

Context-Aware Internet-of-Things Communication via Retrieval-Augmented Generation

Franc Pouhela¹, Dennis Krummacker¹, Hans D. Schotten^{1,2}

¹*German Research Center for Artificial Intelligence (DFKI GmbH), Kaiserslautern*

Email: {franc.pouhela; dennis.krummacker; hans_dieter.schotten}@dfki.de

²*University of Kaiserslautern (RPTU), Germany*

Email: {schotten}@rptu.de

Abstract—A major challenge to the advancement of the Internet of Things (IoT) as we know it, is related to the need of a context-aware communication framework that can dynamically adapt to environments with constantly changing requirements. The paper at hand presents a potential avenue to address this limitation. The study proposes an innovative approach to enable context-awareness in dynamic IoT settings by leveraging the Natural Language Processing (NLP) capabilities of the Large Language Models (LLMs) in conjunction with the Knowledge Graphs (KGs) within a custom Retrieval-Augmented Generation (RAG) system to enable real-time understand and processing of the Natural Language (NL) instructions. This approach makes the communication system flexible and holds the potential for future innovations in the realm of IoT technologies.

Index Terms—IoT, RAG, LLM, AI, 6G

I. INTRODUCTION

The Internet of Things (IoT) has revolutionized modern communication systems by enabling dynamic connectivity between devices, sensors, and networks. However, one of the primary challenges facing the advancement of this technology is achieving a communication framework that is both efficient and capable of adapting to constantly changing environments and requirements. This challenge is amplified in large-scale IoT systems, where devices operate in dynamic conditions and often require context-aware real-time decision making.

The emergence of Large Language Models (LLMs), with their advanced Natural Language Processing (NLP) capabilities, has unlocked new possibilities for multiple industries. These models can generate code, database queries, and various other Natural Language (NL) outputs, which can be leveraged in diverse environments to enable dynamic actions. However, a critical limitation of LLMs is their tendency to hallucinate, introducing significant risks when applied to critical decision-making processes.

Recently, efforts have focused on addressing this limitation, with the integration of Knowledge Graphs (KGs) gaining significant attention. A KG is a structured representation of knowledge, modeling real-world entities, their attributes, and relationships as nodes and edges. This allows for efficient information storage, reasoning, inference, and retrieval by both humans and machines. In mathematical terms, a knowledge graph can be viewed as a directed graph $G = (V, E)$, where: V is the set of vertices (nodes), representing entities or concepts.

E is the set of edges that represent the relationships between the entities. Each edge in the graph is typically labeled with the type of relationship, providing semantic context. This makes KGs particularly useful in applications requiring context-aware reasoning, such as NLP, recommendation systems, etc., where they help encode domain-specific knowledge and facilitate decision-making based on the underlying semantic structure.

By integrating KGs with LLMs, the model can leverage accurate and contextually relevant information from KG, ensuring that its responses are based on real-world knowledge rather than speculative generation. This combination helps validate the generated content, providing factual consistency and reducing the chances of hallucination by supplementing the generative process with reliable, structured knowledge.

This paper presents a context-aware IoT framework that leverages the NLP capabilities of LLMs in conjunction with KGs through a custom Retrieval-Augmented Generation (RAG) technique that enables real-time understanding and processing of Natural Language Instructions (NLIs).

The remainder of this paper is organized as follows: Section II reviews related work relevant to the topic of this paper. Section III describes the communication system of the proposed framework, while Section IV details its architectural components. Section V focuses on the implementation and experimental evaluation. Finally, Section VI concludes the paper with a summary of findings and future research directions.

II. RELATED WORK

Context-aware IoT communication is a critical research area driven by the growing complexity of IoT networks and the demand for efficient, adaptable solutions. While many studies have focused on improving the scalability and efficiency of IoT communication systems, less progress has been made in developing communication systems with the ability to reason about their current state to make insightful decisions.

[1] explores the state-of-the-art in IoT, where physical objects, from everyday items to complex systems, connect and actively exchange information via the Internet. It includes a review of key technological drivers, potential applications, challenges, and future research areas in IoT.

The need for dynamically adapting to changing environments has been addressed in several works, such as in [2].

This study addresses the challenge of deriving value from the vast data generated by the rapidly growing number of IoT sensors, emphasizing the role of context-aware computing in understanding sensor data. It surveys context awareness in the IoT domain, offering an overview of the IoT paradigm and context-aware computing fundamentals.

The use of KGs in IoT systems has gained significant traction due to their ability to encode complex relationships and provide semantic context to data. Research by Maria Bermudez et al. [3] tackles the integration of Industrial Internet of Things (IIoT) data streams in Industry 4.0 through a semantic, process-centered approach. It develops an ontology to describe IIoT elements and constructs a Process-aware IIoT KGs that enriches sensor data with additional context.

A Retrieval-Augmented Generation (RAG) technique combines LLMs with retrieval mechanisms to enhance the reliability and accuracy of generated outputs. Although RAG has been successfully applied in various NLP tasks, its application to IoT communication is relatively new. A study by Lewis et al. [4] introduced RAG as a method to retrieve relevant information from external sources, which is particularly useful in dynamic and evolving environments such as IoT. This technique offers a way to overcome the limitations of hallucinations by grounding the model’s output in factual data.

By building on previous studies, we integrate RAG with KGs to create an innovative framework that has the potential to enhance context-awareness in IoT communication.

III. COMMUNICATION SYSTEM

Efficient and secure IoT communication depends heavily on the choice of messaging pattern and protocol. The publish/subscribe messaging pattern, exemplified by the widely adopted Message Queuing Telemetry Transport (MQTT)[5] protocol, has become the standard for IoT communication. In this study, we introduce a novel protocol named Middleware Message Queuing Protocol (MMQP)[6], inspired by MQTT. MMQP is a lightweight, binary protocol specifically designed for IoT and Machine-to-Machine (M2M) communication.

The communication pattern used in MMQP allows clients to communicate indirectly via a central distribution node known as the broker. Clients can act as both publishers and subscribers. Subscribers submit interest in specific topics by subscribing to them and publishers publish messages to these topics, which are named channels that categorize the message content. The broker distributes messages from publishers to the appropriate subscribers by managing topic hierarchies.

As illustrated in the sequence diagram depicted in Figure 1, the publisher establishes a connection with the broker by sending a CONREQ packet, to which the broker responds with a CONACK packet acknowledging the successful connection. Subsequently, the subscriber subscribes to a specific topic using the TOPREQ packet. Meanwhile, the publisher sends a PUBREQ (publish request) packet to the broker, which can be queued for further processing depending on the Quality of

Service (QoS) level of the message. The broker then forwards the relevant message to the subscribers.

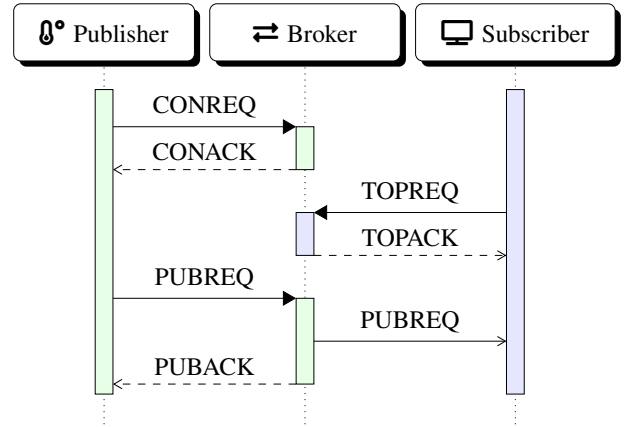


Fig. 1: MMQP Sequence Diagram

A. Quality of Service

MMQP supports three levels of QoS. Messages published to the broker with a QoS level greater than 0 can be queued until their expiry interval is reached. If new clients join, the broker will forward the queued messages accordingly to their subscription pattern.

QoS 0 (At most once): Or best effort delivery, ensures the delivery without any acknowledgment (Figure 2). If the broker is currently busy, the packet may be dropped. This QoS level is primarily used in high data rate scenarios, such as video streaming, where occasional packet loss is not critical.

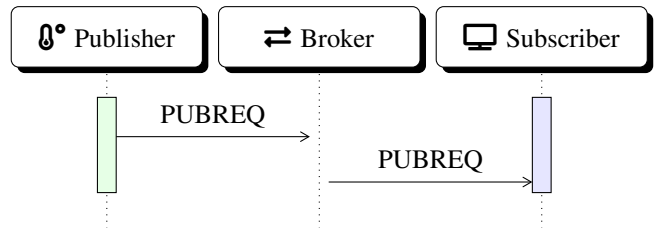


Fig. 2: Publishing with QoS level 0

QoS 1 (At least once): Here, the sender publishes the message, and the receiver (broker or subscriber) acknowledges its receipt. If the acknowledgment is not received, the publisher may resend the message (see Figure 3).

QoS 2 (Exactly once): This QoS level guarantees that the message is delivered exactly once. It involves a two-step process as depicted in (Figure 4).

In the sequence diagrams shown in Figures 2,3, and 4, it is assumed that the subscriber receives messages with QoS level 0. Consequently, no acknowledgment is sent from the subscriber to the broker. This assumption was made to optimize the use of space available in the paper.

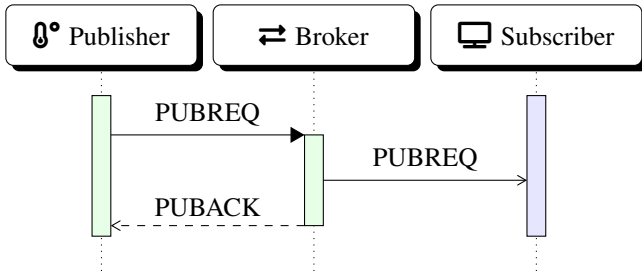


Fig. 3: Publishing with QoS level 1

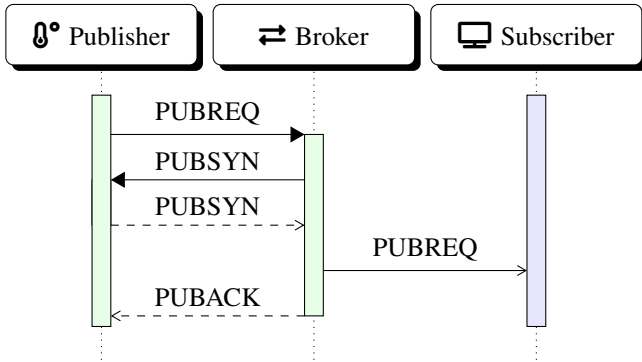


Fig. 4: Publishing with QoS level 2

B. Message Routing

Topics in MMQP function as key identifiers for organizing and distributing messages. These topics are managed hierarchically by brokers and are represented as strings with multiple levels separated by forward slashes (“/”). When a client publishes a message, the broker analyzes the topic hierarchy to identify the appropriate subscribers. The broker then forwards the message to subscribers who are interested in the exact topic or any of its parent levels. For example, publishing a message under *home/room1/light* would prompt the broker to deliver the message not only to subscribers of *home/room1/light* but also to those subscribed to *home/room1* and *home*. See [6] for a thorough description.

IV. ARCHITECTURAL COMPONENTS

The architecture of the proposed communication framework, shown in Figure 5, illustrates its key components. As previously discussed, the framework utilizes a central middleware referred to as the broker or a cluster of brokers to facilitate communication between multiple publishers and subscribers. To efficiently handle a high volume of transactions, the broker employs a service worker pool, enabling the concurrent processing of incoming and outgoing messages.

A. Software Architecture

By leveraging the Entity-Component-System (ECS) [7] design pattern, the broker can support a large number of sessions without significant performance degradation. ECS is a software design pattern frequently used in developing complex and scalable systems, particularly in the space of

game development. It enables code reuse by decoupling data from behavior. Entities typically consists of unique identifiers such as a 32-bit integer, Components model the behavior of entities and Systems, implement the behavior’s logic.

As illustrated in the architecture, the broker can extend its operational capabilities through optional modules. These modules are notified whenever specific events of interest are triggered, allowing them to react appropriately and perform their designated tasks. The Event Dispatcher component ensures the propagation of these events across different layers of the application. While our design showcases three modules, the architecture can support additional modules as needed. In order to interact with the KG and LLMs, modules can leverage the built-in, Python Application Programming Interface (API) that enables loading and running custom Python scripts.

The Knowledge Module is responsible for capturing both transactional and internal events from the broker to populate the KG with contextual data. This data can then be leveraged by the Reasoning Module to perform basic reasoning tasks. This can be crucial in scenarios where multiple brokers in a cluster collaborate to serve a large number of IoT devices. As the specific topics to which clients will subscribe are not known in advance, brokers would typically need to be interconnected to ensure message delivery across all brokers. However, by sharing the same KG, brokers can dynamically establish connections based on the context data provided in the KG, significantly reducing the overhead associated with managing multiple inter-broker connections.

The Scripting Module enables the generation of scripts from NLI provided by a system administrator. Each client connection to the broker is represented as an entity to which various components can be added. These scripts can be associated with specific clients to perform targeted actions, such as monitoring traffic behavior or analyzing subscription patterns. By embedding or referencing the Python API documentation as a system prompt, the LLM can generate scripts tailored to the specific needs of the system.

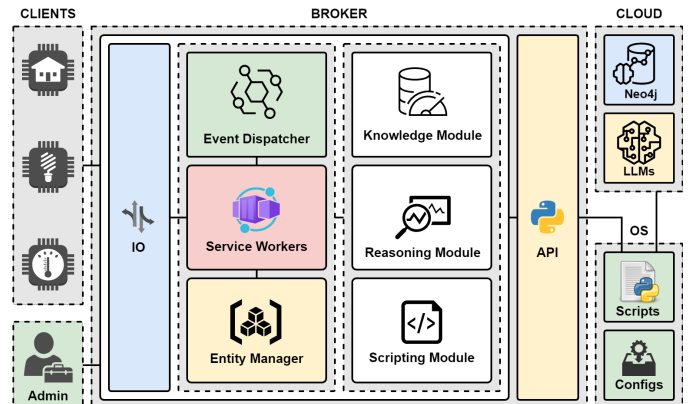


Fig. 5: Framework Architectural Components

B. Augmented Retrieval

As previously mentioned, this framework employs a custom RAG system to retrieve contextual data from a KG. RAG, or Retrieval-Augmented Generation, is a technique that integrates information retrieval with NL generation to enhance the response quality of LLMs. In such a system, a model retrieves relevant documents or data from external sources, such as a knowledge graph, and leverages this information to generate more accurate and informed responses, thereby reducing potential hallucinations.

Figure 6 illustrates the data flow of the RAG process. The process begins with an *Admin* or a client with administrative rights issuing an NL command to the system. This command could be a simple question, a reasoning task, or a request to generate a script for a specific task. For example, the admin may ask the question, "How many clients are currently online?" or request the creation of a script with the prompt: "Write a script to disconnect any client with a message rate higher than 10". The command is then combined with the appropriate context data, such as the KG ontology or the script API description to generate a Cypher Query (CQ) as depicted in the sequence. In this specific case, the LLM generates the CQ, which is then used by the Executor to query the KG. The result of the query is then given back to the LLM, along with some additional instructions on how to formulate the final answer, which is sent back to the requester.

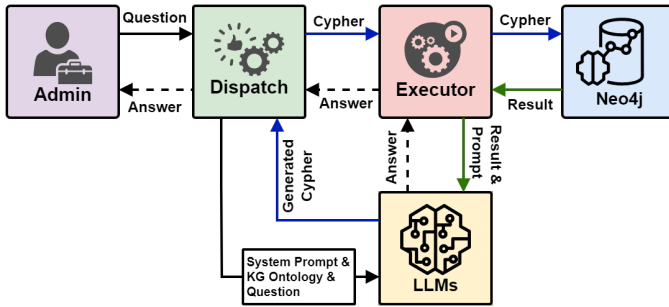


Fig. 6: Augmented Context Retrieval

V. IMPLEMENTATION AND EVALUATION

The proposed framework is implemented in C++17 and can run on both Windows and Linux Operating Systems (OSs). It uses Asio [8] to manage low level network operations. Asio is a cross-platform C++ library for network and low-level I/O programming that provides developers with a consistent asynchronous model using a modern C++ approach. It has a built-in task manager to concurrently handle multiple transactions.

To efficiently manage entities and their related components and systems, We integrated a well established ECS library known as: EnTT [9]. EnTT is a header-only, tiny and easy to use library for game programming and much more written in modern C++. The popular game Minecraft [10] uses EnTT to efficiently manage millions of game objects.

The integrated Python API is achieved with the help of pybind11 [11]. Pybind11 is a lightweight, header-only library that exposes C++ types to Python and vice versa, allowing the creation of Python bindings for existing C++ code. This enabled us to integrate the Python interpreter directly into the broker, allowing it to load, manage and execute Python scripts seamlessly at runtime.

The KG database used in the framework is called: Neo4j [12], which is a graph database management system designed to store and manage data in a graph structure. It is optimized for handling complex and highly connected data, enabling efficient querying and analysis of relationships between nodes. It uses the Cypher Query Language (CQL), a declarative query language, and is widely used in applications such as social networks, recommendation systems, etc.

Figure 7 depicts a KG from our study, illustrating relationships between brokers, clients, and topics. Each node has underlying attributes not visible in the graph such as brokers having IP addresses and ports, topics having names and identifiers. etc. A client's subscription to a topic is represented by the relationship: `IS_SUBSCRIBER_OF`, similarly, `IS_CONNECTED_TO` represents a client's connection to a broker. Relationships can also have additional attributes.



Fig. 7: Knowledge Graph Visualization

Our RAG setup uses Large Language Model Meta AI (LLaMa) [13] models to generate NL responses. LLaMa is a family of open-source LLMs developed by Meta for various NL tasks. These models are designed to generate human-like text, enabling applications such as text and code generation, summarization, Q&A, and more. They utilize the transformer architectures and are optimized for efficiency in terms of size and computational resources, making them suitable for both research and practical applications.

We successfully ran the models listed in Table I locally on an *11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz* with

32GB of RAM using Ollama [14]. Ollama is a platform for running and interacting with LLMs on local devices. It allows users to download and run models directly on their systems, enabling privacy, customization, and offline capabilities. It supports various LLMs and provides.

While any compatible models can be used, the ones selected are among the most widely adopted open-source options available. The only model parameter adjusted during our study was the *temperature*, set to 0.1. This parameter controls the randomness of the model’s outputs, where higher values result in more diverse and creative responses, while lower values produce more focused and deterministic outputs.

Model	Size	Length	Max. Context	Parameters
LLama3.1:8B	4.7GB	4096 tokens	128K tokens	8.0B
LLama3.2:3B	2.0GB	3072 tokens	128K tokens	3.2B
LLama3.2:1B	1.3GB	2048 tokens	128K tokens	1.2B

TABLE I: Models properties

A. Cypher System Prompt

Listing 1 presents the system prompt utilized during our evaluation of the RAG system to generate CQs based on NL instructions. This enables the models to grasp the context of the question and produce queries based on the properties defined in the KG description. This system prompt contains approximately 2711 characters, which is way below the maximal context length of all our models described in Table I. It is also possible to retrieve the KG schema directly from Neo4j, eliminating the need to write it manually.

```

You are an expert in generating Cypher queries for Neo4j knowledge graphs. Your task is to generate precise Cypher queries based strictly on the knowledge graph schema and the instruction provided below.

## STRICT INSTRUCTIONS TO FOLLOW:
1. Do not infer or introduce any nodes, attributes, or relationships not explicitly described in the knowledge graph schema.
2. Only Cypher query output|no explanations, newline characters, special characters, comments, or additional text.
3. Never include additional characters such as newlines, codeblock format characters, etc., only the cypher code.
4. Ensure that any query deleting a node detaches all relationships associated with that node.
5. Ensure that the query is fully grounded in the knowledge graph schema and the provided instruction.
6. If you cannot generate a query, return an empty string.
7. You can get inspiration from the provided examples

### NODE TYPES:
- Broker: Represents a server that manages client connections and message routing.
  - Attributes: 'name', 'ip', 'tcp_port', 'ws_port'.
- Client: Represents a client connecting to a broker to send or receive messages.
  - Attributes: 'name', 'uid', 'ip', 'port'.
- Topic: Represents a message channel for publishing or subscribing.
  - Attributes: 'name', 'uid', 'nbr_publishers', 'nbr_subscribers'.
- Group: Represents a collection of clients subscribed to a common topic.

```

```

- Attributes: 'name', 'nbr_members'.

### NODE RELATIONSHIPS:
- `(Client)-[:IS_CONNECTED_TO]->(Broker)`: A client connects to a broker.
- `(Client)-[:IS_SUBSCRIBER_OF]->(Topic)`: A client subscribes to a topic.
- `(Client)-[:IS_PUBLISHER_OF]->(Topic)`: A client publishes to a topic.
- `(Client)-[:IS_MEMBER_OF]->(Group)`: A client belongs to a group.
- `(Topic)-[:HAS_GROUP]->(Group)`: A topic has an associated group.
- `(Topic)-[:IS_PARENT_OF]->(Topic)`: A parent-child relationship between topics.

## EXAMPLES OF OUTPUTS:
- Question: Retrieve all clients connected to the broker hosted at the IP address 192.168.1.1.
- Output: MATCH (c:Client)-[:IS_CONNECTED_TO]->(b:Broker {ip: "192.168.1.1"}) RETURN c

- Question: Find all topics a client with the id "7463644344" is subscribed to.
- Output: MATCH (c:Client {uid: "Client1"})-[:IS_SUBSCRIBER_OF]->(t:Topic) RETURN t

- Question: List all groups associated with the topic named "TopicA".
- Output: MATCH (t:Topic {name: "TopicA"})-[:HAS_GROUP]->(g:Group) RETURN g

- Question: Retrieve all clients who belong to the group named "GroupAlpha".
- Output: MATCH (c:Client)-[:IS_MEMBER_OF]->(g:Group {name: "GroupAlpha"}) RETURN c

Based strictly on the provided instructions and the graph description above, generate the Cypher query for this instruction:

```

Listing 1: Cypher System Prompt

B. Language Model Program

The code in Listing 2 is an example of a Language Model Program (LMP). A LMP is a structured approach to building applications that utilize LLMs for specific tasks. This approach harnesses the capabilities of language models in a way that aligns with the intended application behavior.

```

import EmpyMQ as eMQ

@eMQ.Tool()
def RunQuery(self, cypher):
    return self.Graph.run(cypher).data()

@eMQ.Action(model="llama3.2:latest", temperature=0.01)
def GenerateCypher(self, question):
    return [
        eMQ.System(self.Prompt1),
        eMQ.User(f"{question}")
    ]

@eMQ.Chain(model="llama3.2:latest", temperature=0.01,
           tools=[RunQuery])
def GenerateAnswer(self, cypher, question):
    return [
        eMQ.System(self.Prompt2),
        eMQ.User(f"{question}")
    ]

def OnAttach(self):
    self.Prompt1 = eMQ.LoadFile("Prompts/Prompt1.pt")
    self.Prompt2 = eMQ.LoadFile("Prompts/Prompt2.pt")

```

Listing 2: Language Model Programming

With this framework, we can implement regular functions as usual, while internally integrating calls to various language models. In addition, functions can be equipped with tools that the model can invoke to retrieve additional information required for its response. To streamline this process, we developed a custom module (*EmptyMQ*) combining our C++ application and a Python module known as **ell** [15].

The `GenerateCypher(...)` function generates a Cypher query based on the provided system prompt and the user’s question. System prompts are loaded from external files and stored as attributes, ensuring that they are readily available when needed. The `GenerateAnswer(...)` function then formulates a response using the results obtained from the `RunQuery(...)` tool, which executes the generated Cypher query. These interactions are triggered by C++.

C. Results Evaluation

Table II presents a set of questions and their corresponding responses, which were used to evaluate the models’ ability to generate Cypher queries based on a knowledge graph schema provided in the system prompt. The results reveal that `LLama3.1:8B` performed exceptionally, achieving a perfect score by correctly answering the 5 questions. In comparison, `LLama3.2:3B` managed to answer only 2 out of 5 correctly. The smallest model, `LLama3.2:1B`, demonstrated the worst performance with a score of 0. (Figure 8).

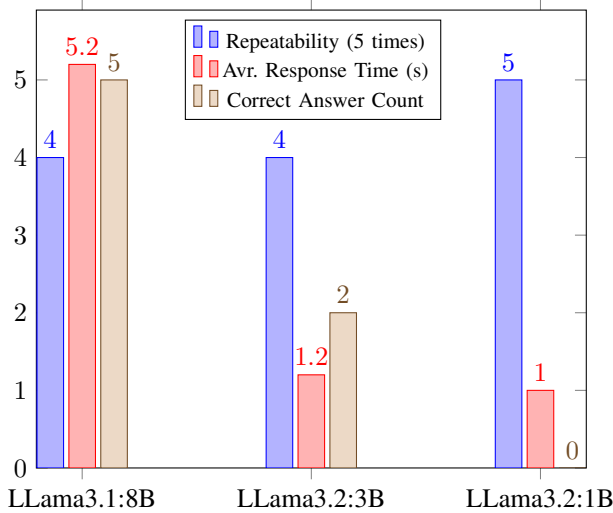


Fig. 8: Evaluation Results

During our study, we observed the importance of properly formulating system prompts. Smaller models often struggled to grasp the dynamics of relationships between different nodes. For example, they frequently misrepresented the direction of the relationship between a client and a topic, incorrectly assigning a topic as a subscriber or publisher to a client, rather than recognizing that the reverse is the only valid relationship. We also recognized that the formulation of the questions played an important role. The model sometimes misinterpreted the order of information, leading to inverted results. For

example, asking, *How many clients are connected to the broker with the IP address 192.168.2.177?* could confuse the model, leaving it unclear whether the IP address referred to the clients or the broker. A clearer prompt, such as *How many clients are connected to the broker hosted at IP address 192.168.2.177?*, would help avoid such confusion.

Our evaluation also included a repeatability test designed to assess the consistency of the models’ responses, regardless of their correctness. This test is crucial because we expect the model to maintain a certain level of consistency across multiple runs. With the model temperature set to a low value of 0.1, we anticipated that the models would produce consistent outputs. This expectation was met, as shown in Figure 8, where all models achieved a relatively high consistency score. Surprisingly, the smallest model proved to be the most consistent in this test. It is important to note that while the results produced by the largest models slightly varied, the differences were still minor and did not affect their correctness. This highlights the dynamic nature of these models and how their behavior can change with different configurations. The largest model `LLama3.1:8B` took considerably longer to generate answers but also demonstrated the highest level of consistency.

On the question of hallucination, the smallest model, `LLama3.2:1B`, was the only one to generate non-existent relationships or attributes not defined in the KG schema provided in the prompt. Concerning compliance with the instructions in the system prompt, the two smaller models occasionally included special characters, such as ‘`’ or similar, around the generated Cypher queries. In contrast, the largest model, `LLama3.1:8B`, consistently adhered to all instructions.

VI. CONCLUSION AND FUTURE WORK

This study presents an architectural and practical framework for incorporating context awareness into a publish/subscribe IoT communication system. The proposed approach integrates the NLP capabilities of LLMs within a custom RAG system to access and utilize context data in subsequent system stages for specific tasks.

The evaluation of the RAG approach reveals that the larger model, `LLama3.1:8B`, outperforms the smaller models but requires significantly more time to generate responses. It also underscores the challenges of crafting effective system prompts to elicit accurate outputs from models. Additionally, the repeatability tests highlight the necessity of validating model consistency, even when their answers are incorrect, as a key measure of reliability. An important observation from the study pertains to the hallucination issue shows that one model generated non-existent information. This highlights a limitation of small models with limited internal knowledge. As LLMs continue to evolve, their ability to handle NL and reason effectively is expected to improve considerably.

In conclusion, this work provides valuable insights into developing robust, context-aware communication systems. Leveraging fine-tuning or adapter-based training methods could enhance the model’s ability to handle domain-specific tasks

Question 1:	How many clients are connected to the broker hosted at IP address 192.168.2.177?	
LLama3.1:8B	<code>MATCH (c:Client)-[:IS_CONNECTED_TO]->(b:Broker ip: "192.168.2.177") RETURN COUNT(c)</code>	✓
LLama3.2:3B	<code>MATCH (b:Broker ip: "192.168.2.177")-[:HAS_CLIENT]->(c) RETURN c COUNT(c)</code>	✗
LLama3.2:1B	<code>MATCH (b:Broker ip: "192.168.2.177")-[:IS_CONNECTED_TO]->(c:Client) RETURN count(c)</code>	✗
Question 2:	List all topics and their associated groups for which the group has more than 10 members.	
LLama3.1:8B	<code>MATCH (t:Topic)-[:HAS_GROUP]->(g:Group) WHERE g.nbr_members > 10 RETURN t, g</code>	✓
LLama3.2:3B	<code>MATCH (t:Topic)-[:HAS_GROUP]->(g:Group) WHERE g.nbr_members > 10 RETURN t, g</code>	✓
LLama3.2:1B	<code>MATCH (g:Group)-[:HAS_GROUP]->(t:Topic) WHERE t.id > "0" AND g.nbr_members > 10 RETURN t.name, g.nbr_members</code>	✗
Question 3:	Delete the topic with the name "DeprecatedTopic" and detach all relationships associated with it.	
LLama3.1:8B	<code>MATCH (t:Topic name: "DeprecatedTopic") DETACH DELETE t</code>	✓
LLama3.2:3B	<code>MATCH (t:Topic name: "DeprecatedTopic") DETACH DATA OPTIONAL MATCH (t)-[r]-[n] SET [...]</code>	✗
LLama3.2:1B	<code>MATCH (t:Topic name: "DeprecatedTopic")-[:HAS_GROUP]->(g:Group) DETACH DELETE [...]</code>	✗
Question 4:	Count the number of clients in each group for all groups associated with the topic "TopicZ".	
LLama3.1:8B	<code>MATCH (t:Topic name: "TopicZ")-[:HAS_GROUP]->(g:Group) OPTIONAL MATCH (c:Client)-[:IS_MEMBER_OF]->(g) RETURN g.name AS groupName, COUNT(c) AS clientCount</code>	✓
LLama3.2:3B	<code>MATCH (t:Topic name: "TopicZ")-[:HAS_GROUP]->(g:Group) WITH g, COUNT(DISTINCT c) AS nbr_members RETURN g.name, nbr_members</code>	✗
LLama3.2:1B	<code>MATCH (g:Group)-[:HAS_GROUP]->(t:Topic name: "TopicZ")-[:IS_PARENT_OF]*->(pg:Group) [...]</code>	✗
Question 5:	List all parent-child topic pairs where the parent has more than 5 publishers and the child has fewer than 3 subscribers.	
LLama3.1:8B	<code>MATCH (p:Topic nbr_publishers: gt: 5)-[:IS_PARENT_OF]->(c:Topic nbr_subscribers: lt: 3) RETURN p, c</code>	✓
LLama3.2:3B	<code>MATCH (p:Topic)-[:IS_PARENT_OF]->(c:Topic) WHERE p.nbr_publishers > 5 AND c.nbr_subscribers < 3 RETURN p, c</code>	✓
LLama3.2:1B	<code>MATCH (t:Topic)-[:HAS_GROUP]->(g:Group) WHERE g.nbr_members < 3 AND t.nbr_publishers > 5 RETURN t.name, g.name</code>	✗

TABLE II: Cypher Queries Generation Results

with greater accuracy. Future studies could also investigate the scalability of the framework in large-scale IoT systems, examining its performance under diverse workloads and real-time constraints. Ultimately, integrating more advanced reasoning capabilities and incorporating user feedback mechanisms will further improve the system’s adaptability and reliability. By addressing these challenges, the proposed approach can pave the way for highly efficient and intelligent context-aware communication systems in IoT environments.

ACKNOWLEDGMENT

The authors acknowledge the financial support by the German Federal Ministry for Education and Research (BMBF) within the projects Open6GHub {16KISK003K} & 6G-Terafactory {16KISK186}.

REFERENCES

[1] Bandyopadhyay, S. and Sen, J., “Internet of things: Applications and challenges in technology and standardization,” *Wireless personal communications*, vol. 58, no. 1, pp. 49–69, 2011.

[2] Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D., “Context aware computing for the internet of things: A survey,” *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[3] Bermudez-Edo, M., Elsaleh, T., Barnaghi, P., and Taylor, K., “Iot-lite: a lightweight semantic model for the internet of things and its use with dynamic semantics,” *Personal and Ubiquitous Computing*, vol. 21, pp. 475–487, 2017.

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

[5] OASIS. Mqtt version 5.0, oasis standard. [Online]. Available: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>

[6] Pouhela, F., Sanon, S. P., Krummacker, D., and Schotten, H. D., “Everything Interconnected via Cyberspace,” in *MMQP: A Lightweight, Secure and Scalable IoT Communication Protocol*, IEEE. IEEE, 8 2024. [Online]. Available: https://www.researchgate.net/publication/381742313_MMQP_A_Lightweight_Secure_and_Scalable_IoT_Communication_Protocol

[7] Pouhela, F., Krummacker, D., and Schotten, H. D., “Entity component system architecture for scalable, modular, and power-efficient iot-brokers,” in *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*, 2023, pp. 1–6. [Online]. Available: https://www.researchgate.net/publication/373318000_Entity_Component_System_Architecture_for_Scalable_Modular_and_Power-Efficient_IoT-Brokers

[8] Asio c++ library. [Online]. Available: <https://think-async.com/Asio/>

[9] Entt. [Online]. Available: <https://github.com/skypjack/entt>

[10] Minecraft. Minecraft. [Online]. Available: <https://www.minecraft.net/en-us>

[11] pybind11 community. pybind11. [Online]. Available: <https://github.com/pybind/pybind11>

[12] Neo4j. Genai apps, grounded in your data. [Online]. Available: <https://neo4j.com/>

[13] AI, M. Introducing llama 3.2. [Online]. Available: <https://www.llama.com/>

[14] Team, O. Get up and running with large language models. [Online]. Available: <https://ollama.com/>

[15] ell. The language model programming library. [Online]. Available: <https://docs.ell.so/>