Queryable Microarchitecture Knowledge Base using Retrieval-Augmented Generation

Vignesh Manjunath
Graz University of Technology
Graz, Austria
vignesh.manjunath@student.tugraz.at

Jesus Pestana
Pro2Future GmbH
Graz, Austria
jesus.pestana@pro2future.at

Tobias Scheipel, Marcel Baunach
Graz University of Technology
Graz, Austria
{tobias.scheipel, baunach}@tugraz.at

Abstract—**Microarchitecture documentation, such as datasheets and user manuals, is indispensable for embedded software development. However, the extensive volume and complexity of these documents render information retrieval a time- and effort-intensive task. To address this challenge, we propose a framework for constructing a queryable knowledge base on microarchitecture documentation, leveraging Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs). As a proof of concept, we implement a knowledge base on AURIX TriCore TC27x documentation and evaluate this knowledge base by querying it with a curated set of questions. The generated responses are evaluated by measuring their semantic similarity to reference answers. In our evaluation, we assess the performance of six LLMs with different model architectures and sizes. The results show that the smaller models (with 8 billion and 3 billion parameters) achieve similarity scores comparable to those of the larger model (with 72 billion parameters). These initial findings demonstrate the robustness of our framework for creating queryable knowledge bases and the potential of smaller LLMs for efficient information retrieval in this context.**

Keywords-Embedded systems, information extraction, retrieval-augmented generation

I. INTRODUCTION

Embedded software development relies on microarchitecture documentation, including datasheets and user manuals, to implement device drivers and various software functionalities. These documents contain information on, e.g., peripheral configuration, memory management, and internal microcontroller behavior. However, finding relevant information is a time-consuming and effort-intensive task since these documents are often hundreds or even thousands of pages long. Moreover, the required information may be dispersed across various sections within a single document or distributed across multiple documents, making it challenging to obtain comprehensive information efficiently. For instance, peripheral configuration information is often fragmented across the datasheet, application notes, and errata documents.

To retrieve information quickly and efficiently, we propose a framework to build a queryable knowledge base on microarchitecture documentation. The main idea is to transform the target microarchitecture documentation into a structured knowledge base, which is subsequently integrated with an information retrieval process involving Retrieval-Augmented Generation (RAG) [1] and a Large Language Model (LLM) [2]. By integrating the knowledge base with the information retrieval process, the framework facilitates querying for Open-Domain Question Answering (ODQA) tasks. In addition, we implement a filtering concept to support document-specific information retrieval.

As a proof of concept and demonstration of our framework, we build a queryable knowledge base on AURIX TriCore TC27x [3] documentation. We evaluate the knowledge base using a set of questions and reference answers. First, we query the knowledge base with the questions and record the generated responses. Next, we compute the semantic similarity between generated responses and their corresponding reference answers. This similarity score reflects the quality of the responses in terms of their relevance and alignment with the reference answers.

The primary focus of this paper is on the development of the proposed framework and a preliminary evaluation to assess the performance of the framework. The framework currently uses a simple naive RAG pipeline with a single retriever to retrieve information from the knowledge base. Further refinement of the RAG pipeline and extensive evaluation of the approach are currently a work-in-progress and are out of scope for this paper.

The rest of the paper is organized as follows: Section II describes the methodology of the proposed framework. Section III presents the current evaluation approach and the preliminary results. Section IV discusses the related work, and Section V concludes the paper with a brief outlook on future work.

II. METHODOLOGY

The naive RAG pipeline in our framework consists of two primary components: a retriever, which is responsible for identifying and extracting the most relevant information from the knowledge base, and a generator (an LLM), which formulates a coherent and contextually appropriate response to a given user query based on the extracted information. In this section, we first explain the process of creating a structured knowledge base using the target microarchitecture documentation, followed by the process of information retrieval and response generation.

Manuscript received May 20, 2025; revised July 31, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.95

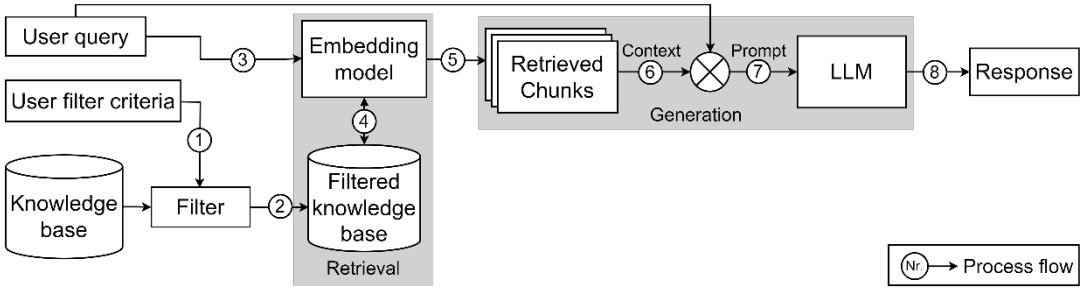

Figure 1: Information retrieval and response generation process.

A. Knowledge Base Creation

Microarchitecture documents are typically PDF files from hardware vendors. In general, these PDFs have complex table structures, images, and non-pertinent information, such as headers and footers. To make the information in the PDFs suitable for processing by an LLM, we first convert these PDFs into Markdown format using the PyPDF2 [4] Python library. Next, we remove non-pertinent information, translate figures into corresponding textual descriptions, and format complex table structures. Subsequently, we add metadata to every document, including details such as titles, versions, and tags.

Depending on the input PDFs, these Markdown files can be lengthy and may exceed the *context length* (i.e., the amount of text, in tokens, the model can process) of an LLM. Hence, we split the Markdown files into equally sized chunks based on word count. Next, we link each chunk with its corresponding document metadata and encode these chunks into dense vector representations (referred to as 'embeddings') using an embedding model (e.g., all-MiniLM-L6-v2 [5]). Lastly, we build and associate indexes for these embeddings using the FAISS library [6] to facilitate faster retrieval of document chunks relevant to a user query. The indexing step completes the creation of the knowledge base.

B. Information Retrieval and Response Generation

The information retrieval and response generation process begins with a user query and optional filter criteria and involves the sequence of steps (denoted by (Nr.)) illustrated in Figure 1.

In steps ① and ②, we filter the knowledge base and extract the embeddings corresponding to the document tag(s) specified by the user filter criteria. The resulting filtered knowledge base is then used to retrieve information relevant to the user query. If no filter criteria are specified, then the information is retrieved from the entire knowledge base.

In step ③, we encode the user query using the embedding model and then use the FAISS library to perform a similarity search on the knowledge base in step ④. The similarity search retrieves indexes of the most similar embeddings from the knowledge base, and these retrieved indexes are used to obtain the corresponding document chunks in step ⑤. Steps ③ through ⑤ represent the information retrieval process.

In step ⑥, the retrieved document chunks are concatenated as *context*. The *context* is then integrated with the user query and the rules for generation as a *prompt* in step ⑦. The rules instruct the LLM to generate a response based only on the provided *context*. The LLM uses the information contained in the *prompt* to generate the final response to the user query in step ⑧. Steps ⑥ through ⑧ correspond to the response generation process.

III. Proof of Concept and Evaluation

A. Evaluation Setup

To demonstrate and evaluate our framework, we implement a queryable knowledge base on a set of documents specific to the AURIX TriCore TC27x architecture. These documents include the core architecture user manuals, Instruction Set Architecture (ISA) description, and errata. We convert these documents into Markdown format and split them into chunks of 100 words each. Next, we encode these chunks into embeddings and then build and associate indexes with these embeddings. The resulting knowledge base is evaluated through semantic similarity analysis.

In our evaluation, we use Copilot and Nemotron [7] LLMs to generate a test dataset using the TC27x documents. The generated test dataset comprises 326 question-answer pairs, and we reviewed 25% of them to check their factual correctness. The answers in the test dataset serve as reference answers for evaluating the quality of the generated responses. Next, we integrate the TC27x knowledge base with the RAG pipeline and use the test dataset to benchmark six LLMs with different model architectures and sizes. Table 1 lists the LLMs under evaluation, and their short names represent the LLM family and the number of model parameters.

We conduct our evaluation by querying the LLM with the test questions and recording the generated responses. This evaluation is systematically repeated for all the LLMs under evaluation, and their responses are recorded. The evaluation is performed on a system equipped with three NVIDIA A100 80GB GPUs [10].

In addition to the response quality, we also measure the mean inference time for each LLM to assess its computational efficiency. As shown in Table 1, larger models exhibit higher inference times (e.g., Nemotron-70B at 28.25 s), while smaller models respond significantly faster (e.g., Llama3.2-1B at 1.70 s), illustrating the trade-off between model size and computational cost. In contrast, R1_DQwen_7B, although smaller than Nemotron_70B, exhibits a comparable inference time (27.34 s). This extended processing time is likely attributed to its chain-of-thought reasoning approach, which requires longer reasoning chains and tracking multiple logical branches, thereby increasing the computational effort required.

TABLE 1: LLMS UNDER EVALUATION.

LLM short name	Number of model parameters	Model size (GiB)	Mean inference time (seconds)
Llama3.2_1B [9]	1.23 billion	2.31	1.70
Llama3.2_3B [9]	3.21 billion	5.98	7.42
Qwen2.5_3B [8]	3.09 billion	5.76	5.66
R1_DQwen_7B [13]	7.62 billion	14.19	27.34
Llama3.1_8B [9]	8.03 billion	14.96	6.24
Nemotron_70B [7]	70.60 billion	131.5	28.25

B. Semantic Similarity Score Computation

In Natural Language Processing (NLP), semantic similarity scores are used to measure how closely two texts are aligned in meaning and context. In our work, we use an ensemble approach to compute the similarity score between the responses generated by different LLMs and their respective reference answers. The ensemble approach leverages two popular NLP metrics: BERTScore-F1 [11] and SBERT similarity score [12].

BERTScore-F1 measures how similar individual tokens are between two sentences by considering their meaning and context, while the SBERT similarity score compares the overall meaning of two sentences by transforming them into vector representations and measuring their closeness. Both scores range from -1 to +1, with values closer to +1 indicating a higher degree of similarity.

For each response generated by the LLMs under evaluation, we calculate the corresponding similarity scores, compute their means, and present the results in Table 2. The results indicate that the mean similarity scores remain consistent across both evaluation metrics. The model R1_DQwen_7B achieves the lowest mean similarity scores of all the LLMs under evaluation. This lower performance can be primarily due to two factors: (1) the inclusion of chain-of-thought reasoning in its responses, which introduces additional content, and (2) deviations in final answers, thereby reducing alignment with the expected outputs.

In contrast, most of the other smaller models achieve similarity scores closely aligned with those of the larger Nemotron_70B model. In particular, the smaller Llama3.1_8B model slightly outperforms the larger Nemotron_70B model, achieving the highest similarity score of 0.67 (highlighted using bold text in Table 2). This finding demonstrates the potential of smaller LLMs for effective information retrieval.

TABLE 2: MEAN SIMILARITY SCORE.

LLM short name	Mean BERT score-F1	Mean SBERT similarity score
Llama3.2_1B	0.57	0.61
Llama3.2_3B	0.65	0.65
Qwen2.5_3B	0.64	0.66
R1_DQwen_7B	0.50	0.49
Llama3.1_8B	**0.67**	**0.67**
Nemotron_70B	0.63	0.65

The similarity scores across models remain moderately close to +1, indicating a relatively high degree of similarity between the generated responses and their corresponding reference answers. This consistency highlights the robustness of our knowledge base framework and the computational efficiency of some smaller models, which are capable of generating contextually relevant outputs while significantly reducing GPU memory consumption and computational overhead compared to the larger Nemotron_70B model.

IV. RELATED WORK

In recent years, several approaches have leveraged various RAG architectures to address a broad range of tasks. Surveys such as [14-16] provide comprehensive overviews of RAG-based methods across multiple domains and applications, including domain-specific information retrieval, software safety analysis, and code generation. This section focuses specifically on existing approaches that employ RAG for domain-specific information retrieval.

Similar to our work, AeroQuery [17] and IDAS [18] use a naive RAG pipeline with vector similarity search to extract information from aerospace standards (e.g., DO-178C) and vehicle user manuals, respectively. In contrast, Kieu et al. [19] employ a hybrid retrieval approach that combines keyword-based and vector-based search results to enhance the explainability of AUTOSAR specifications. However, these approaches are evaluated on relatively small-scale datasets, typically involving only around 20 queries, which limits the generalizability and robustness of their findings.

Simoni et al. [20] introduce a multi-retriever RAG system that retrieves both textual information and code to answer cybersecurity-related queries. Similarly, Balu et al. [21] use multiple retrievers (one per document) to extract information from automotive standards. While both approaches reduce redundancy and summarize outputs from individual retrievers, the aggregated information can exceed the LLM's context length, potentially hindering response quality.

Some approaches [22–25] involve Graph-RAG, which retrieves relevant information from graph structures rather than isolated textual chunks. CyKG-RAG [22] applies this to cybersecurity by leveraging domain-specific knowledge graphs for multi-hop Q&A tasks. HSG-RAG [23] constructs hierarchical semantic knowledge graphs to improve retrieval from embedded systems documentation (such as API reference manuals). Liu et al. [24] use Graph-RAG to retrieve information from automotive software specifications, and Ojima et al. [25] extract information from event graphs representing automotive failure incidents. Although these methods demonstrate improved contextual retrieval, they often encounter challenges related to traceability and the limited context length of LLMs, particularly when aggregating information from numerous graph nodes or documents.

In contrast, our approach adopts a naive RAG pipeline augmented with a pre-retrieval filtering mechanism, which helps mitigate the context length limitations commonly encountered in graph-based or multi-retriever RAG systems. This filtering strategy enhances retrieval quality by selecting document chunks based on the user filter criteria, thereby improving the relevance of the retrieved information with respect to the user

query. Furthermore, our preliminary evaluation results indicate that smaller LLMs can achieve performance levels comparable to their larger counterparts, thereby highlighting the feasibility of resource-efficient deployments without significant loss in retrieval quality.

V. Conclusion and Future Work

In this work, we presented a framework for building a queryable knowledge base on microarchitecture documentation using RAG with an LLM. Our proof-of-concept based on TC27x documentation demonstrates the feasibility of this approach for quick and efficient information retrieval in embedded software development. As a preliminary evaluation, we used semantic similarity metrics to assess the performance of six LLMs with different model architectures and sizes. The results show that smaller models, including those with 8 billion and 3 billion parameters, can achieve similarity scores comparable to those of a significantly larger model with 72 billion parameters. These findings highlight the robustness of our framework and the potential of smaller LLMs as resource-efficient alternatives for domain-specific information retrieval tasks.

While the preliminary evaluation demonstrates the feasibility of our approach, further work is required to enhance both the evaluation methodology and the underlying system. As future work, we plan to evaluate the factual correctness of the generated responses. This will involve developing or integrating more rigorous evaluation metrics and possibly including human-in-the-loop assessments.

In addition, we intend to refine the current naive RAG pipeline to improve retrieval quality. This includes optimizing document chunking strategies, enhancing query formulation, and exploring more advanced search and ranking strategies. These improvements are expected to increase the precision and relevance of retrieved content, thereby improving the robustness of our knowledge base framework.

Another important step in our future work is the implementation of an explicit traceability mechanism. Although naive RAG inherently allows tracking of generated responses back to the retrieved chunks, we intend to formalize this process by extracting and verifying the relevance of each chunk with respect to the final answer. This will enable more explainable and reliable responses, thereby increasing user trust in the knowledge base framework.

Finally, we plan to fully automate the conversion of source PDFs into structured markdown files. This includes extracting key elements like text, headings, and tables to streamline content preparation. Automating this step will significantly reduce manual preprocessing effort and ensure consistency and scalability in building and updating knowledge bases.

Acknowledgment

This work has been supported by the BMIMI, BMWET, and FFG, Contract No. 911655: "Pro²Future II", Graz University of Technology (TU Graz), and Elektrobit Automotive GmbH.

References

[1] P. Lewis et al., 'Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks', in *Advances in Neural Information Processing Systems*, 2020, vol. 33.

[2] V. Ashish et al., 'Attention is all you need', Advances in neural information processing systems, vol. 30, p. I, 2017.

[3] 'AURIX TC27x D-Step 32-bit Single-Chip Microcontroller User Manual v2.2', Infineon Technologies AG.

[4] 'PyPDF2'. [Online]. Available: https://pypdf2.readthedocs.io/en/3.x/

[5] 'Sentence-Transformer Model: all-MiniLM-L6-v2'. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[6] M. Douze et al., 'The FAISS library', arXiv preprint arXiv:2401. 08281, 2024.

[7] W. Zhilin et al., 'HelpSteer2-Preference: Complementing Ratings with Preferences', arXiv [cs.LG]. 2024.

[8] Q. Team, 'Qwen2.5: A Party of Foundation Models'. Sep-2024. Available: https://qwenlm.github.io/blog/qwen2.5/

[9] A. Grattafiori et al., 'The llama 3 herd of models', arXiv preprint arXiv:2407. 21783, 2024.

[10] Nvidia, 'LLM Benchmarking: Fundamental Concepts'. [Online]. Available: https://developer.nvidia.com/blog/llm-benchmarking-fundamental-concepts/.

[11] T. Zhang and Others, 'Bertscore: Evaluating text generation with bert', arXiv preprint arXiv:1904. 09675, 2019.

[12] N. Reimers and I. Gurevych, 'Sentence-bert: Sentence embeddings using siamese bert-networks', arXiv preprint arXiv:1908. 10084, 2019.

[13] DeepSeek-AI, 'DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning', arXiv [cs.CL]. 2025.

[14] M. Arslan et al., 'A Survey on RAG with LLMs', Procedia Computer Science, vol. 246, pp. 3781–3790, 2024.

[15] Y. Gao et al., 'Retrieval-augmented generation for Large Language Models: A survey', arXiv preprint arXiv:2312. 10997, vol. 2, p. 1, 2023.

[16] R. Chen et al., 'Retrieval-Augmented Generation with Knowledge Graphs: A Survey', in Computer Science Undergradaute Conference 2025@ XJTU.

[17] S. Yadav, 'AeroQuery RAG and LLM for Aerospace Query in Designs, Development, Standards, Certifications', in 2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2024, pp. 1–6.

[18] B. Hernandez-Salinas et al., 'IDAS: Intelligent Driving Assistance System using RAG', IEEE Open Journal of Vehicular Technology, 2024.

[19] K. Kieu et al., 'Empowering Automotive Software Development with LLM-RAG Integration', 2024.

[20] M. Simoni et al., 'Morse: Bridging the gap in cybersecurity expertise with retrieval augmented generation', in Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing, 2025, pp. 1213–1222.

[21] B. V. Balu et al., 'Towards Automated Safety Requirements Derivation Using Agent-based RAG', arXiv preprint arXiv:2504. 11243, 2025.

[22] K. Kurniawan et al., 'CyKG-RAG: Towards knowledge-graph enhanced retrieval augmented generation for cybersecurity', 2024.

[23] Z. Lu et al., 'HSG-RAG: Hierarchical Knowledge Base Construction for Embedded System Development', ACM Transactions on Design Automation of Electronic Systems.

[24] F. Liu et al., 'Enhancing Automotive PDF Chatbots: A Graph RAG Approach with Custom Function Calling for Locally Deployed Ollama Models', in Proceedings of the 2024 International Conference on Artificial Intelligence, Digital Media Technology and Interaction Design, 2024, pp. 6–13.

[25] Y. Ojima et al., 'Knowledge Management for Automobile Failure Analysis Using Graph RAG', in 2024 IEEE International Conference on Big Data (BigData), 2024, pp. 6624–663