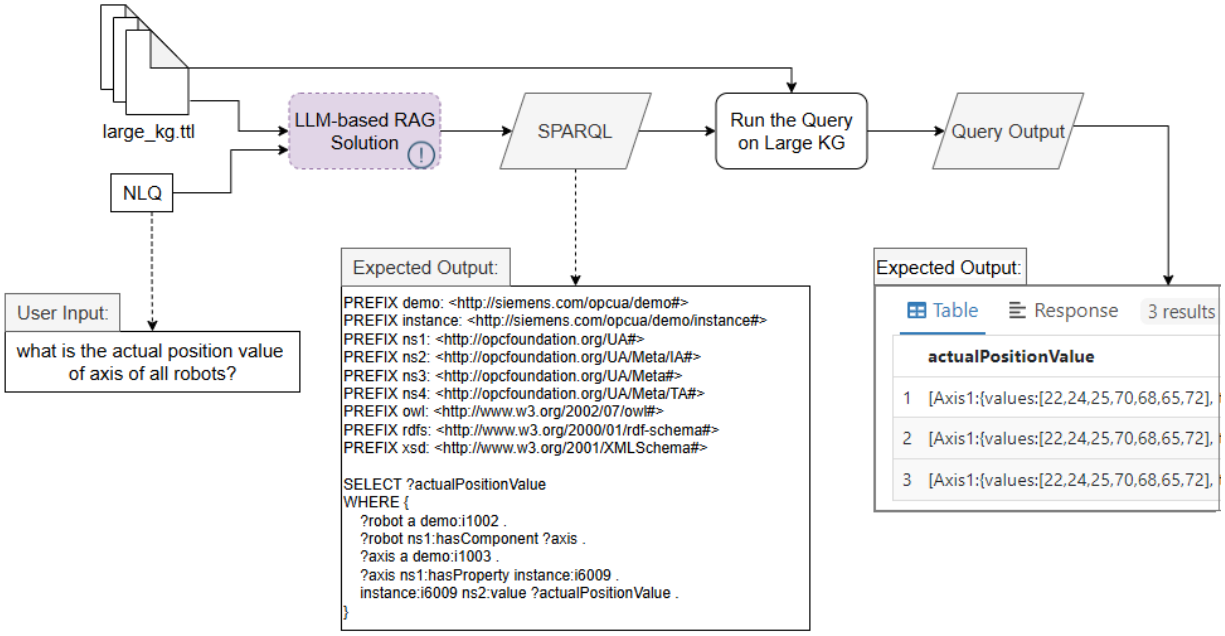

Fig. 2. Example input-output for information retrieval use case

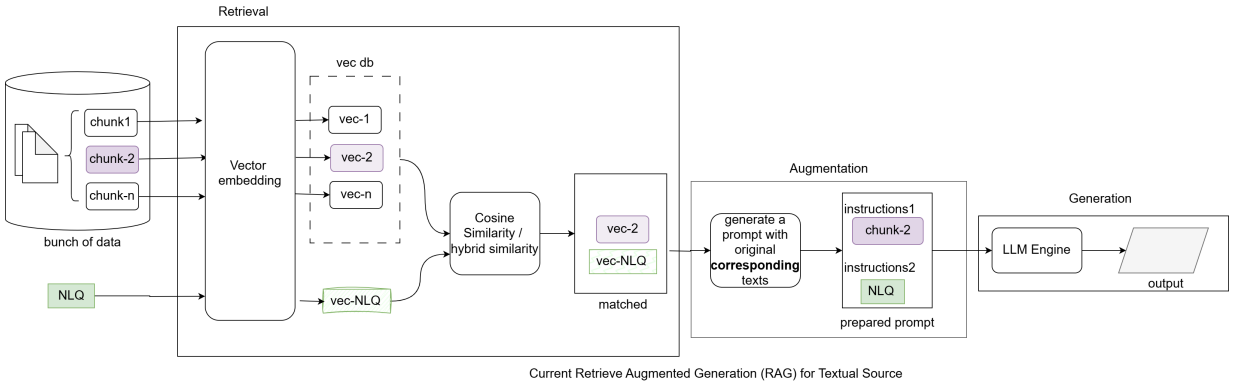

Fig. 3. Standard RAG approach for the textual data

2.4.1. *Retrieval Phase*

The retrieval stage serves as the foundation for enhancing the generative process by incorporating **external knowledge sources**. When an input query *NLQ* is received, it is first transformed into a dense vector representation using an embedding model such as *BERT [13], RoBERTa [14], or OpenAI embedding models*. This vector representation facilitates the identification of the most semantically relevant *chunk*, represented as:

$$C = \{c_1, c_2, \ldots, c_n\} \tag{9}$$

where each c_i corresponds to a chunk (the text segments partitioned into fixed sizes, e.g., paragraphs) from a pre-existing knowledge base. The retrieval process is typically conducted using approximate nearest neighbor (ANN) [15] search techniques such as FAISS [16], allowing for efficient and scalable document retrieval from large datasets [17]. The primary objective of this phase is to enrich the language model's output by providing it with relevant, up-to-date, and domain-specific information.

2.4.2. Augmentation Phase

Once the retrieval process is completed, the gathered documents serve as contextual supplements for the language model. The augmentation step involves integrating the retrieved content into the input prompt, effectively expanding the context window available to the model. This step is crucial as it not only ensures factual accuracy but also reduces the likelihood of hallucinations, where the model might otherwise generate misleading or false information.

Various methods can be used to structure the augmented input, such as:

- Specifying the instruction set,
- Concatenating the retrieved text,
- Ranking passages based on relevance,
- Applying weighted fusion techniques.

By augmenting the model with non-parametric, dynamically retrieved knowledge, it becomes more adaptable and better suited for real-time, context-aware responses.

2.4.3. Generation Phase

In the final stage, the pre-trained LLMs generates a response that is informed by both the original query and the retrieved documents. Models such as GPT-4 [18], T5 [19], or BART [20] employ autoregressive decoding strategies, including greedy search, beam search, or nucleus sampling, to produce fluent and coherent textual outputs.

Since the model's response is influenced by external knowledge sources via a retrieval step, it tends to yield higher factual accuracy and improved contextual relevance compared with purely generative approaches. Furthermore, by incorporating retrieval, RAG pipelines enhance interpretability, as responses can be traced back to specific reference documents. This makes the approach especially advantageous in high-stakes domains such as fail-safe systems, medical diagnosis, legal analysis and academic research.

3. Purposed Approach: A Graph-based-RAG

We have extensive industrial information models for OPC UA available in XML format, which we aim to leverage for validation, question-answering, and information model generation. Based on our previous work [3], it has been demonstrated that LLMs can effectively generate SPARQL queries by leveraging semantic artifacts and NLQs. Building on this approach, a structured prompt—comprising a combination of predefined instructions, semantic artifacts (e.g., knowledge graphs), and user-provided textual input (e.g., NLQ s)—is formulated and submitted to the LLM engine. The model then processes this input to generate the corresponding SPARQL query for the validation use case. The overall process is depicted in Figure 4.

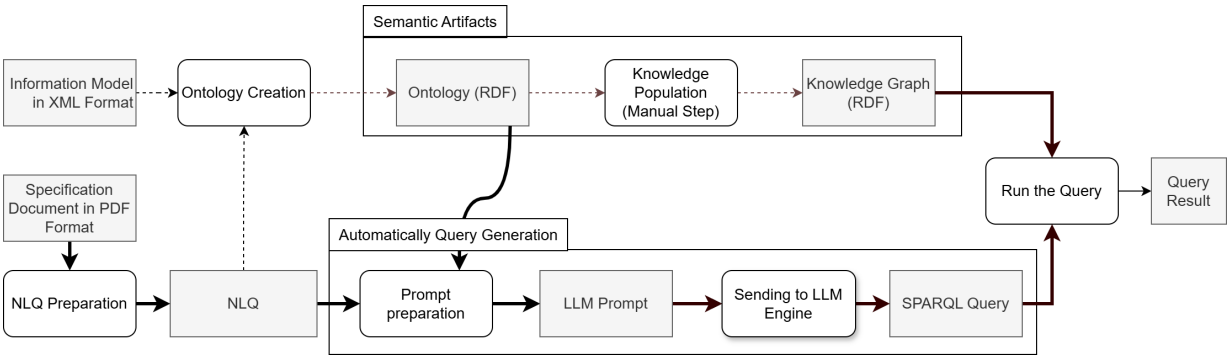

Fig. 4. End-to-end design for validation use case with small-size semantic artifacts [3]

Building on this approach, compliance rule sentences are extracted from the specification document and used as NLQs. Simultaneously, information models converted into Turtle (TTL) format, where they are enriched with instance data. A structured prompt is then formulated to generate a SPARQL query, incorporating both the NLQ and a limited subset of the ontology to optimize query generation. Once the prompt is prepared, the GPT-4o API

is invoked to generate the SPARQL query, which is subsequently executed on the knowledge graph. The results indicate that this method performs effectively, demonstrating the feasibility of using LLMs for automated SPARQL generation and execution in a validation context [3].

Although this approach represents a significant step toward integrating KGs and LLMs, the experiments in [3] were conducted using information models of limited size. However, in real-world scenarios, we often deal with significantly larger semantic resources. For example, the OPC UA Robotics nodeset.ttl file (after conversion into RDF format) contains approximately 22,000 triples, exceeding the token limits of all existing LLM engines. Therefore, while the proposed setup has proven effective, a scalable solution is required to extend its applicability to large-scale knowledge graphs.

Unlike conventional RAG approaches, the dataset cannot be trivially divided into smaller chunks with corresponding embeddings due to the high degree of interconnectivity among entities and attributes within the knowledge graphs. This structural complexity prevents standard chunk-based retrieval techniques from preserving the semantic relationships required for accurate query generation and reasoning. To address this challenge, we propose a novel solution for handling large-scale knowledge graphs efficiently. This end-to-end approach extends beyond the validation use case, making it applicable to a broader range of knowledge-driven tasks. By leveraging this framework, LLMs can be effectively utilized for various applications that require structured knowledge retrieval, reasoning, and validation. The proposed solution is illustrated in Figure 5.

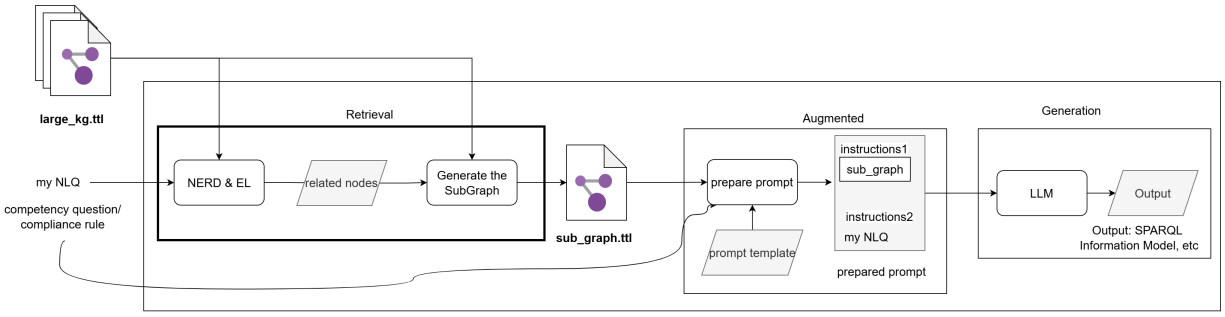

Fig. 5. The proposed RAG solution for the KGs

Figure 5 introduces a novel, comprehensive step called **Retrieval**, which plays a pivotal role in the proposed approach. This step enables the extraction of a subgraph based on the user's NLQ, ensuring semantics are preserved and only the most relevant portion of the knowledge graph is utilized. By retrieving a focused subgraph instead of processing the entire large-scale knowledge graph—often consisting of thousands of triples and exceeding token limitations—this method significantly enhances efficiency. The extracted subgraph is then incorporated into the LLM prompt, making the approach scalable and suitable for various knowledge-driven tasks, including query generation, validation, and reasoning. It can be seen that it is quite different than the classical RAG approaches described in Section 2.

The algorithm of our RAG approach is as follows: First, the domain-specific entities are extracted from the given NLQ text. These entities are then mapped to their corresponding nodes within the information model, which represents a large-scale dataset. This process, known as Named-entity Recognition and Disambiguation (NERD) and Entity Linking (EL), is also performed using LLMs. Subsequently, a subgraph is constructed as described in the first in the below section, thereby completing the retrieval process. The extracted subgraph, along with the NLQ, is then integrated into a predefined prompt template and submitted to the LLM engine. The LLM subsequently generates the required output. In our use cases, this output typically consists of either a SPARQL query or a newly generated set of nodes.

4. First Step: Retrieval

The retrieval pipeline consists of two formal components: (i) Named Entity Recognition and Disambiguation (NERD) and Entity Linking (EL), and (ii) Subgraph Expansion.

4.1. NERD and EL

The initial phase of the retrieval process entails the recognition of named entities within the user-provided NLQ, followed by their alignment with corresponding node URIs in the underlying KG.

$$\text{NERD} : \mathcal{L} \rightarrow 2^V \tag{10}$$

For a given query $q \in \mathcal{L}$, NERD returns a set of candidate entity labels:

$$E_q = \text{NERD}(q) \tag{11}$$

The objective is to map detected mentions to their corresponding URIs (U) in the KG:

$$\text{EL} : E_q \rightarrow U \tag{12}$$

For each mention $e \in E_q$, EL resolves a unique URI:

$$\text{EL}(e) = u_e \tag{13}$$

Therefore the proposed and implemented solution follows the architecture illustrated in Figure 6.

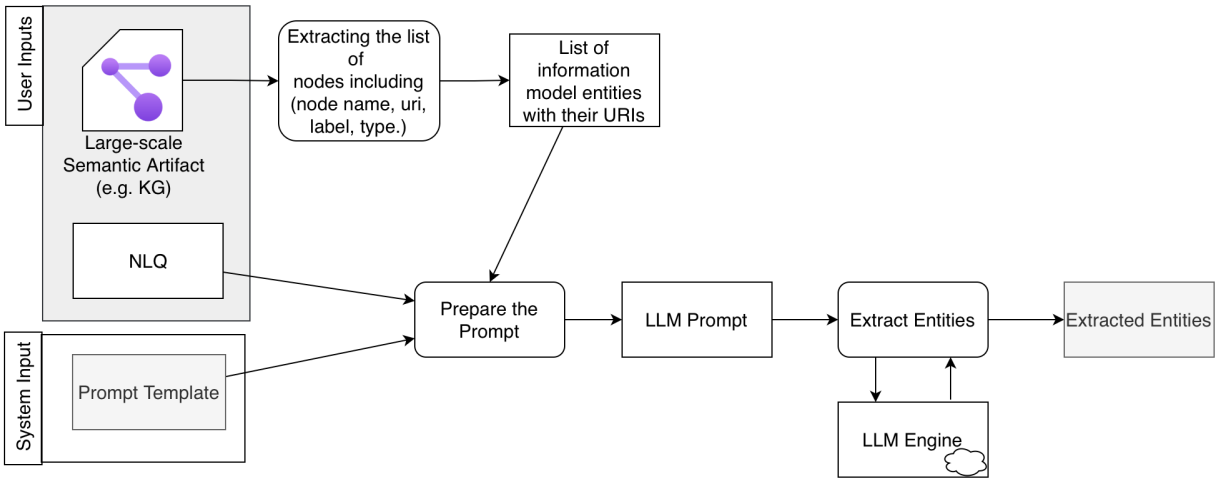

Fig. 6. The system design of NERD and EL

In this context, the primary inputs are a large-scale KG and an NLQ, with the objective of performing NERD and EL.

However, several challenges must be carefully considered and addressed when formulating an effective solution:

- The entities are domain-specific and distinct from common dictionary words, making them unique (e.g. "ControllerIdentifier_AR").
- Many entities are multi-word expressions rather than individual terms (e.g "MotionDeviceCategory").
- Although users typically mention entities using their human-readable labels, the underlying node URIs—which carry the actual semantic identity in the graph—are often completely different. Therefore, matching user input to the correct node requires resolving this label–URI discrepancy.
- Node URIs often consist of non-descriptive alphanumeric combinations, making them difficult to interpret (e.g. "i5001").
- Users tend to provide their input intuitively, without strictly adhering to predefined terminology.

An example illustrating some of these challenges is presented in Table 1. The table presents three different NLQ s, each reflecting intuitive variations in expressing the same entities. These examples consider not only the direct usage of entity labels as they appear in the knowledge graph but also variations in capitalization, plural forms, typos, and compound vs. separated word usage. Moreover, in the second and third examples, the intuitive phrasing for *ActualPosition* gradually changes, first becoming *actual position*, then *current position* and later evolving into

Table 1
Examples of differently written user inputs referring to the same entity for NERD and EL

Input: Semantically Equivalent NLQ s	**Expected Output:**	
	Matched Entity Labels	**Node URIs**
What is the actual position of Axis of all robots on the plant?	actual position ->ActualPosition	http://siemens.com/opcua/Robotics/ActualPosition
	Axis ->axis	https://opcfoundation/Robotics/i1003
	robots ->Robot	http://siemens.com/opcua/demo/i1002
What is the current position of axes of all robots?	current position ->ActualPosition	http://siemens.com/opcua/Robotics/ActualPosition
	axes ->Axis	https://opcfoundation/Robotics/i1003
	robots ->Robot	http://siemens.com/opcua/demo/i1002
What are precise coordinates of axis of each robot?	precise coordinates ->ActualPosition	http://siemens.com/opcua/Robotics/ActualPosition
	axis ->Axis	https://opcfoundation/Robotics/i1003
	robot ->Robot	http://siemens.com/opcua/demo/i1002

Precise position. The extent to which the system can handle such intuitive variations depends on the system's requirements and overall quality.

Traditional NERD and Entity Linking (EL) approaches, as well as pretrained models, fail to achieve sufficient accuracy in resolving these challenges. To address this, we leverage LLMs for NERD and EL. However, before utilizing LLMs, a preprocessing step is necessary.

4.1.1. Preprocessing

To enable effective use within our system, knowledge graphs are first preprocessed. The converted information model in RDF format is complex which is challenging for an LLM to process and generate SPARQL queries from it. Therefore, we simplified some parts of the knowledge graph such as: OWL axioms in the KG are replaced with simplified triples.

Next, statements representing redundant relations are pruned. These are considered redundant if the same relationship between nodes is already expressed through other statements. After this pruning step, a mapping table is generated to serve as a lookup table for entity linking.

This mapping extracts all nodes from the KG and structures them according to several attributes: `subject_uri`, `subject_name`, `nodeid`, `browseName`, `label`, and `entity_types`. The information is saved in CSV format, from which a list of human-readable labels is derived to support the LLM-based matching process. A snippet of this CSV is shown in Table 2.

Table 2
An example row of mapping CSV file

Subject URI	**Subject name**	**Node ID**	**Browse name**	**Label**	**Entity type**
http://opcfoundation.org /UA/DI/i15088	i15088	http://opcfoundation.org/ UA/DI/i15088	http://opcfoundation.org/ UA/DI/Model	Model	http://opcfoundation.org/ UA/PropertyType;
					http://www.w3.org/2002/07/ owl#NamedIndividual;
					http://opcfoundation.org/ UA/Meta/Variable

The corresponding definition of the first node in the the Table 2 whose subject_uri is , is shown Turtle (TTL) format as below:

An example node of OPC UA Robotics KG in Turtle format

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix NS0: <http://opcfoundation.org/UA/> .
@prefix NS1: <http://opcfoundation.org/UA/DI/> .
@prefix me: <http://opcfoundation.org/UA/Meta/> .
@prefix ia: <http://opcfoundation.org/UA/Meta/IA/> .

NS1:i15088 a me:Variable, owl:NamedIndividual, NS0:PropertyType;
  ia:dataType NS0:LocalizedText;
  ia:nodeId "http://opcfoundation.org/UA/DI/i15088"^^xsd:anyURI;
  ia:browseName "http://opcfoundation.org/UA/DI/Model"^^xsd:anyURI;
  ia:accessRestrictions "0"^^xsd:unsignedShort
  rdfs:label "Model";
  rdfs:isDefinedBy NS1:i15063;
  ia:isVariable "true"^^xsd:boolean;
  ia:userAccessLevel "1"^^xsd:unsignedByte;
  ...
```

4.1.2. Prompt Preparation

The aim of this step is to identify words and phrases within the NLQ that exhibit semantic and syntactic similarity to the labels in the prepared list. While LLMs improve the feasibility of this task, the potential for hallucination—the generation of incorrect or misleading outputs—remains a concern. To mitigate this issue, a prompt template is constructed, incorporating the NLQ and the extracted label list from the KG. This structured template as below ensures that the LLM receives well-defined input, thereby enhancing accuracy in entity identification.

Prompt template for entity linking

Task: Identify and match the named entities from a given text and a given list. A word or phrase must be associated with a single entity in the list if there is a match, but not every word or phrase has to have a corresponding item in the list. Find exact matches, semantic matches, or syntactic similar matches.
Given Text: {NLQ}
Given List: {label_list}
Output: The result should be a JSON array of dictionaries, each with keys "matched_text" and "entity".
Example Output:

```
[{ "matched_text": "original part of the text", "entity": "entity from label list"}, ... ]
```

4.2. Subgraph Extraction

After the extraction of entities from a NLQ and the retrieval of their corresponding node URIs, the next essential step is to construct a meaningful subgraph. In this context, a "meaningful" subgraph refers to one that contains enough information to generate a SPARQL query. To achieve this, the subgraph must align with the constraints and semantics of the NLQ. Simply extracting the nodes corresponding to the mentioned entities is insufficient, as it is often necessary to include intermediary nodes and relationships that are implied but not explicitly stated in the NLQ. Therefore, we developed a subgraph extraction approach using graph modularization techniques, which retrieves all the necessary context information to answer an NLQ.

The approach is also presented in Algorithm 1.

Our subgraph extraction method introduces novel contributions within the context of Graph-based-RAG applications. As described in Algorithm 1, the operational workflow proceeds as follows:

Algorithm 1 Subgraph construction for RAG

```
Extract entities from the NLQ along with their corresponding subject URIs.
Determine the shortest paths between these nodes.
Include the intermediate nodes present within these paths in the list.
Retrieve the definitions of the relationships that exist between the included nodes.
if no path can be found then
    Include only the directly linked nodes.
end if
for each node in the path do
    if the node is a class then
        Include its instances in the subgraph.
    else if the node is an instance then
        Include its corresponding class in the subgraph.
    end if
end for
Incorporate all "hasComponent" and "hasProperty" relationships associated with the included nodes.
Generate a new RDF file representing the refined subgraph using the previously established prefixes and considering OWL axioms.
```

Entities mentioned in the NLQ are first identified and mapped to their corresponding node URIs. Among these entities, the shortest connecting path is determined using Dijkstra's algorithm, and a single optimal path is selected. A crucial aspect of the approach is ensuring that all given entity nodes are included in the path, along with all intermediate nodes that link them.

Notably, previous Graph-based-RAG approaches based on graph embeddings often failed to reliably recover the intermediate nodes, which limited the contextual completeness of the retrieved subgraphs. To address this limitation, our method explicitly constructs the path through symbolic reasoning and then expands the subgraph according to the following expansion-rules:

1. If a node is a class, include its instances.
2. If a node is an instance, include its classes.
3. Include nodes connected via `hasComponent` and `hasProperty` relations.
4. If a node is a class, also include its parent classes.

The most critical constraint in the expansion step is the token limit imposed by the target language model deployment. In particular, due to the relatively low token threshold of models such as GPT-4o, it is often not feasible to apply all expansion rules fully. In such cases, the method continues with a partially expanded subgraph, where expansion rules are applied in a prioritized order (as listed above). When the token budget becomes restrictive, the system first drops the lowest-priority rule (i.e., expansion-rule 4), and then proceeds backwards through the remaining rules until the token limit can be satisfied.

The formal representation of the retrieval phase is as follows. Given the linked URIs:

$$U_q = \{u_1, u_2, \ldots, u_n\}, \tag{14}$$

the first goal is to find a shortest path containing all nodes in U_q:

$$P^* = \arg\min_{P \supseteq U_q} |P|. \tag{15}$$

The induced subgraph is:

$$G_P = G[P^*]. \tag{16}$$

The shortest-path subgraph G_P is expanded with additional context according to symbolic rules. *Class-to-instance expansion:*

$$v \in \text{Class} \;\Rightarrow\; \text{Instances}(v) \subseteq G'. \tag{17}$$

Instance-to-class expansion:

$$v \in \text{Instance} \;\Rightarrow\; \text{Type}(v) \subseteq G'. \tag{18}$$

Component and property relations:

$$(v, p, o) \in T \ \wedge \ p \in \{\text{hasComponent}, \text{hasProperty}\} \ \Rightarrow \ o \in G'. \tag{19}$$

Optional superclass expansion:

$$v \in \text{Class} \ \Rightarrow \ \text{SuperClass}(v) \subseteq G'. \tag{20}$$

The final retrieved subgraph used for query generation is:

$$G' = G_P \cup \text{ExpansionRules}(G_P). \tag{21}$$

5. Second Step: Augmentation

After retrieving the subgraph, the next step involves preparing the prompt for the LLMs to generate a response. This process requires the extracted subgraph, the identified and matched entities, and the original NLQ. These elements are then used to finalize the prompt, which is constructed based on a predefined template containing relevant instructions.

The prompt template for SPARQL generation in the Validation use case is prepared as follows. While other prompts could potentially fulfill our validation requirements, this particular template has been formalized based on extensive experiments and empirical observations.

Prompt template for SPARQL generation in validation use-case:

By using this ontology/knowledge graph:
{SCHEMA}
First, generate the path and relations.
Then, generate a "SPARQL query" with all needed prefixes, including `xsd`.
Use those paths and relations to validate the Natural language sentence,
Use the {QT} keyword of SPARQL with only sufficient properties.
Please consider the instance of the classes as well.
Use also this entity, node URI matches:
{ENTITIES}
Natural language sentence:
{NLQ}

The parameters are defined as strings enclosed in curly brackets, such as `{SCHEMA}`, `{QT}`, `{ENTITIES}`, and `{NLQ}`. These placeholders are programmatically populated to finalize the prompt.

- **`{SCHEMA}`** is replaced with the extracted subgraph.
- **`{QT}`** represents the type of the generated query, such as `ASK` or `SELECT`. In the case of query generation, it is set to `ASK`, whereas for the *Information Retrieval* use case, it is replaced with `SELECT`.
- **`{ENTITIES}`** contains the matched entities, which are incorporated to facilitate query generation.
- **`{NLQ}`** represents the main user input, serving as the NLQ to be processed.

This structured approach is designed to ensure consistency and automation in prompt generation for different use cases.

In addition to its application in query generation, an other prompt template has been specifically designed for the purpose of generating an information model. This template below serves as a structured framework to guide the process of constructing and refining the information model based on predefined parameters and user requirements.

Prompt template for XML generation in information model generation use-case:

This is given OPC UA information model in xml format:
{XML}
Natural language sentence:
{NLQ}
Update the xml acordingly. and return only the modified and the additional nodes in xml format.
Don't forget to add the root node definition of the given xml to this new generated xml.
Keep just <?xml ... > node and <UANodeSet ... > exactly as xml.

The parameters, enclosed in curly brackets (e.g., XML, NLQ), serve as placeholders that are programmatically populated to complete the prompt. XML represents the XML-formatted subgraph generated from NLQ-type-1 (Section 2.2.3), while NLQ-type-2 defines the modifications applied to the extracted information model. An example for this use case with sample NLQs will be presented in Experiments Section 7.

Formal Definition of Augmentation. Given a retrieved subgraph G', the set of linked entities U_q, and an NLQ q, the augmentation step constructs a prompt P for the LLM.

$$P = \Phi(G', U_q, q, \theta) \tag{22}$$

where Φ is a deterministic template function and θ contains task-specific parameters such as query type (e.g., `ASK`, `SELECT`) or XML mode.

Thus, augmentation is the process of mapping structured KG context and NLQ input into a well-formed prompt string:

$$\Phi : (G', U_q, q) \rightarrow P. \tag{23}$$

6. Third Step: Generation

In the previous section, examples of prompts for both use cases were provided. These prompts are sent to the LLM engine, and the generated results are processed accordingly. This study utilized LLM engine: GPT-4o. Both models were accessed programmatically via API calls within the Azure platform. The same LLM configurations were applied to both the entity linking and text generation components. The parameter settings for these configurations are as follows:

LLM configuration

- **model = <gpt deployment>** → Specifies the model deployment being used. In Azure, this refers to the deployed instance of the LLM (e.g., GPT-4o) that is accessed via API.
- **messages = <prepared prompt>** → Represents the input messages sent to the LLM, typically structured as a conversation history with user and assistant messages.
- **max_tokens = 4096** → Defines the maximum number of tokens (words, subwords, or characters) that the model can generate in a single response.
- **temperature = 0** → Controls the randomness of the model's output. A value of 0 means the responses are more deterministic and consistent, while higher values (e.g., 0.7) increase variability.
- **top_p = 0.95** → Implements nucleus sampling, where the model considers only the top 95% of probable token choices, helping to balance randomness and coherence in responses.
- **frequency_penalty = 0** → Adjusts the likelihood of repeated phrases. A higher value discourages repetition by penalizing frequent token usage.
- **presence_penalty = 0** → Encourages the use of new words by increasing the probability of introducing novel tokens, reducing redundancy in responses.
- **stop = None** → Defines specific stop sequences that, if encountered, will terminate the response. When set to `None`, the model continues generating output until it reaches the token limit.
- **stream = False** → Determines whether the response is streamed in real-time. If set to `False`, the entire response is generated before being returned; if `True`, output is streamed incrementally.

Thus, depending on the specific use case, either a SPARQL query or an information model in XML format is generated.

Formal Definition of Generation. Given the constructed prompt P and an LLM model M, the generation step produces an output y:

$$y = M(P), \tag{24}$$

where M is a probabilistic sequence-to-sequence model.

Depending on the task, the output space is:

$$y \in \begin{cases} \mathcal{S}_{\text{ASK}} \cup \mathcal{S}_{\text{SELECT}}, & \text{(SPARQL query)} \\ \mathcal{X}_{\text{IM}}, & \text{(populated XML-based information model)} \end{cases} \tag{25}$$

Thus, generation is formalized as:

$$M : P \to y. \tag{26}$$

7. Experiments and Results

This section presents a comprehensive evaluation of the proposed system. The experimental analysis is organized into three subsections. Subsection 7.1 introduces the datasets used throughout the evaluation, including the OPC UA knowledge graphs and the SAREF ontology. Subsection 7.2 the two baseline algorithms implemented for comparison. Subsection 7.3 reports the experiments and results for the three main tasks addressed in this work—semantic rule validation, information retrieval, and information model generation—each represented by a dedicated set of test-case scenarios.

All code artifacts, the NLQ sets prepared for each use case and dataset, their corresponding ground-truth SPARQL queries, the result Excel sheets, the baseline and our RAG implementations, as an end-to-end working tool are available in our GitHub repository [21]. The curated SAREF dataset used in our cross-standard evaluation is also provided in the same location. The OPC UA datasets (Robotics and PackML KGs) employed in our experiments are confidential and therefore cannot be shared; however, the entire pipeline can be executed on the publicly released SAREF dataset.

7.1. Datasets

To evaluate the proposed system across heterogeneous industrial scenarios, we rely on two categories of semantic artifacts: (i) domain-specific OPC UA knowledge graphs, and (ii) the SAREF ontology as a cross-domain benchmark for assessing generalizability. The OPC UA artifacts comprise two companion specifications—Robotics and PackML—while SAREF serves as an independent, non-OPC UA ontology, enabling us to validate the applicability of our pipeline beyond a single industrial standard.

7.1.1. OPC UA Standard

All OPC UA NodeSet XML files were converted into RDF/Turtle format and, where necessary, manually populated with additional instances to support more complex NLQs. Due to confidentiality constraints, the complete TTL versions cannot be publicly released; however, each graph faithfully reflects the full content of the original XML specification. In every case, the resulting knowledge graphs far exceed the context window of existing LLMs, making full-graph prompting infeasible and highlighting the need for selective retrieval via our RAG pipeline.

Robotics: The Robotics knowledge graph is derived from the official OPC UA Robotics companion specification, based on the `Opc.Ua.Robotics.NodeSet2.xml`[7] file. For experimentation, the RDF/Turtle representation was manually enriched with additional robot instances (e.g., `ArticulatedRobot`, `CartesianRobot`) to enable NLQs involving concrete individuals rather than only schema-level definitions. The final Robotics knowledge graph contains 21,943 triples (approximately 285,741 tokens when serialized), with all token counts computed using OpenAI's tokenization method. It represents detailed robotic structures such as joint models, coordinate systems, tools, device hierarchies, and capability definitions. Its rich structure makes it a suitable testbed for evaluating multi-hop reasoning and instance-aware SPARQL generation. A reduced visualization is included in Appendix A for illustrative purposes; the full graph is significantly larger.

PackML The PackML knowledge graph is constructed from the official OPC UA PackML companion specification, using the `Opc.Ua.PackML.NodeSet2.xml`[8] file as the base artifact. The resulting Turtle-based KG represents standardized PackML concepts including machine modes, command sets, transitions, and state-machine logic. The PackML knowledge graph comprises 11,869 triples (approximately 587,807 tokens when serialized). Although smaller than Robotics, it is semantically richer in state-based behavior, making it a valuable complementary domain for NLQs involving operational states and transitions. Together, the Robotics and PackML graphs allow us to evaluate the system across two distinct modeling paradigms within the OPC UA ecosystem—structural modeling and state-machine modeling.

7.1.2. SAREF

To assess cross-standard generalizability, we additionally evaluate our pipeline on the SAREF ontology, a widely used semantic model for IoT, smart environments, and industrial energy systems. In contrast to the instance-heavy OPC UA artifacts, SAREF is primarily schema-driven, offering a structurally distinct semantic space. For our experiments, we combined three SAREF modules[9]—Core, Energy, and INMA (Industry and Manufacturer)—to construct a sufficiently large and diverse knowledge graph enriched with customised instantiations. The resulting integrated graph of SAREF contains 2,921 triples (around 49,475 serialized tokens) and serves as an independent benchmark to testing ontology-agnostic performance, confirming that our retrieval and SPARQL-generation approach is not tied to a single industrial standard. The merged graph and corresponding competency questions, also used in our previous work [3], are available in the GitHub repository [21].

7.2. Baseline RAG Algorithms

To evaluate the performance of the proposed framework relative to established retrieval approaches, two representative RAG pipelines from the literature were selected and re-implemented. All baselines were executed using identical NLQs, identical prompting templates (except for required method-specific metadata), and identical LLM configurations. Below, we summarise the design principles and operational characteristics of each baseline.

[7] https://github.com/OPCFoundation/UA-Nodeset/tree/latest/Robotics
[8] https://github.com/OPCFoundation/UA-Nodeset/tree/latest/PackML
[9] https://SAREF.etsi.org/

7.2.1. Lightweight RAG

As a baseline, we implement a lightweight RAG pipeline that follows the canonical two-stage architecture: (i) embedding-based retrieval over a vector store, and (ii) single-shot SPARQL generation by an LLM conditioned on the retrieved context. The RDF knowledge source is ingested into an OpenAI vector store using the same retrieval layer that underpins ChatGPT's "Upload file / Add to context" functionality, in accordance with the official OpenAI File Search and RAG guidelines.[10] For each natural-language query, the system retrieves the top-k most relevant text chunks via the `file_search` API and injects them into a fixed prompt template for query generation.

The pipeline is intentionally kept minimal—without graph reasoning, re-ranking, or iterative retrieval—so that it functions as a clear lower-bound baseline. This configuration aligns with the "naive" or "baseline RAG" setups commonly recommended in OpenAI's best-practice guidelines and in the broader literature, ensuring that performance gains in our advanced methods can be attributed specifically to graph-based retrieval and reasoning components rather than to architectural complexity.

7.2.2. Schema-aware RAG

We introduce a schema-aware RAG approach to address limitations of embedding-based retrieval, which often suffers from retrieval noise and semantic drift in ontology-grounded SPARQL tasks. Instead of relying on dense vectors, the method retrieves evidence directly from the RDF graph by computing simple keyword-based similarity (Jaccard overlap) between the NLQ and each triple. This yields a fully deterministic and transparent retrieval process. The approach additionally identifies the most relevant classes and properties, enabling the LLM to operate on a compact, NLQ-aligned subgraph rather than the entire ontology.

Compared to classical embedding-based RAG, this method is structure-preserving, model-agnostic, and avoids the operational complexity of vector databases. It consistently provides a cleaner and more semantically faithful context for SPARQL generation, particularly in domains driven by formal ontology constraints.

7.3. Use Cases and Results

The evaluation spans three major use cases: (1) semantic rule validation, (2) information retrieval, and (3) information model generation. Each use case contains dedicated test cases designed to demonstrate system functionality, comparative performance, robustness, and cross-ontology generalizability.

7.3.1. Validation

The objective of this use case is to demonstrate that the proposed validation mechanism operates reliably across different OPC UA domains. Two industrially relevant OPC UA Companion Specifications were selected: *OPC UA Robotics* and *OPC UA PackML*. Using multiple domains illustrates that the method is not limited to a single OPC UA modeling style, providing the first indication of generalizability. The theoretical motivation and technical relevance of validation were discussed in the Background section; here, we report the experimental setup and results.

Experimental Setup Following[3], NLQs are extracted from textual specifications and used to generate ASK-type SPARQL queries for validating semantic artifacts. Each NLQ is run five times using the LLM configuration described in Section 6. A test case is considered successful if at least three out of five executions yield the expected result. In total, 12 NLQs are evaluated for the Robotics specification and 13 for the PackML specification. All prepared NLQ sets are available in the GitHub repository [21].

Test Case 1: OPC UA Robotics 12 compliance rule statements were extracted and manually annotated from the Robotics Companion Specification. Table 3 shows a representative entry from our validation set, including the natural language question (NLQ), its corresponding ground truth SPARQL query, and evaluation results. The evaluation follows a majority criterion. The full suite of annotated cases is available in the project repository [21]. An NLQ is considered correctly handled if the system-generated SPARQL matches or is semantically equivalent to the annotated ground truth.

[10] https://platform.openai.com/docs/guides/retrieval

Table 3
OPC UA Robotics Validation: NLQ execution results and pass/fail decision

NLQ	Ground Truth SPARQL	Evaluation (x/5)	Result

The InverseName of Moves is IsMovedBy.	`PREFIX NS2: <http://opcfoundation.org/UA/Robotics/>`
	`PREFIX ta: <http://opcfoundation.org/UA/Meta/TA/>`
	`ASK {`
	`  NS2:moves ta:inverseName "IsMovedBy" .`
	`}`
		5/5	Pass

Test Case 2: OPC UA PackML This test set was constructed using a methodology analogous to that employed in the Robotics evaluation. Twenty compliance rule statements were manually extracted and annotated from the PackML Companion Specification [3]. Table 4 presents an illustrative annotated instance, showing the natural language query (NLQ), its corresponding ground-truth SPARQL ASK query, and the evaluation outcome. The ASK-type query returns a Boolean value (**True** or **False**), indicating whether the tested semantic artifact satisfies the specified compliance requirement. For the NLQ shown in the example, our RAG method was executed five times, and in four of these runs it generated a correct SPARQL query that passed the validation. The complete PackML evaluation results for all NLQs are available in the associated repository [21].

Table 4
OPC UA PackML Validation: NLQ execution results and pass/fail decision

NLQ	Ground Truth SPARQL	Evaluation (x/5)	Result

The labels of all subclasses of Structure have the label which includes "DataType"	`PREFIX ns3: <http://opcfoundation.org/UA/>`
	`PREFIX rdf1: <http://www.w3.org/1999/02/22-rdf-sy...`
	
	`ASK {`
	`  ?subclass rdfs1:subClassOf ns3:Structure .`
	`  ?subclass rdfs1:label ?label .`
	`  FILTER(CONTAINS(str(?label), "DataType"))`
	`}`
		5/5	Pass

Validation Evaluation Results For the validation use case, each NLQ was executed five times, and Table 5 summarises the pass/fail outcomes, execution counts, and accuracies for both knowledge graphs. We report two accuracy measures: (i) NLQ-level accuracy, where a query is deemed correct if at least three of its five executions succeed, and (ii) execution-level accuracy, which reflects the overall success rate across all individual runs. Our RAG approach achieves NLQ-level accuracies of 75% for Robotics and 74% for PackML, together with consistently high execution-level performance, demonstrating its effectiveness for semantic artifact validation.

Table 5
Validation results: NLQ count, pass/fail accuracy, and execution accuracy.

	Number of NLQs	Pass/Fail Accuracy	Per-Execution Level Accuracy
Robotics	12	0.75	0.70 (=47/60)
PackML	13	0.77	0.72 (=42/60)

7.3.2. Information Retrieval

This use case evaluates the system's ability to retrieve semantically relevant information from large industrial ontologies using NLQs. Five NLQs were designed for the *OPC UA Robotics* information model. For each query, outputs were generated using the proposed system and the two benchmark RAG approaches. All prompts were identical across systems except for necessary method-specific additions (e.g., entity-extraction metadata applied only in our approach).

The same evaluation pipeline was then applied to the *SAREF* ontology to assess cross-ontology stability. Finally, to explore the robustness of the system, four progressively more complex or under-specified variants of each Robotics NLQ were created. These variants allow us to analyze the system's behavior under ambiguity, incomplete information, and increasing reasoning depth.

Test Case 1: Five OPC UA NLQs evaluated across three RAG systems Each of the five NLQs was executed five times per system, producing accuracy-based comparative results. Table 6 summarises the outcomes of executing each of the five Robotics NLQs five times, reporting results under both evaluation settings. The table compares our RAG method with two baseline RAG variants to facilitate a direct performance assessment. As shown in Table 7, our RAG method attains the highest accuracy at both the NLQ and execution levels, substantially outperforming the baselines on the Robotics knowledge graph.

Table 6
Accuracy comparison for five NLQs in the Robotics domain across three retrieval methods: our RAG approach and two baseline systems

NLQ ID	Exact NLQs	Our RAG		Lightweight RAG		Schema-aware RAG	
		exec acc	pass/fail	exec acc	pass/fail	exec acc	pass/fail
1	How many instances of Robot are there? what are their Label?	5/5	Pass	0/5	Fail	5/5	Pass
2	What are the RobotCategoryEnumeration in this knowledge graph?	5/5	Pass	0/5	Fail	4/5	Pass
3	What are the components of ArticulatedRobot?	5/5	Pass	0/5	Fail	0/5	Fail
4	Fetch the ActualPosition attribute of the ArticulatedRobot	5/5	Pass	0/5	Fail	0/5	Fail
5	What are the possible values and their Label for the AxisMotionProfileEnumeration?	5/5	Pass	0/5	Fail	5/5	Pass

Table 7
Information retrieval performance on the Robotics dataset comparing OurRAG with two baseline RAG models

KG\Models	Our RAG		Lightweight RAG		Schema-aware RAG	
	exec acc	pass/fail	exec acc	pass/fail	exec acc	pass/fail
Robotics	1.00 (=25/25)	1.00 (=5/5)	0.0 (=0/25)	0.0 (=0/5)	0.56 (=14/25)	0.60 (=3/5)

Test Case 2: Five NLQs as competency questions for the SAREF ontology, evaluated using the same three RAG systems. The NLQs prepared for the SAREF ontology—previously used in [3]—were evaluated using the same prompting configuration and system setup as in Test Case 1. Table 8 reports the results obtained by executing each of the five NLQs five times, presenting outcomes for both evaluation settings. Three RAG approaches were executed, and their results are included for comparison. Furthermore, Table 9 provides a summary of the total information retrieval performance on the SAREF ontology. Across all evaluations, our RAG approach clearly outperforms the baselines.

Comparable performance across OPC UA and SAREF demonstrates the **consistency** and **generalizability** of the proposed method.

Test Case 3: NLQ Variants for Robustness Assessment Table 10 lists the five Robotics NLQs introduced in Table 6, together with two additional variants of each query that differ in their linguistic form:

- **Exact** — the canonical formulation that reproduces the KG labels verbatim,
- **Variation A** — a mild rephrasing that keeps all domain-specific terms intact,
- **Variation B** — a more substantial rephrasing that substitutes some KG labels with close synonyms,

These three versions enable a systematic study of robustness, letting us analyse the system's tolerance to paraphrasing, its ability to semantically normalise queries, and its performance on multi-hop reasoning. As the wording gradually deviates from the original KG labels, we observe a correspondingly smooth reduction in accuracy (see Section 8 for details).

Table 11 presents the accuracy of NLQs and their variations from Table 10. Even though the accuracy decreases while using synonyms or uncommon terminology in variations, it remains above 88% when correct EL is applied.

Table 8
Accuracy comparison for five NLQs in the SAREF domain across three retrieval methods: our RAG approach and two baseline systems

NLQ ID	Exact NLQs	Our RAG		Lightweight RAG		Schema-aware RAG	
		run acc	pass/fail	run acc	pass/fail	run acc	pass/fail
1	What is the instance of the temperature sensor?	4/5	Pass	3/5	Pass	5/5	Pass
2	Which service does a temperature sensor offer?	4/5	Pass	1/5	Fail	0/5	Fail
3	What task does a temperature sensor device accomplish?	5/5	Pass	2/5	Fail	4/5	Pass
4	What function does a temperature sensor device have?	4/5	Pass	1/5	Fail	0/5	Fail
5	In which state is a temperature sensor currently in?	5/5	Pass	0/5	Fail	4/5	Pass

Table 9
Information retrieval performance on the SAREF dataset comparing Our RAG with two baseline RAG models

KG\Models	Our RAG		Lightweight RAG		Schema-aware RAG	
	exec acc	pass/fail	exec acc	pass/fail	exec acc	pass/fail
SAREF	0.88 (=22/25)	1.00 (=5/5)	0.0 (=0/25)	0.20 (=1/5)	0.52 (=13/25)	0.60 (=3/5)

7.3.3. Information Model Generation

This use case evaluates the system's ability to generate or extend information-model fragments within large semantic artifacts through a two-stage process:

1. Stage 1 (Scope Extraction): Given an NLQ-type-1 (e.g., "An information model for a robot with three axes"), the system identifies the contextual scope and extracts the corresponding subgraph (i.e., the relevant sub-information model).
2. Stage 2 (Model Population): Given an NLQ-type-2 describing a requirement (e.g., "add speed and motor to the articulated robot"), the system produces the corresponding model extension operations.

Standards like OPC UA provide comprehensive information models covering numerous machine variations, making it difficult for engineers to instantiate models for specific variants. This use case addresses this challenge by enabling automated generation of machine-specific models from standardized templates.

In Stage 1, users submit an NLQ-type-1 specifying requirements. After performing NERD and EL, the system applies RAG to extract a relevant subgraph including matched nodes and related entities, yielding a focused information model. Stage 2 enables iterative refinement (NLQ-type-2). Users submit additional NLQs to add entities, attributes, or relationships. The LLM generates only new or modified nodes, which are algorithmically integrated into the existing model, allowing dynamic extension beyond the original KG content.

Test Case 1: Using the base Robotics KG, we executed:

1. Created a compact model via NLQ-type-1: *"An ArticulatedRobot with 3 axes."*
2. Added attributes via NLQ-type-2: *"Add speed and motor attributes to the articulated robot."*

Results are presented as follows:

Table 10
Correctness evaluation of five NLQ pairs (original form and natural-language variations) for information retrieval on the Robotics knowledge graph

NLQ ID	NLQ	EL Acc	Query Generation Acc	Fail/Pass
1	**Exact:** How many instances of Robot are there? What are their Label?	5/5	5/5	Pass
	Variation A: How many instances of robot are there? What are their labels?	5/5	5/5	Pass
	Variation B: How many instances of robot are there? What are their names?	5/5	4/5	Pass
2	**Exact:** What are the RobotCategoryEnumeration in this knowledge graph?	5/5	5/5	Pass
	Variation A: What categories of robots can be created using this model?	5/5	5/5	Pass
	Variation B: What are the different types of robots that can be built with this model?	0/5	0/5	Fail
3	**Exact:** What are the components of ArticulatedRobot?	5/5	5/5	Pass
	Variation A: What are the components of the articulated robot?	5/5	4/5	Pass
	Variation B: What are the parts of the articulated robot?	5/5	5/5	Pass
4	**Exact:** Fetch the ActualPosition attribute of the ArticulatedRobot.	5/5	5/5	Pass
	Variation A: Fetch the actual position attribute of the articulated robot.	5/5	3/5	Pass
	Variation B: What is the real position of the articulated robot?	1/5	1/5	Fail
5	**Exact:** What are the possible values and their Label for the AxisMotionProfileEnumeration?	5/5	5/5	Pass
	Variation A: What are the possible values and corresponding names for the axis motion profile enumeration?	5/5	5/5	Pass
	Variation B: What are the different motion profile settings for an axis, along with their corresponding names?	0/5	0/5	Fail

Original node definition from the base KG generated after NLQ-type-1

```
<UAObject NodeId="ns=4;i=5001" BrowseName="4:ArticulatedRobot" ParentNodeId="i=85">
    <DisplayName>ArticulatedRobot</DisplayName>
    <Description>A typical three-axis industrial articulated robot with 2
    power trains that moves 3 axes. Power train 1 moves axis 1, power train
    2 moves axis 2 and axis 3.</Description>
    <References>
        <Reference ReferenceType="i=40">ns=2;i=1002</Reference>
        <Reference ReferenceType="i=47">ns=4;i=5024</Reference>
        <Reference ReferenceType="i=47">ns=4;i=5025</Reference>
        <Reference ReferenceType="i=47">ns=4;i=5042</Reference>
    </References>
</UAObject>
```

Table 11
Entity Linking (EL) and SPARQL Query Generation performance results grouped by NLQ variations

NLQ Form	EL Acc	Per-Execution Level Acc	NLQ-level Acc	Acc with given EL
Exact	1.00 (=25/25)	1.00 (=25/25)	1.00 (=5/5)	1.00 (=25/25)
Variation A	1.00 (=25/25)	0.88 (=22/25)	1.00 (=5/5)	0.88 (=22/25)
Variation B	0.44 (=11/25)	0.40 (=10/25)	0.40 (=2/5)	0.91 (=10/11)

Updated node definition with the newly added attributes generated after NLQ-type-2

```
<UAObject NodeId="ns=4;i=5001" BrowseName="4:ArticulatedRobot" ParentNodeId="i=85">
    <DisplayName>ArticulatedRobot</DisplayName>
    <Description>A typical three-axis industrial articulated robot with 2
    power trains that moves 3 axes. Power train 1 moves axis 1, power train
    2 moves axis 2 and axis 3.</Description>
    <References>
        <Reference ReferenceType="i=40">ns=2;i=1002</Reference>
        <Reference ReferenceType="i=47">ns=4;i=5024</Reference>
        <Reference ReferenceType="i=47">ns=4;i=5025</Reference>
        <Reference ReferenceType="i=47">ns=4;i=5042</Reference>
        <Reference ReferenceType="i=47">ns=4;i=6001</Reference>
        <Reference ReferenceType="i=47">ns=4;i=6002</Reference>
    </References>
</UAObject>
<UAObject NodeId="ns=4;i=6001" BrowseName="4:Speed" ParentNodeId="ns=4;i=5001">
    <DisplayName>Speed</DisplayName>
    <Description>Speed attribute of the articulated robot.</Description>
    <References>
        <Reference ReferenceType="i=40">i=61</Reference>
        <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5001</Reference>
    </References>
</UAObject>
<UAObject NodeId="ns=4;i=6002" BrowseName="4:Motor" ParentNodeId="ns=4;i=5001">
    <DisplayName>Motor</DisplayName>
    <Description>Motor attribute of the articulated robot.</Description>
    <References>
        <Reference ReferenceType="i=40">i=61</Reference>
        <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5001</Reference>
    </References>
</UAObject>
```

The updated ArticulatedRobot node includes `Motor` and `Speed` attributes (bold). Both are added as new nodes with corresponding `NodeId`s, `DisplayName`s, and references.

The model achieves significant size reduction: from over 21,943 triples in the original Robotics KG to only 217 triples (pre-XML conversion), creating a lightweight yet functionally complete model for the specific use case.

The complete model appears in Appendix B and was validated by domain experts.

8. Discussion

Graph-RAG in a nutshell: We introduce Graph-RAG, a retrieval–augmentation–generation architecture expressly designed for large, RDF-based semantic artifacts whose size and structural complexity exceed the context window of contemporary LLMs. The pipeline begins with an LLM-driven NERD and an EL stage that detects domain entities in an NLQ and deterministically links them to canonical URIs in the target KG. These starting nodes act as anchors for a symbolic subgraph–construction module: the shortest paths containing all starting nodes are computed, then selectively expanded with (i) class–instance pairs, (ii) `hasComponent`/`hasProperty` branches, and (iii) optional super-classes, while also satisfying a token budget. The result is a self-contained, semantically coherent slice G' of the original KG that preserves namespaces, edge directions and axiomatic context yet remains compact enough to fit inside the LLM prompt.

Key advantages over baseline RAG variants: Unlike purely text-centric *lightweight RAG* pipelines or triple-bag *schema-aware RAG* baselines, Graph-RAG operates on structurally faithful subgraphs. This yields four concrete

benefits that were repeatedly confirmed in our experiments: (i) **Topology preservation** prevents the relational semantics from being flattened away by naïve chunking. (ii) Namespace fidelity exposes fully qualified URIs, enabling the LLM to emit syntactically correct and executable SPARQL. (iii) Grounded generation curbs hallucination. Every triple in the prompt is guaranteed to originate from the authoritative model. (iv) Dual reasoning over both schema and instance data lets the same pipeline support validation, retrieval and instantiation. Empirically, these design choices translate into substantial accuracy gains over both of the baselines (see Section 7.2): on Robotics, Graph-RAG reaches 100% NLQ-level accuracy, while schema-aware and lightweight baselines trail at 60% and 0%, respectively (Table 7); on SAREF the gap remains pronounced (100% vs. 60% and 20%).

Use-case 1: Semantic rule validation: Across 25 compliance statements—12 for Robotics and 13 for PackML—Graph-RAG reaches 75% and 77% NLQ-level accuracies, with per-execution success rates of 70% and 72%, respectively (Table 5). Importantly, a run is considered *correct* when the produced SPARQL captures the intent of the NLQ—even if its syntax diverges from the single ground-truth query used for annotation. Manual inspection shows that most residual errors originate from entity-linking slips; when the right entities are anchored, the pipeline tends to generate sound validation logic.

The benefit of structural context can be illustrated with the statement, which is among the NLQ set for validation of Robotics KG: "*The* `IsConnectedTo` *reference has no distinct* `InverseName`." A baseline, unaware of the graph's axiomatics, typically issues a generic `FILTER NOT EXISTS { ?ref ns3:inverseName ?inv }` pattern that merely checks for the attribute's absence. Graph-RAG, in contrast, sees in the retrieved subgraph that `IsConnectedTo` is an `owl:SymmetricProperty`. It therefore formulates a different test: the `inverseName` *may exist* but must equal the forward label, which is exactly how OPC UA encodes symmetric references. This example highlights how graph-aware retrieval yields validation queries that are not only syntactically correct but also semantically aligned with domain conventions.

Use-case 2: Information retrieval: On the Robotics KG Graph-RAG delivers perfect performance on NLQs with canonical phrasing (25/25 successful executions, 5/5 NLQs), while the lightweight text-centric baseline fails completely and the schema-aware triple bag reaches only 56% execution-level accuracy (Table 7). Results on SAREF confirm generalizability: our method keeps an 88% execution-level score and 100% NLQ-level accuracy, whereas the vector-store baseline again collapses (0%) and the schema-aware baseline plateaus at 52%. The advantage stems from (i) precise entity anchoring, (ii) automatic inclusion of `hasComponent`/`hasProperty` chains, and (iii) namespace-explicit prompts that prevent prefix confusion—common failure modes observed in the baselines.

Robustness to NLQ paraphrasing: The three-tier paraphrase study (Exact, Variation A, Variation B; Table 10) probes the limits of our EL module. With perfect linking, SPARQL generation remains robust at **91 %** even for Variation B (synonyms, non-canonical phrases). Overall accuracy drops to 40% when linking fails, indicating that entity disambiguation is now the primary bottleneck, a clear avenue for future work. When linking succeeds (oracle EL column in Table 11) the pipeline consistently exceeds 88% across all paraphrase levels.

Use-case 3: Information-model generation: Graph-RAG also supports *instantiation*. Starting from 21,943 triples in the full Robotics KG, the system distilled a 217-triple sub-model for "*an articulated robot with three axes*." Subsequent NLQ-driven extensions correctly injected `Speed` and `Motor` nodes, complete with new `NodeId`s and bidirectional references, yielding a domain-expert-validated XML fragment (Appendix B). These interactive steps, underscore the viability of Graph-RAG for dynamic model engineering tasks.

Key Insights Across the three evaluated ontologies, Robotics, PackML, and SAREF, Graph-RAG delivers strong, task-appropriate performance. In information-retrieval tests it outperforms both baseline pipelines by 40–100 pp, while in validation use case it attains 75–77% NLQ-level accuracy, and in information-model generation it produces structurally and semantically correct extensions. These results trace back to its ability to supply the LLM with compact, topology-preserving, namespace-explicit subgraphs, yielding precise SPARQL and fewer hallucinations. Remaining errors are dominated by entity-linking slips, indicating a clear avenue for future improvement.

9. Related Work

9.1. Text-based Retrieval-Augmented Generation

Recent advances in large language models (LLMs) have revitalized research on semantic parsing and NLQ-to-SPARQL translation [22–24]. Most existing systems rely on text-based RAG, where RDF triples are serialized into text and retrieved using dense embeddings [7, 25]. While effective for small, shallow, or uniformly structured benchmarks, such methods are not graph-aware and therefore do not incorporate hierarchy, part–whole relations, inheritance, or typed edges. As a result, they often fail to retrieve coherent or complete context for large industrial ontologies such as OPC UA and SAREF, where answering even simple NLQs may require multihop traversal and schema-informed reasoning rather than surface-level similarity.

9.2. Graph-based RAG

Graph-based RAG extends classical RAG by incorporating graph-structured information during retrieval. Existing approaches can be grouped into methods operating directly on preexisting knowledge graphs and methods that induce task-specific graphs from unstructured corpora [26]. The latter category includes the two-stage pipeline of [27], which constructs entity graphs from large podcast [28] and news datasets [29]. Microsoft's GraphRAG also follows this paradigm by converting PDF segments into lightweight graph structures for question answering. However, these text-derived pipelines are designed for unstructured corpora and cannot preserve the typed semantics, class hierarchies, or constraints of industrial ontologies.

Further domain-specific work includes a graph-based RAG system for medical question answering [30] and the KG-RAG4SM framework [31], which integrates vector, traversal, and query-based retrieval for schema matching across datasets such as MIMIC4 [32], Synthea5 [33], CMS6 [34], and EMED [32]. Although these studies demonstrate the potential of graph-based retrieval, performance remains highly sensitive to the choice of graph embeddings [35], which are known to struggle with NLQ-driven multihop reasoning.

Despite this progress, several gaps remain. Existing Graph-RAG systems do not address large OWL-based industrial knowledge graphs such as OPC UA and SAREF. Embedding-based retrieval methods such as Node2Vec and GNN variants perform poorly for NLQ-driven graph reasoning, as also confirmed by our initial experiments. Moreover, current methods do not combine ontology signals with symbolic graph traversal, and therefore lack schema-aware retrieval mechanisms suited to industrial semantic artifacts.

9.3. Similar Approaches

Our prior work investigates the use of LLMs for generating SPARQL queries and validating semantic artifacts over small-scale industrial knowledge bases [3, 11, 36]. Other related efforts include SPARQL generation for bioinformatics knowledge graphs [37] and model-comparison studies showing strong LLM dependence in KGQA performance [38]. Additional industrial work includes [39], which integrates Amazon Bedrock with OPC UA companion specifications but relies on SQL-based retrieval and document parsing rather than graph-based reasoning. These studies indicate increasing interest in LLM-assisted interaction with structured data; however, none address large industrial graphs, and none provide ontology-aware subgraph extraction required for reliable SPARQL generation.

9.4. Positioning Our Approach

Our approach, referred to as Our RAG, differs from existing text-based and graph-based methods in several respects. First, NLQ interpretation is grounded in explicit entity linking to industrial URIs rather than text similarity. Second, instead of chunk retrieval, embedding search, or community detection, we compute the minimal connected subgraph containing all linked nodes and use this as the structurally coherent context for query generation. Third, we expand this subgraph using symbolic rules that capture class–instance relations, component structure, and property definitions characteristic of industrial ontologies. To our knowledge, no prior system combines these elements in a single pipeline or demonstrates ontology-agnostic retrieval and SPARQL generation across both OPC UA and

SAREF. For comparative evaluation, we also implemented a lightweight text-based RAG baseline and a schema-aware graph-based baseline without traversal. Across all experiments, Our RAG achieved the highest correctness, indicating that graph-structural retrieval is essential for complex industrial NLQ tasks.

10. Conclusion and Future Work

This study introduced a graph-based RAG architecture that enables natural-language interaction with large-scale semantic artifacts such as OPC UA and SAREF. By integrating LLM-based NERD and EL, symbolic shortest-path retrieval, and controlled rule-driven subgraph expansion, the system overcomes the token and structural limitations that render classical RAG pipelines ineffective for industrial knowledge graphs. Through this design, the work successfully addresses the three research questions defined in Section **RQ1**–**RQ3**: **RQ1 (Design)** is satisfied by introducing a scalable architecture for NLQ–KG interaction capable of extracting complete, symbolic subgraphs from large RDF models; **RQ2 (Application)** is answered through the demonstrated support for validation, information retrieval, and information model generation; and **RQ3 (Generalization)** is confirmed through experiments across heterogeneous semantic artifacts, including OPC UA Robotics, OPC UA PackML, and SAREF. Correspondingly, the system fulfils all contributions defined in **CONT1–CONT4**, including the development of an end-to-end graph-based RAG framework, its evaluation across multiple industrial tasks, empirical demonstration of cross-domain generalizability, and performance improvements over state-of-the-art baselines.

Across all use cases, the proposed approach consistently outperforms text-based and schema-aware baselines, particularly in tasks requiring multi-hop reasoning, class–instance alignment, and structural completeness. The results confirm that symbolic, path-oriented retrieval—rather than embedding-based chunk retrieval—is essential for enabling accurate SPARQL generation, semantic rule validation, and NLQ-driven instantiation in industrial contexts.

Future work will focus on extending the proposed architecture to additional industrial semantic artifacts such as AAS and WoT, enabling broader applicability across heterogeneous modeling ecosystems. Beyond natural-language queries, the framework will be expanded to support fusion-based interaction strategies that integrate multiple input modalities, including structured templates, engineering metadata, and domain-specific contextual signals. Such multimodal extensions would allow the system to handle more complex engineering scenarios where textual, symbolic, and model-level information must be combined for accurate reasoning. Furthermore, applying the pipeline to larger cross-domain semantic infrastructures will help assess its generalizability and reveal further optimization opportunities. Collectively, these directions aim to transform the system into a comprehensive, versatile interface for next-generation semantic engineering workflows.

11. Acknowledgements

This work is supported by the Horizon Europe Research and Innovation Actions under grant number 101092908 (Smart Edge).

References

[1] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[2] Mohaimenul Azam Khan Raiaan, Md Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access*, 2024.

[3] Nilay Tufek, Aparna Saissre Thuluva, Valentin Philipp Just, Fajar J Ekaputra, Tathagata Bandyopadhyay, Marta Sabou, and Allan Hanbury. Validating semantic artifacts with large language models. In *European Semantic Web Conference*, pages 92–101. Springer, 2024.

[4] Amanda Kau, Xuzeng He, Aishwarya Nambissan, Aland Astudillo, Hui Yin, and Amir Aryani. Combining knowledge graphs and large language models. *arXiv preprint arXiv:2407.06564*, 2024.

[5] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[6] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[7] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[8] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*, 2020.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[11] Nilay Tufek, Aparna Sai Sree Thuluva, Valentin Philipp Just, and Marta Sabou. Towards extraction of validation rules from opc ua companion specifications. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2023.

[12] OPC Foundation. Opc ua robotics companion specification nodeset. https://github.com/OPCFoundation/UA-Nodeset/tree/latest/Robotics. Accessed: 2025-11-20.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[14] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[15] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[16] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[17] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*, 2024.

[18] OpenAI. Gpt-4 technical report, 2023. URL https://openai.com/research/gpt-4. Accessed: 2025-04-22.

[19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7871–7880. Association for Computational Linguistics, 2020.

[21] Nilay Tüfek ÖzkaYa. Industrialgraphrag: A repository for rag on industrial knowledge graphs. https://github.com/nilaytufekozkaya/IndustrialGraphRAG, 2025. GitHub repository, retrieved December 5, 2025.

[22] Shengli Song, Wen Huang, and Yulong Sun. Semantic query graph based sparql generation from natural language questions. *Cluster Computing*, 22(Suppl 1):847–858, 2019.

[23] Haemin Jung and Wooju Kim. Automated conversion from natural language query to sparql query. *Journal of Intelligent Information Systems*, 55(3):501–520, 2020.

[24] Hamada M Zahera, Manzoor Ali, Mohamed Ahmed Sherif, Diego Moussallem, and Axel-Cyrille Ngonga Ngomo. Generating sparql from natural language using chain-of-thoughts prompting. In *SEMANTICS*, pages 353–368, 2024.

[25] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

[26] Tyler Thomas Procko and Omar Ochoa. Graph retrieval-augmented generation for large language models: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[27] Darren Edge, Hao Trinh, Nan-Chen Cheng, Justin Bradley, Andrew Chao, Apurva Mody, Samuel Truitt, and Jeff Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[28] Kevin Scott. Behind the tech. https://www.microsoft.com/en-us/behind-the-tech, 2024. Podcast.

[29] Yuchen Tang and Yiming Yang. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries, 2024. URL https://arxiv.org/abs/2401.15391. arXiv preprint.

[30] Maneeha Rani, Bhupesh Kumar Mishra, Dhavalkumar Thakker, and Mohammad Nouman Khan. To enhance graph-based retrieval-augmented generation (rag) with robust retrieval techniques. In *2024 18th International Conference on Open Source Systems and Technologies (ICOSST)*, pages 1–6. IEEE, 2024.

[31] Chuangtao Ma, Sriom Chakrabarti, Arijit Khan, and Bálint Molnár. Knowledge graph-based retrieval-augmented generation for schema matching. *arXiv preprint arXiv:2501.08686*, 2025. URL https://arxiv.org/abs/2501.08686.

[32] Alistair E. W. Johnson, Tom J. Pollard, Lu Shen, Li-wei H. Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G. Mark. A real-world dataset and benchmark for foundation model adaptation in medical image classification. *Scientific Data*, 10(1):1–9, 2023. .

[33] Jason Walonoski, Sybil Klaus, Eldesia Granger, Dylan Hall, Andrew Gregorowicz, George Neyarapally, Abigail Watson, and Jeff Eastman. Synthea™ novel coronavirus (covid-19) model and synthetic data set. *Intelligence-based medicine*, 1:100007, 2020.

[34] Anzar Afaq, Andrew Dolgert, Yuyi Guo, Chris Jones, Sergey Kosyakov, Valentin Kuznetsov, Lee Lueking, Dan Riley, and Vijay Sekhri. The cms dataset bookkeeping service. In *Journal of Physics: Conference Series*, volume 119, page 072001. IOP Publishing, 2008.

[35] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[36] Yashoda Saisree Bareedu, Thomas Frühwirth, Christoph Niedermeier, Marta Sabou, Gernot Steindl, Aparna Saisree Thuluva, Stefani Tsaneva, and Nilay Tufek Ozkaya. Deriving semantic validation rules from industrial standards: An opc ua study. *Semantic Web*, 15 (2):517–554, 2024.

[37] Vincent Emonet, Jerven Bolleman, Severine Duvaud, Tarcisio Mendes de Farias, and Ana Claudia Sima. Llm-based sparql query generation from natural language over federated knowledge graphs. In *Proceedings of Special Session on LLMs at ISWC 2024 - Bridging the Gap: Building Knowledge-Enhanced Gen AI*. ISWC, 2024. URL https://chat.expasy.org.

[38] Antonello Meloni, Diego Reforgiato Recupero, Francesco Osborne, Angelo Salatino, Enrico Motta, Sahar Vahadati, and Jens Lehmann. Assessing large language models for sparql query generation in scientific question answering. In *Proceedings of the 23rd International Semantic Web Conference (ISWC 2024)*. ISWC, 2024.

[39] Sounavo Dey. Build your factory assistant with amazon bedrock and opc ua, 2025. URL https://community.aws/content/2ucWToDNFQg8d1BuiHasKmhmbBi/build-your-factory-assistant-with-amazon-bedrock-and-opc-ua. Accessed: 2025-03-31.

Appendix A. Demonstration of KG

Fig. 7 below contains a simplified visualization of the the robotics knowledge graph used for validation use case in Section 7.3.1, for information retrieval use case in Section 7.3.2 and as the base knowledge graph for information model generation use case in Section 7.3.3.

Fig. 7. Simplified visualization of the Robotics knowledge graph

Appendix B. Complete Updated Information Model From Information Model Generation Use-Case

The following listing shows the complete information model generated in information model generation use case after the requested attributes are added to the information model.

Complete Generated Information Model

```
<?xml version='1.0' encoding='utf-8'?>
<UANodeSet LastModified="2021-02-23T07:42:39Z"
xmlns="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd"
xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd"
xmlns:si="http://www.siemens.com/OPCUA/2017/SimaticNodeSetExtensions">
  <NamespaceUris>
    <Uri>http://opcfoundation.org/UA/XML/</Uri>
    <Uri>http://opcfoundation.org/UA/Robotics/</Uri>
    <Uri>http://opcfoundation.org/UA/DI/</Uri>
    <Uri>http://siemens.com/robot/demo/</Uri>
  </NamespaceUris>
```

```
<Models>
  <Model ModelUri="http://siemens.com/robot/demo/" PublicationDate="2024-
  12-06T05:36:38-08:00" Version="1.00">
    <RequiredModel ModelUri="http://opcfoundation.org/UA/"
    PublicationDate="2021-09-15T00:00:00+00:00" Version="1.04.10" />
    <RequiredModel ModelUri="http://opcfoundation.org/UA/DI/"
    PublicationDate="2022-11-03T00:00:00+00:00" Version="1.04.0" />
    <RequiredModel ModelUri="http://opcfoundation.org/UA/Robotics/"
    PublicationDate="2021-05-20T00:00:00+00:00" Version="1.01.2" />
  </Model>
  <Model ModelUri="http://opcfoundation.org/UA/XML/"
  PublicationDate="2023-01-14T00:00:00+00:00" Version="1.0.0">
    <RequiredModel ModelUri="http://opcfoundation.org/UA/"
    PublicationDate="2021-09-15T00:00:00+00:00" Version="1.04.10" />
  </Model>
  <Model ModelUri="http://opcfoundation.org/UA/Robotics/"
  PublicationDate="2021-05-20T00:00:00+00:00" Version="1.01.2">
    <RequiredModel ModelUri="http://opcfoundation.org/UA/"
    PublicationDate="2021-09-15T00:00:00+00:00" Version="1.04.10" />
    <RequiredModel ModelUri="http://opcfoundation.org/UA/DI/"
    PublicationDate="2022-11-03T00:00:00+00:00" Version="1.04.0" />
  </Model>
  <Model ModelUri="http://opcfoundation.org/UA/DI/" PublicationDate="2022-
  11-03T00:00:00+00:00" Version="1.04.0">
    <RequiredModel ModelUri="http://opcfoundation.org/UA/"
    PublicationDate="2021-09-15T00:00:00+00:00" Version="1.04.10" />
  </Model>
</Models>
<UAObjectType NodeId="ns=2;i=1002" BrowseName="2:MotionDeviceSystemType">
  <DisplayName>MotionDeviceSystemType</DisplayName>
  <Description>Contains the set of controllers and motion devices in a
  closely-coupled motion device system.</Description>
  <References>
    <Reference ReferenceType="i=47">ns=2;i=5002</Reference>
    <Reference ReferenceType="i=47">ns=2;i=5001</Reference>
    <Reference ReferenceType="i=47">ns=2;i=5010</Reference>
    <Reference ReferenceType="i=40" IsForward="false">ns=4;i=5001</Reference>
  </References>
</UAObjectType>
<UAObjectType NodeId="ns=2;i=1004" BrowseName="2:MotionDeviceType">
  <DisplayName>MotionDeviceType</DisplayName>
  <Description>Represents a specific motion device in the motion device
  system like a robot, a linear unit or a positioner. A MotionDevice
  should have at least one axis.</Description>
  <References>
    <Reference ReferenceType="i=46">ns=2;i=16351</Reference>
    <Reference ReferenceType="i=46">ns=2;i=16353</Reference>
    <Reference ReferenceType="i=46">ns=2;i=16359</Reference>
    <Reference ReferenceType="i=46">ns=2;i=16354</Reference>
    <Reference ReferenceType="i=46">ns=2;i=16362</Reference>
    <Reference ReferenceType="i=47">ns=2;i=15305</Reference>
    <Reference ReferenceType="i=47">ns=2;i=16443</Reference>
    <Reference ReferenceType="i=47">ns=2;i=5029</Reference>
    <Reference ReferenceType="i=47">ns=2;i=5091</Reference>
    <Reference ReferenceType="i=47">ns=2;i=16566</Reference>
    <Reference ReferenceType="i=40" IsForward="false">ns=4;i=5044</Reference>
  </References>
</UAObjectType>
<UAObject NodeId="ns=4;i=5024" BrowseName="2:Controllers"
ParentNodeId="ns=4;i=5001">
  <DisplayName>Controllers</DisplayName>
  <Description>Contains the set of controllers in the motion device
  system.</Description>
  <References>
    <Reference ReferenceType="i=40">i=61</Reference>
    <Reference ReferenceType="i=47">ns=4;i=5043</Reference>
    <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5001</Reference>
  </References>
</UAObject>
<UAObject NodeId="ns=4;i=5042" BrowseName="2:SafetyStates" ParentNodeId="ns=4;i=5001">
  <DisplayName>SafetyStates</DisplayName>
  <Description>Contains safety-related data from motion device system.</Description>
  <References>
    <Reference ReferenceType="i=40">i=61</Reference>
    <Reference ReferenceType="i=47">ns=4;i=5045</Reference>
    <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5001</Reference>
  </References>
</UAObject>
<UAObject NodeId="ns=4;i=5050" BrowseName="3:ParameterSet" ParentNodeId="ns=4;i=5044">
  <DisplayName>ParameterSet</DisplayName>
  <Description>Flat list of Parameters</Description>
  <References>
    <Reference ReferenceType="i=40">i=58</Reference>
    <Reference ReferenceType="i=47">ns=4;i=6114</Reference>
```

```
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5044</Reference>
    </References>
  </UAObject>
  <UAObject NodeId="ns=4;i=5051" BrowseName="2:PowerTrains" ParentNodeId="ns=4;i=5044">
    <DisplayName>PowerTrains</DisplayName>
    <Description>PowerTrains is a container for one or more instances of the PowerTrainType.</Description>
    <References>
      <Reference ReferenceType="i=40">i=61</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5056</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5070</Reference>
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5044</Reference>
    </References>
  </UAObject>
  <UAObject NodeId="ns=4;i=5055" BrowseName="2:Axis_AR0" ParentNodeId="ns=4;i=5049">
    <DisplayName>Axis_AR0</DisplayName>
    <References>
      <Reference ReferenceType="i=40">ns=2;i=16601</Reference>
      <Reference ReferenceType="i=46">ns=4;i=6122</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5058</Reference>
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5049</Reference>
    </References>
  </UAObject>
  <UAObject NodeId="ns=4;i=5061" BrowseName="4:Axis_AR1" ParentNodeId="ns=4;i=5049">
    <DisplayName>Axis_AR1</DisplayName>
    <References>
      <Reference ReferenceType="i=40">ns=2;i=16601</Reference>
      <Reference ReferenceType="i=46">ns=4;i=6133</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5062</Reference>
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5049</Reference>
    </References>
  </UAObject>
  <UAObject NodeId="ns=4;i=5065" BrowseName="4:Axis_AR2" ParentNodeId="ns=4;i=5049">
    <DisplayName>Axis_AR2</DisplayName>
    <References>
      <Reference ReferenceType="i=40">ns=2;i=16601</Reference>
      <Reference ReferenceType="i=46">ns=4;i=6145</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5069</Reference>
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5049</Reference>
    </References>
  </UAObject>
  <UAObject NodeId="ns=4;i=5025" BrowseName="2:MotionDevices" ParentNodeId="ns=4;i=5001">
    <DisplayName>MotionDevices</DisplayName>
    <Description>Contains any kinematic or motion device which is part of the motion device system.</Description>
    <References>
      <Reference ReferenceType="i=40">i=61</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5044</Reference>
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5001</Reference>
    </References>
  </UAObject>
  <UAObject NodeId="ns=4;i=5044" BrowseName="2:MotionDeviceIdentifier_AR" ParentNodeId="ns=4;i=5025">
    <DisplayName>MotionDeviceIdentifier_AR</DisplayName>
    <References>
      <Reference ReferenceType="i=40">ns=2;i=1004</Reference>
      <Reference ReferenceType="i=46">ns=4;i=6108</Reference>
      <Reference ReferenceType="i=46">ns=4;i=6109</Reference>
      <Reference ReferenceType="i=46">ns=4;i=6110</Reference>
      <Reference ReferenceType="i=46">ns=4;i=6111</Reference>
      <Reference ReferenceType="i=46">ns=4;i=6112</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5049</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5050</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5051</Reference>
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5025</Reference>
    </References>
  </UAObject>
  <UAObject NodeId="ns=4;i=5049" BrowseName="2:Axes" ParentNodeId="ns=4;i=5044">
    <DisplayName>Axes</DisplayName>
    <Description>Axes is a container for one or more instances of the AxisType.</Description>
    <References>
      <Reference ReferenceType="i=40">i=61</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5055</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5061</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5065</Reference>
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5044</Reference>
    </References>
  </UAObject>
<UAObject NodeId="ns=4;i=5001" BrowseName="4:ArticulatedRobot" ParentNodeId="i=85">
    <DisplayName>ArticulatedRobot</DisplayName>
    <Description>A typical three-axis industrial articulated robot with 2
    power trains that moves 3 axes. Power train 1 moves axis 1, power train
    2 moves axis 2 and axis 3.</Description>
    <References>
      <Reference ReferenceType="i=40">ns=2;i=1002</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5024</Reference>
      <Reference ReferenceType="i=47">ns=4;i=5025</Reference>
```

```
      <Reference ReferenceType="i=47">ns=4;i=5042</Reference>
      <Reference ReferenceType="i=47">ns=4;i=6001</Reference>
      <Reference ReferenceType="i=47">ns=4;i=6002</Reference>
    </References>
  </UAObject>
  <UAObject NodeId="ns=4;i=6001" BrowseName="4:Speed" ParentNodeId="ns=4;i=5001">
    <DisplayName>Speed</DisplayName>
    <Description>Speed attribute of the articulated robot.</Description>
    <References>
      <Reference ReferenceType="i=40">i=61</Reference>
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5001</Reference>
    </References>
  </UAObject>
  <UAObject NodeId="ns=4;i=6002" BrowseName="4:Motor" ParentNodeId="ns=4;i=5001">
    <DisplayName>Motor</DisplayName>
    <Description>Motor attribute of the articulated robot.</Description>
    <References>
      <Reference ReferenceType="i=40">i=61</Reference>
      <Reference ReferenceType="i=47" IsForward="false">ns=4;i=5001</Reference>
    </References>
  </UAObject>
</UANodeSet>
```