Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.11x9/ACCESS.2024.0429xxxx

A Modular RAG Framework for Translating Natural Language to Domain-Specific Language Queries

PRAGYAN DAS[1], NITISH KUMAR JHA[1][2], KOUSHIK ROUT[1], MEENAKSHI S.P.[3], KALAVATHI B[3], HARSHITH REDDY AENUGU[3], SUBIN KUMAR[3], ARYAN SINGH BISHT[3], and JAHNAVI MAJUMDER[3]

[1]Samsung R&D Institute India, Samsung, Bangalore, India (e-mail: pragyan.das@samsung.com; nitish.jha@samsung.com; k.rout@samsung.com)
[2]Optum (e-mail: nitish.jha@optum.com)
[3]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India (e-mail: spmeenakshi@vit.ac.in; kalaavathi.b@vit.ac.in; harshithreddyaenugu@gmail.com; subin.kumar2022@vitstudent.ac.in; aryansingh.bisht2022@vitstudent.ac.in; jahnavimajumder7@gmail.com)

Samsung R&D Institute India and Vellore Institute of Technology (VIT), Vellore.

ABSTRACT As AI systems increasingly interface with backend monitoring and search infrastructure, the need to translate natural language instructions into structured queries (e.g., PromQL, Elasticsearch DSL, Datadog Query Language) is growing rapidly. In this study, we propose a generalized RAG framework capable of generating such queries across domains. Our solution leverages large language models augmented with adaptive retrieval pipelines to generate precise and semantically valid queries from natural language instructions.
Building upon recent advances in RAG efficiency and agentic adaptability, our methodology introduces a modular framework combining retriever embeddings, contextual compression, and prompt orchestration to support multi-domain applicability. We implement and evaluate multiple variations of the framework using different vector databases, embedding models, and retrieval strategies. Experimental results on PromQL show that our best-performing configuration achieves 83.33% syntactic validity, outperforming non-RAG LLM baselines (e.g., LLaMA 3.1 8B at 78.49% and LLaMA 3.3 70B at 68.28%).
This work contributes to the future of natural language interface systems by demonstrating a robust, extensible, and modular RAG framework capable of translating natural language to executable DSL queries with measurable performance gains.

INDEX TERMS Retrieval-Augmented Generation (RAG), PromQL, Natural Language Processing (NLP), Structured Query Generation, Large Language Models (LLMs).

I. INTRODUCTION

THE translation of natural language (NL) instructions into structured query languages such as PromQL, OpenSearch DSL, and Datadog Query Language etc. has become increasingly critical in the era of AI-driven observability and monitoring. Users often express analytical or diagnostic intents through unstructured commands, which must then be transformed into valid, executable queries to interact with backend systems. Traditional approaches, including rule-based systems and pattern matching, lack generalizability and cannot handle ambiguous or multi-intent inputs. Conversely, large language models (LLMs) offer fluent and flexible understanding of language but are prone to hallucinations and ungrounded outputs, particularly when domain-specific knowledge is required.

Retrieval-Augmented Generation (RAG) is an emerging paradigm that augments LLMs with a dedicated retrieval step. In a typical RAG setup, the user's input query is first used to retrieve semantically relevant documents from a knowledge base—such as API documentation, metric definitions, or system logs. These documents are then fed into the language model alongside the input, enabling the model to generate outputs that are better grounded, more accurate, and contextually aware. RAG thereby serves as a bridge between general-purpose language understanding and domain-specific information retrieval.

In the context of NL-to-DSL translation, RAG offers significant advantages. First, it allows the system to reference canonical documentation for syntax and semantics, reducing the likelihood of invalid query construction. Second, by retrieving real usage examples, RAG enables better intent disambiguation. Third, it opens opportunities for multi-domain extensibility—where the same architecture can adapt to different query languages by swapping out domain-specific retrievers and prompt templates.

However, designing an effective RAG system for DSL generation poses unique challenges: (i) the retrieved context must align with low-level syntax rules, (ii) the LLM must be guided to produce deterministic structured outputs, and (iii) evaluation requires metrics that go beyond language fluency to measure syntactic correctness and semantic grounding. This work addresses these challenges by proposing a modular RAG architecture optimized for DSL query synthesis from natural language inputs, with a specific focus on PromQL as the target language.

II. LITERATURE REVIEW

Retrieval-Augmented Generation (RAG) has gained prominence as a hybrid approach that integrates large language models (LLMs) with external knowledge retrieval to enhance generation quality and factual grounding. Initially developed to improve open-domain question answering, RAG architectures have evolved significantly to address more structured and domain-specific generation tasks. This section reviews the progression of RAG in both general and domain-specific contexts, and highlights unresolved issues relevant to translating natural language (NL) into executable domain-specific languages (DSLs) such as PromQL.

A. GENERAL ADVANCES IN RAG DESIGN AND EFFICIENCY

The foundational work by Şakar and Emekci [1] outlines how the choice of retrievers, embeddings, and context compression significantly affects retrieval latency and generation accuracy. Other contributions such as Efficient RAG [8] introduce lightweight verification layers to ensure low-latency outputs without compromising validity—essential for real-time structured query systems. However, these studies are predominantly benchmarked on open-ended QA tasks, not constrained, syntax-sensitive DSL outputs.

B. CONTEXT-AWARE RETRIEVAL AND REASONING

Recent efforts have focused on improving reasoning through agentic and multi-hop retrieval strategies. Singh et al. [2] proposed Agentic RAG, where autonomous retriever agents handle planning and feedback, while MultiHop-RAG [14] demonstrated that multi-step retrieval improves compositional query generation. Despite success in reasoning-heavy domains, these architectures remain largely unexplored for structured queries where exact syntax and semantics must be preserved.

C. HANDLING HALLUCINATION AND RETRIEVAL FAILURES

Mindful-RAG [3] and REAPER [9] identify critical RAG failure modes—hallucination due to irrelevant or weakly linked context, and inconsistent outputs caused by poorly aligned retrieval. While their frameworks introduce diagnostic tools and probabilistic retrieval planning, they do not specifically address syntax verification for DSLs or metrics for structured query correctness.

D. DOMAIN-SPECIFIC EXTENSIONS AND HYBRID RETRIEVAL

Blended-RAG [5] and the four-module architecture by Shi et al. [15] combine sparse and dense retrievals to improve semantic precision. These hybrid techniques significantly reduce hallucination in API generation tasks, suggesting potential applicability to DSL generation. Yet, their performance is not evaluated on strict syntactic formats like PromQL or OpenSearch.

E. PROMPTING AND PLANNING FOR STRUCTURED GENERATION

Chain-of-Thought (CoT) prompting [12] and its extension with knowledge graphs (KAM-CoT [13]) improve reasoning by decomposing generation into intermediate steps. These approaches demonstrate high accuracy in ScienceQA and symbolic reasoning, but face limitations when transferred to DSLs that require deterministic grammar and operator precision. Plan-with-Code [17] confirms that CoT can reduce function hallucinations in DSLs, though evaluation remains limited to robotics and automation scripts.

F. OBSERVATIONS FROM MULTI-MODAL AND EMBEDDED DOCUMENT CONTEXTS

Several works have explored extracting structured insights from diverse formats—e.g., S3LLM [16] for scientific software, and Tripathy et al. [25] for real-world audio environments. While they highlight the role of robust feature engineering and context curation, they do not focus on query generation. Similarly, hybrid generation methods [7], [11] show gains in code education, but lack adaptation mechanisms for DSL-specific syntax.

G. EVALUATION GAPS AND LIMITATIONS IN LITERATURE

Although tools like eRAG [23] and Megalodon [18] advance long-context evaluation and Kendall-based correlation metrics, current literature lacks robust, fine-grained evaluation tailored to structured query tasks. Most models are scored on answer similarity (e.g., ROUGE) rather than syntactic correctness, semantic alignment, or execution validity—all essential for PromQL or Datadog query generation.

H. RESEARCH GAP AND PROBLEM STATEMENT

In summary, existing literature has made significant strides in hybrid retrieval, agentic planning, and prompt design. However, these solutions primarily target open-domain or loosely structured generation tasks. They do not fully address the challenge of translating NL to executable DSLs where:

- strict syntax must be enforced,

- hallucination can lead to runtime failures,
- retrieval must align with formal language specifications,
- and evaluation must account for both syntax and execution validity.

This gap motivates our work.

Problem Statement: *How can we design a modular and extensible RAG architecture that accurately translates natural language into executable DSL queries—while minimizing hallucination, improving syntax validity, and supporting generalization across multiple observability platforms?*

I. KEY INFERENCES GUIDING OUR APPROACH

1) **Hybrid retrieval strategies** (dense + sparse) enhance grounding and reduce hallucination [5], [15].
2) **Agentic RAG systems** improve planning in multi-step reasoning tasks but need adaptation for syntax-sensitive DSLs [2], [14].
3) **Syntax correction pipelines** post-generation are essential to enforce executable formats [3], [9].
4) **Prompt orchestration** like CoT and code-guided planning improve structured reasoning but require integration with verification mechanisms [12], [13], [17].

These inferences collectively support the design of an advanced RAG pipeline tailored for NL-to-DSL translation in observability and search platforms.

III. METHODOLOGY

A. OVERVIEW OF METHODOLOGY

Our research aims to construct and refine a Retrieval-Augmented Generation system that efficiently converts natural language instructions into structured PromQL queries. This work builds upon the agentic RAG paradigm proposed by Singh et al. [2], where retrieval is handled through autonomous, planning-capable components. Our methodology encompasses comprehensive data sourcing, extraction techniques, multiple RAG implementations, and specialized pipelines for syntax verification and correction, addressing the common failure points identified in contemporary RAG systems [3].

B. DATA SOURCES AND EXTRACTION

1) Prometheus Documentation

Our primary reference source for generating accurate PromQL queries was the official Prometheus documentation, which we accessed through two complementary approaches:

Web Scraping with BeautifulSoup: For certain system iterations, we directly scraped the Prometheus documentation website using Python's BeautifulSoup library. This approach aligns with the intelligent extraction methods described in recent literature for parsing heterogeneous documents [6], allowing us to dynamically capture the latest documentation changes.

Manual Extraction and PDF Conversion: Alternative versions used manually extracted documentation converted to PDF format. This approach created a stable documentation snapshot that could be reliably indexed and retrieved, implementing the document processing principles outlined by Şakar and Emekci [1] for maximizing retrieval precision.

2) Sample Datasets

Kaggle Sample Dataset: We utilized a Kaggle dataset containing paired natural language queries and corresponding PromQL outputs, providing diverse query examples ranging from simple to complex scenarios.

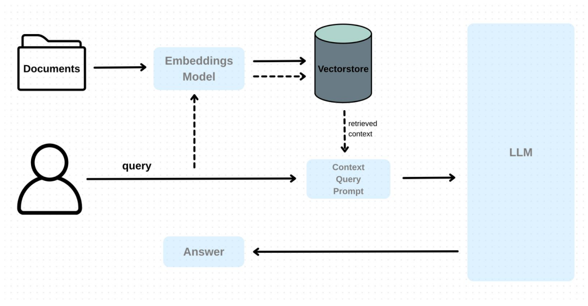

FIGURE 1. **Overview of the RAG Pipeline**

Synthetic and Domain-Specific Queries: We supplemented the Kaggle dataset with synthetic queries focusing on metric names and descriptions. This approach implements the hybrid semantic search concepts discussed in the Blended-RAG literature [5], enhancing context precision for domain-specific applications.

3) Document Chunking

Efficient processing of large documents required implementing a strategic chunking approach, following the modular architecture principles for RAG efficiency proposed by Şakar and Emekci [1]:

Purpose: Large documents such as comprehensive Prometheus documentation were divided into manageable segments to facilitate precise vectorization and similarity-based retrieval.

Method: We employed text splitting techniques with predefined chunk sizes and overlaps between consecutive chunks, preserving contextual continuity at chunk boundaries.

Benefits: This approach enhanced processing speed and retrieval accuracy by ensuring the most relevant document sections were available during query generation, addressing the context mismatch issues identified in the Mindful-RAG study [3].

For instance, PDF documentation was divided into 1000-character chunks with 200-character overlaps, ensuring that information spanning multiple chunks maintained necessary context for accurate retrieval.

C. SYSTEM ARCHITECTURE AND COMPONENT VARIATIONS

1) Language Models (LLMs)

We experimented with different LLMs to assess their impact on query interpretation and generation:

Llama-3.1-8b-instant: Used in multiple versions (rag_v1, rag_v2, rag_v5, rag_v6, rag_v7).

Mistral-7B-Instruct-v0.3: Implemented in rag_v3 and rag_v4.

Llama-3.3-70b-versatile: Employed in rag_v8 for enhanced understanding of nuanced queries.

Our LLM selection strategy considered the findings of Wei et al. [12] regarding model size correlation with reasoning performance, particularly for structured outputs requiring logical composition.

2) Databases for Document Storage

Two database systems were compared for storing extracted documentation and datasets:

Chroma: Utilized in versions such as rag_v1, rag_v3, and the agentic_rag variant.

Milvus Lite: - Standard Configuration: Used in rag_v2, rag_v4, and rag_v5. - **Multi-Collection Setup:** Adopted in rag_v6, rag_v7, and rag_v8, allowing for separate collections (e.g., one for scraped documentation, one for PDF-based documentation, and one for sample datasets).

This database implementation follows the recommendations from Şakar and Emekci [1] regarding vectorstore choices that critically influence latency and retrieval precision.

3) Embedding Models

To convert text data into vectors for similarity-based retrieval, we experimented with:

all-MiniLM-L6-v2: Employed when using Chroma.

BAAI/bge-m3: Used in combination with Milvus Lite.

These models were selected based on their balance of performance and computational efficiency, following the embeddings comparative analysis presented by Şakar and Emekci [1].

4) Retrieval Mechanisms

Different retrieval techniques were evaluated based on the comparative findings in recent RAG literature:

Similarity-Based Retrieval: Versions like rag_v1, rag_v3, rag_v7, and rag_v8 rely on cosine similarity measures between query and document embeddings.

Dense Retrieval: Implemented in rag_v2 and rag_v6, focusing on richer semantic matching.

Hybrid Retrieval (Dense + Sparse): Versions rag_v4 and rag_v5 combine dense retrieval with keyword-based sparse techniques to better handle ambiguous queries, implementing the hybrid approach advocated in the Blended-RAG research [5] for improving retrieval accuracy.

5) Prompt Strategies

Standard Prompting: Most versions use straightforward prompts where the natural language query is directly mapped to PromQL.

Chain-of-Thought (CoT) Prompting: For versions rag_v2, rag_v5, and rag_v6, we integrated a step-by-step reasoning process following the methodology introduced by Wei et al. [12]. This intermediate reasoning layer was designed to improve both syntax and semantic accuracy by breaking down complex queries into logical steps, which has been shown to significantly boost performance on multi-step reasoning tasks, especially with larger models.

6) Syntax Correction Pipelines

To address issues with PromQL syntax generation, we implemented additional LLM passes with a custom prompt, addressing the failure points identified in the Mindful-RAG study [3] related to inconsistent answer formats. These prompts are specifically designed to fix common syntax errors, such as scalar and vector metric mismatches where functions expect one or the other, as well as issues like missing braces or incorrect operators. In our post-processing iterations:

rag_v7a: Takes the output of rag_v7 and applies an LLM-based correction pass with the custom syntax-fix prompt.

rag_v8a: Similarly processes outputs from rag_v8 using the same targeted LLM-based approach.

This method ensures that common PromQL syntax errors are systematically addressed through an additional reasoning step, resulting in more reliable and executable queries, following the diagnostic framework for identifying and addressing RAG failure points [3].

D. AGENTIC RAG IMPLEMENTATION

The agentic_rag approach enhances query generation by integrating a React Agent that dynamically selects specialized tools based on the query context, building directly on the agentic paradigm proposed by Singh et al. [2]. This modular design allows the system to address different aspects of natural language to PromQL conversion through dynamic context planning, iterative query refinement, and reflective feedback mechanisms. We have developed two versions that differ mainly in the toolset they employ:

agentic_rag: This version coordinates the following tools: - **CSV Tool:** Processes tabular data, specifically leveraging the Kaggle dataset containing sample natural language queries paired with their corresponding PromQL outputs. - **Documentation Reference (docs_tool):** Provides syntax and function details from official references, guiding operator usage and function parameters. - **Metric Resolver (metrics_tool):** Identifies appropriate metric names and measurements mentioned in a query. - **PromQL Query Generator (query_generation_tool):** Directly converts well-specified requirements into PromQL after consulting documentation and metric references.

agentic_rag_v2: In this enhanced version, the toolset is refined to improve query accuracy: - **Documentation Reference (docs_tool) - Metric Resolver (metrics_tool) - PromQL Query Generator (query_generation_tool) - PromQL Syntax Correction Tool** (promql_syntax _correction_tool): Refines generated queries by addressing common syntax issues, such as scalar and vector mismatches, ensuring

TABLE 1. Comparative Analysis of RAG Versions Implemented in This Study

Version	LLM	Database	Embedding Model	Retrieval
rag_v1	Llama-3.1-8b-instant	Chroma	all-MiniLM-L6-v2	Similarity
rag_v2	Llama-3.1-8b-instant	Milvus Lite	BAAI/bge-m3	Dense
rag_v3	Mistral-7B-Instruct-v0.3	Chroma	all-MiniLM-L6-v2	Similarity
rag_v4	Mistral-7B-Instruct-v0.3	Milvus Lite	BAAI/bge-m3	Hybrid
rag_v5	Llama-3.1-8b-instant	Milvus Lite	BAAI/bge-m3	Hybrid
rag_v6	Llama-3.1-8b-instant	Milvus Lite (multi)	BAAI/bge-m3	Dense
rag_v7	Llama-3.1-8b-instant	Milvus Lite (multi)	BAAI/bge-m3	Dense
rag_v8	Llama-3.3-70b-versatile	Milvus Lite (multi)	BAAI/bge-m3	Dense

TABLE 2. Agentic RAG Implementation Variations

Version	LLM	Database	Embedding Model	Tools Used
agentic_rag	Llama-3.1-8b instant	Chroma	all-MiniLM-L6-v2	csv_tool, metrics_tool, docs_tool, query_generation_tool
agentic_rag_v2	Llama-3.3-70b versatile	Milvus Lite (multi)	BAAI/bge-m3	docs_tool, metrics_tool, query_generation_tool, promql_syntax_correction_tool

more reliable outputs, following the context stitching principles from the Hybrid Text Generation-Based RAG model [7].

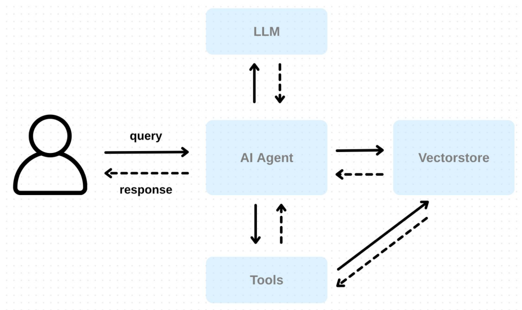

FIGURE 2. Overview of the Agentic RAG Pipeline

By dynamically selecting the appropriate tools for each step of the query generation process, both versions of agentic_rag improve overall performance, particularly when handling complex or ambiguous queries. This approach aligns with the query rewriting layer described in the intelligent extraction research [6], which transforms vague user prompts into structured intermediate representations.

E. IMPLEMENTATION DETAILS AND TESTING INFRASTRUCTURE

The implementation of the proposed RAG-based architecture was carried out using modular components to ensure flexibility and scalability across different query domains. Document processing was handled using the BeautifulSoup4 library for HTML scraping and PyPDF2 for extracting text from PDF documents. For vector storage, both Chroma and Milvus Lite were employed, with optimized dimensional parameters for each embedding model. Query generation was orchestrated using the LangChain framework, facilitating modularity across retriever, reranker, and generation steps.

Experiments were conducted on systems with at least 16 Gb ram, google colab(16 Gb ram and 16 Gb gpu). Vector embeddings and model checkpoints required approximately 10 GB of storage. A local instance of Prometheus was deployed using Docker, and synthetic time-series metrics were generated to simulate realistic workloads. Queries were validated through the Prometheus HTTP API, allowing automated syntax and output verification..

IV. EVALUATION

A. EXPERIMENTAL SETUP

The system was evaluated using a dataset of natural language queries, spanning a wide range of PromQL functionalities such as metric filtering, aggregation, and time-series operations. Evaluation metrics included syntactic correctness and semantic alignment. Validity was confirmed using a local Prometheus deployment.

B. PERFORMANCE ANALYSIS

Among the configurations tested, RAG v4 achieved the highest query validity (83.33%), followed by RAG v5 (81.72%) and RAG v8a (77.96%). These outperformed standalone LLMs including LLaMA 3.1 8B (78.49%) and LLaMA 3.3 70B (68.28%). The early versions (v1–v3) ranged from 62.90% to 79.57%, while RAG v6 and v7 were less effective due to less optimized retrieval logic.

TABLE 3. PromQL Query Validity Rates Across RAG Versions

RAG Version	Validity Rate (%)
RAG v1	79.57
RAG v2	63.44
RAG v3	62.90
RAG v4	83.33
RAG v5	81.72
RAG v6	51.61
RAG v7	58.60
RAG v7a	56.29
RAG v8	73.12
RAG v8a	77.96

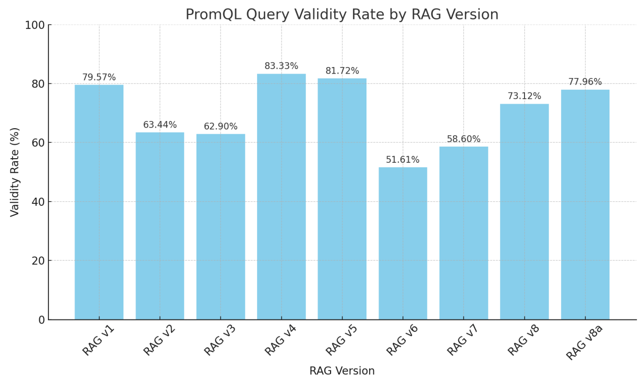

FIGURE 3. PromQL Query Validity Rate by RAG Version

C. KEY FINDINGS

RAG systems significantly outperformed baseline LLMs. Hybrid retrievers and syntax correction steps contributed to this improvement. Interestingly, larger models did not always yield better performance—RAG v4, using Mistral-7B, surpassed LLaMA-70B-based pipelines.

D. ERROR ANALYSIS

Failures were traced to metric mismatches, syntax errors, and improper aggregations. Latency was observed in setups using multi-vector collections and larger embeddings. These findings highlight the trade-off between retrieval richness and computational cost.

E. COMPARATIVE ADVANTAGE

RAG v4 demonstrated a 48.92% improvement over the ChatGPT baseline, a 4.84% improvement over LLaMA 3.1 8B, and a 15.05% gain over LLaMA 3.3 70B. These results validate the architectural design of hybrid retrieval combined with syntax-aware generation.

V. RESULTS AND DISCUSSION

The experimental results affirm the effectiveness of RAG-based systems over standard LLM approaches. Among the evaluated systems, RAG v4 achieved the highest validity score (83.33%), outperforming both retrieval-free LLM baselines and earlier RAG iterations. Hybrid retrieval methods, employed in v4 and v5, consistently improved both precision and semantic relevance, confirming their benefit in contexts with ambiguous or incomplete input.

Interestingly, although larger models typically outperform smaller ones, our baseline comparison revealed that LLaMA 3.1 8B (78.49%) outperformed the much larger LLaMA 3.3 70B (68.28%) when both were used without retrieval augmentation. This result suggests that model size alone does not ensure better performance for structured query tasks, especially without task-specific tuning or retrieval support. In contrast, RAG v7a—despite using retrieval—achieved only 56.29% validity, indicating that naive integration of retrieval is insufficient without proper pipeline optimization.

Error patterns revealed that incorrect metric resolution and time-range misconfiguration were the most frequent causes of failure. The use of Chain-of-Thought (CoT) prompting in RAG v5 and v6 provided mixed results—enhancing reasoning in some cases but also introducing syntactic inconsistencies, which required downstream correction through syntax-aware modules.

From a deployment standpoint, modular agentic RAG pipelines with multi-collection retrievers demonstrated strong potential for generalization across backend languages. These findings support continued development in retrieval quality, prompt adaptation, and execution-aware feedback to further improve domain-specific query generation.

VI. LIMITATIONS

While the proposed RAG architecture demonstrates strong performance in translating natural language into structured queries, several limitations remain. The most prominent constraint is the absence of a large, standardized dataset comprising PromQL documentation paired with natural language queries. This scarcity hinders both the retrieval quality and the fine-tuning of prompt templates, limiting the system's ability to generalize across a broader range of query intents. A robust RAG system depends heavily on diverse and richly annotated domain-specific corpora, which are currently unavailable for PromQL and many similar backend query languages.

Additionally, the lack of open benchmark datasets for other target systems like OpenSearch or Datadog complicates the development of multi-domain generalization mechanisms. Domain adaptation becomes significantly more difficult when representative documents and query logs are limited or proprietary.

Another key limitation lies in the computational demands of evaluating RAG systems with large language models. Testing across multiple LLM sizes and architectures requires substantial compute resources and extended processing time, which may be impractical in real-world deployments without dedicated hardware. This constraint restricts broader experimentation with ensemble or model-switching strategies that could otherwise boost performance.

Addressing these limitations would require community-driven efforts to curate shared datasets, open-source domain logs, and modular evaluation suites for structured query generation across domains.

VII. CONCLUSION

This work presents a comprehensive modular RAG framework for translating natural language instructions into domain-specific query languages, with demonstrated effectiveness on PromQL generation. Through systematic evaluation of eight distinct RAG configurations and two agentic variants, we established that retrieval-augmented approaches significantly outperform standalone large language models for structured query synthesis.

Our key contributions include: (1) a modular architecture that enables seamless adaptation across different vector databases, embedding models, and retrieval strategies; (2) empirical validation showing that hybrid retrieval methods achieve superior performance, with RAG v4 reaching 83.33% syntactic validity—a 48.92% improvement over ChatGPT baselines and 4.84% over LLaMA 3.1 8B; (3) the counterintuitive finding that model scale does not guarantee better performance for structured query tasks, as evidenced by Mistral-7B outperforming LLaMA-70B configurations; and (4) the development of specialized syntax correction pipelines that address common failure modes in DSL generation.

The agentic RAG implementation demonstrates the potential for dynamic tool selection and iterative refinement in complex query scenarios, though our results indicate that naive integration of retrieval mechanisms without proper optimization can degrade performance. The modular design principles we established—combining contextual compression, prompt orchestration, and multi-domain extensibility—provide a foundation for future work in natural language interfaces for observability and monitoring systems.

Beyond PromQL, this framework's generalizability positions it as a viable solution for other domain- specific languages including OpenSearch DSL and Datadog Query Language. The systematic evaluation methodology and performance benchmarks we present establish a reference point for future research in NL-to-DSL translation systems.

Our work addresses a critical gap in AI-driven observability infrastructure by demonstrating that well- architected RAG systems can achieve both syntactic correctness and semantic grounding—essential requirements for production deployment. As organizations increasingly rely on natural language interfaces for complex backend operations, this research provides both theoretical insights and practical implementation guidance for building robust, extensible query generation systems.

Future directions include expanding the framework to additional query languages, developing larger standardized datasets for domain-specific evaluation, and investigating ensemble methods that combine multiple RAG configurations for enhanced reliability. The modular architecture we present serves as a foundation for these extensions while maintaining the core principles of accuracy, extensibility, and minimal tuning requirements that make this approach practical for real-world deployment.

REFERENCES

[1] T. Şakar and H. Emekci, "Maximizing RAG efficiency: A comparative analysis of RAG methods," *Natural Language Processing*, vol. 31, no. 1, pp. 1–25, 2025. [Online]. Available: https://doi.org/10.1017/nlp.2024.53
[2] A. Singh et al., "Agentic Retrieval-Augmented Generation: A survey on Agentic RAG," *arXiv preprint*, arXiv:2501.09136, 2025.
[3] "Mindful-RAG: A Study of Points of Failure in Retrieval-Augmented Generation," 2023. [Online]. Available: https://doi.org/10.1109/FLLM63129.2024.10852457
[4] "Multi-Agent RAG Chatbot Architecture for Decision Support in Net-Zero Emission Energy Systems," *IEEE ICIT*, 2024. https://doi.org/10.1109/ICIT58233.2024.10540920
[5] "Blended-RAG: Improving RAG Retriever-Augmented Generation Accuracy with Semantic Search and Hybrid Query-Based Retrievers," 2024. https://doi.org/10.1109/MIPR62202.2024.00031
[6] "Intelligent Extraction from Diverse Documents," 2024. https://doi.org/10.1109/CSITSS64042.2024.10816908
[7] "Hybrid Text Generation-Based RAG model," 2024. https://doi.org/10.3390/fi15050180
[8] "Efficient and Verifiable Responses Using RAG," *Proc. ACM SIGIR*, pp. 45–58, 2024.
[9] "REAPER: Reasoning-Based Retrieval Planning for Complex RAG Systems," *ACM CIKM*, 2024.
[10] "Enhancing Comprehension with LLM/TTS/RAG," *ACM Multimedia*, 2025.
[11] "Tailoring Your Code Companion," *IEEE TALE*, 2024. https://doi.org/10.1109/TALE62452.2024.10834365
[12] J. Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," *arXiv preprint*, arXiv:2201.11903, 2022.
[13] D. Mondal et al., "KAM-CoT: Knowledge Augmented Multimodal Chain-of-Thoughts Reasoning," *arXiv preprint*, arXiv:2401.12863, 2024.
[14] Y. Tang and Y. Yang, "MultiHop-RAG: Benchmarking Retrieval-Augmented Generation for Multi-Hop Queries," *arXiv preprint*, arXiv:2401.15391, 2024.
[15] Y. Shi et al., "Enhancing Retrieval and Managing Retrieval," *arXiv preprint*, arXiv:2407.10670, 2024.
[16] "S3LLM: Large-Scale Scientific Software Understanding with LLMs," *arXiv preprint*, arXiv:2403.10588, 2024.
[17] "Plan with Code: Comparing Approaches for Robust NL to DSL Generation," *arXiv preprint*, arXiv:2408.08335, 2024.
[18] "Megalodon: Efficient LLM Pretraining and Inference with Unlimited Context Length," *NeurIPS*, 2024.
[19] "Ada-LEval: Evaluating Long-Context LLMs," *ACL*, 2024.
[20] "Optimizing RAG Techniques for Automotive Industry PDF Chatbots," *ACM SIGIR*, 2024.
[21] "PodGPT: Audio-Augmented RAG for Multilingual STEMM Query Generation," *IEEE ICASSP*, 2024.
[22] H. Koo et al., "Optimizing Query Generation for Enhanced Document Retrieval in RAG," *arXiv preprint*, arXiv:2407.12325, 2024.
[23] A. Salemi and H. Zamani, "Evaluating Retrieval Quality in RAG," *arXiv preprint*, arXiv:2404.13781, 2024.
[24] X. Li, "Application of RAG in Complex Query Processing," *Advances in Computer, Signals and Systems*, 2024. https://doi.org/10.23977/acss.2024.080608
[25] S. A. Tripathy et al., "Sound AI Engine for Detection and Classification of Overlapping Sounds in Home Environment," *2023 IEEE Delhi Section Flagship Conference (DELCON)*, pp. 1–6, 2023. doi: https://doi.org/10.1109/DELCON57910.2023.10127311