Security architectural pattern recommendation in a RAG-based assistant: Results of an LLM-as-a-judge evaluation

Rebeca M. P. Sombra
Instituto de Ciências Matemáticas e de Computação/Universidade de São Paulo
São Carlos - SP, Brazil
rebeca.maia@usp.br

Domenico Amalfitano
Department of Electrical Engineering and Information Technology/University of Naples Federico II
Naples, Italy
domenico.amalfitano@unina.it

Lina Garcés
Instituto de Ciências Matemáticas e de Computação/Universidade de São Paulo
São Carlos - SP, Brazil
linagarces@usp.br

Abstract

In software architecture, Large Language Models (LLMs) are increasingly adopted as decision-support tools to streamline complex, time-consuming tasks. They are particularly effective at recommending information that is relevant to specific contexts. When considering the quality attributes that influence the design of system components, integrating security patterns early in the development process is essential for mitigating and preventing vulnerabilities. To aid in this endeavor, we previously introduced *ArchSec.AI*, a retrieval-augmented generation (RAG)-based assistant that utilizes a curated, domain-specific knowledge corpus of security architectural patterns. To assess both the individual components and the overall effectiveness of *ArchSec.AI*, we employed an LLM-as-a-judge evaluation tool. This tool measures key metrics such as faithfulness, answer relevance, context precision, and context recall. In addition to previous qualitative assessments, the quantitative results presented in this paper indicate that tools such as *ArchSec.AI*require significant engineering effort to provide more reliable recommendations based on the context specified in user prompts.

Keywords

Retrieval Augmented Generation, Software Architecture, Security Design Patterns, LLM-as-a-judge

1 Introduction

Large Language Models (LLMs),usually trained on large corpora of web-sourced content, have been particularly useful in software engineering tasks due to their ability to process natural language queries without requiring familiarity with high-end information, making them valuable in contexts where software engineers need quick access to architectural knowledge [3]. However, they are limited to the information available at the time of training, which makes it challenging to provide up-to-date, context-specific information. Recent advances have led to the exploration of Retrieval-Augmented Generation (RAG) based tools to assist with various tasks in the architecture design process, thereby mitigating this limitation [4, 6]. RAG is an advanced AI technique that enhances the model's retrieval process by fetching relevant external information to improve response accuracy and contextual depth [16]. Instead of relying solely on static training data, RAG improves factual grounding, enables knowledge updates without full retraining, and supports domain adaptation and knowledge merging [13], including in domains such as software architecture and security. Within the software development life cycle, architectural design is one of the most expensive and time-consuming processes, requiring architects to make decisions that satisfy the system's requirements and underlying quality attributes, such as security [2]. In systems where security is a major quality attribute, an architect implementing authentication, confidentiality, and other mechanisms can use RAG-based solutions to bridge the gap between skill and the necessary expertise by recommending security architectural patterns. Architectural patterns provide a mechanism to capture domain experience and knowledge, allowing it to be reapplied when a new architectural problem arises, serving as packages of design decisions that recur in practice [2]. *ArchSec.AI*, a RAG-powered assistant, was proposed [19] to help software engineers and security professionals in the decision-making process of selecting the appropriate security architectural pattern for a specific context. The corpus that powers *ArchSec.AI* was built from a previous systematic literature mapping that, combined with the snowballing technique, established the state of the art of security architectural patterns [17]. The *ArchSec.AI*'s architecture follows a typical RAG system's architecture [20], comprising an ingestion pipeline, used to populate the vector DB; a retrieval pipeline, a module that queries the vector DB and retrieves relevant entries to the user's input; and the generation pipeline: The layer that uses the retrieved data to augment the prompt and an LLM to generate answers. An initial assessment of *ArchSec.AI* was conducted through an exploratory study that used qualitative data from human-as-a-judge evaluation, a method often used to assess the quality and accuracy of model-generated results, relying primarily on the expertise of human evaluators. This approach has contributed to a more nuanced and comprehensive understanding of the LLM under scrutiny. However, it is important to note that this method is inherently subjective, which complicates reproducibility [13]. The LLM-as-a-judge approach was introduced by Zheng *et al. (2023)* [21] and involves prompting LLMs to evaluate each sample's quality, then using a second LLM to evaluate LLM outputs using automated metrics. This approach initially requires creating test cases that cover a diverse set of prompts. Then, the LLM is applied to generate responses for each test. The outputs can be assessed on multiple criteria, including relevance, comprehensiveness, and groundedness [13]. Some works proposing RAG-based assistants in the software architecture domain employed various strategies to evaluate their performance [4–6]. *ArchMind* [6], an RAG-based tool, was proposed to help architects with tasks such as pattern recommendation and recording design decisions using a structured Architecture Design Record (ADR) template. The adopted evaluation relied on a small experiment comparing

LLM- and human-generated ADRs. In [4], a RAG design assistant was proposed to support computer scientists and engineers with computer architecture issues, allowing users to better understand the structure with less coding or hardware design proficiency. In its assessment, two experiments were conducted: applying question prompts in the model without RAG context, and applying them with the RAG context, and comparing their results. The work of [5] introduces DRAFT, an AI-based tool that combines domain-specific fine-tuning, RAG, and few-shot learning to enhance the model's capability to generate architectural design decisions and to record key information about the context of the system being developed. DRAFT was evaluated against existing approaches on a dataset of 4,911 ADRs, using various LLMs, and analyzed with automated metrics and human evaluations. These three studies primarily focused on qualitative assessments. One of them incorporated quantitative data from an automated evaluation, using traditional NLP metrics such as ROUGE, BLEU, and BERTScore. These metrics prioritize the order and accuracy of words and phrases but do not adequately assess the overall quality of the RAG system, which requires a more holistic approach. We propose a preliminary quantitative evaluation of *ArchSec.AI* using a state-of-the-art tool, the RAGAs framework [8], which evaluates the performance of RAG systems via *llm-as-a-judge* evaluation. This study provides an objective assessment of *ArchSec.AI*, highlights areas for improvement, and tracks the impact of changes over time. By leveraging the results of this evaluation, we gain insights that help monitor the quality of *ArchSec.AI* over time. The remainder of this paper is organized as follows. Section 2 presents the experimental design adopted to conduct the exploratory study. Section 3 presents and discusses the overall results obtained from the exploratory study. Section 4 presents the strategies used to mitigate the impact of the threats. Finally, Section 5 presents the paper's final remarks.

2 Methods

The main goal of this exploratory study is to measure how accurately *ArchSec.AI* retrieves important documents to ground its responses in recommendations on security architecture patterns, as well as to assess the quality of the generated text for specific answers. From the results obtained, we can directly identify which components need improvement. To guide our evaluation, we formulate the following **research question**: *RQ: How accurate are the responses of ArchSec.AI when compared to a ground-truth answer?* The following null and alternative hypotheses were formulated: H_0**:** There are no differences between the *ArchSec.AI* responses and the ground-truth answers. H_1**:** There are differences between the *ArchSec.AI* responses and the ground-truth answers. The **experimental object** consists of one RAG-based tool: *ArchSec.AI*. The first author of this work served as the principal subject in this study and executed the intervention. The intervention involves executing security pattern recommendation requests in *ArchSec.AI* , using the following zero-shot, Chain-of-Thought prompt template:

'I am a <person of interest>, and I need to design a <technical domain> software system for <application domain> that implements <security attribute> mechanisms. The system has the following requirements: 1., 2., ... Recommend architectural security patterns that solve <security problem>.'

Across all prompt executions, the controlled variables were the judge model, prompt template, and hyperparameters (i.e., *ArchSec.AI*'s temperature = 0.1; Judge temperature = 0; Maximum tokens = 2048; top_p = 0.2; and top_k chunks = 6). As factors, we used the variables in the prompt: person of interest, technical domain, application domain, security attribute, system's requirements, and security problem. Such variables were selected because they are common in security architecture pattern documentation. The treatments (or factors' values) were obtained from descriptions of five distinct security architectural patterns: Trusted Base [14], Patching [10], Mediate between application and user [15], Error Detection/Correction [11], and Voter Authentication [18]. Such patterns were randomly selected from the 98 patterns in the *ArchSec.AI*'s corpus. Random pattern selection was done by using a Python script available in a Zenodo repository [17]. Factors' information was extracted from those patterns and imputed as variable values in prompt templates. The reference answers (*ground truth*) were defined based on a summary of the pattern's full description. A summary based on the full pattern description was considered adequate if it contained key information to identify the problem and the solution the pattern was intended to address. As dependent variables, the LLM judge model analyzed the outcomes and measured them based on four quality metrics evaluated by the RAGAs framework [8]: (i) Faithfulness: judges how well the LLM' answer is grounded based on the given context, maintaining consistency and avoiding contradictions; (ii) Answer Relevance: decides if generated answer is pertinent to the provided question, effectively addressing the core inquiry; (iii) Context Precision: assesses the degree to which relevant chunks in the retrieved context are placed at the top of the ranking; and (iv) Context Recall: assesses how many of the relevant pieces of information were successfully retrieved. These metrics ensure that RAG-generated responses are not only relevant but also grounded in factual evidence. As a control group, specifications of five randomly selected security patterns were used. Additionally, to execute this evaluation, we used the following stack: RAG Evaluation Framework: RAGAs 0.4.3; LLM Judge Model: OpenAI GPT 4.0; LLM Judge Embedding Model: OpenAI 'text-embedding-ada-002'.

3 Results and Discussion

First, the Python script [17] was used to evaluate the performance of an RAG model on a dataset of questions and their corresponding answers. The goal is to calculate the previously selected metrics. The code uses libraries such as *ragas* to evaluate the model's performance, *openai* to interact with the OpenAI API, and *pandas* for data manipulation. Here are the main steps that the script takes: (1) Collects chunks of text (retrieved contexts) using a retriever model. (2) Defines an evaluation set of five questions filled out according to the prompt template, adapting it with information about each of the 5 security patterns and its respective ground-truth answers. (3) Evaluates the performance of the RAG model on the evaluation set through the calculation of the selected metrics. (4) Calculates the average values for each metric and stores them in a DataFrame. (5) Saves the evaluation results to a CSV file named "score.csv". The obtained results are shown in Table 1. We established a threshold of >0.85 as a baseline to determine the quality of the results. The original RAGAS paper does not establish absolute thresholds like 0.85 [8], but leaves the choice of thresholds to the implementation

phase. However, some practical guides and blogs utilizing RAGAS propose production thresholds based on empirical experience—such as 0.85 [1, 7].

Table 1: Metrics results per question

question	faithfulness	answ_rel	cont_prec	cont_rec
Q1	0.27	0.91	0.0	0.0
Q2	0.81	0.78	0.0	0.0
Q3	0.77	0.88	0.00	0.0
Q4	1.0	0.0	0.0	0.0
Q5	0.62	0.89	0.0	0.83

For **question 1**, we obtained a Faithfulness score of 0.27. This indicates that the patterns retrieved do not adequately match the context provided in the question. However, we achieved an Answer Relevancy score of 0.91, suggesting that, although the patterns proposed by the assistant differ from those defined in the ground truth answer, they still offer good alternatives for addressing the requirements and problems outlined in the question. Additionally, we recorded Context Precision and Context Recall scores of zero, indicating that the assistant is failing to retrieve the most important documents and pieces of information from the corpus necessary to respond to the question effectively. For **question 2**, we achieved a Faithfulness score of 0.81, indicating that the retrieved patterns are somewhat useful and consistent with the context. However, our Answer Relevancy score was 0.78, suggesting that the proposed patterns do not accurately match the original query. Additionally, we recorded Context Precision and Context Recall scores of zero, which means that the assistant is not retrieving the most important documents and information from the corpus to effectively address the question. For **question 3**, we obtained a Faithfulness of 0.77. This indicates that the generated response does not accurately reflect the provided context. We also achieved an Answer Relevancy score of 0.88, suggesting that although *ArchSec.AI* did not propose the expected patterns outlined in the ground truth answer, the alternative patterns still meet the requirements and address the problem presented in the question. However, we recorded Context Precision and Context Recall scores of zero, which means that the assistant is failing to retrieve the most important documents and pieces of information necessary to effectively answer the question. For **question 4**, we received a Faithfulness score of 1, along with scores of zero for Answer Relevancy, Context Precision, and Context Recall. The inability of *ArchSec.AI* to locate the information in the provided documents suggests that the question is poorly formulated and contradictory. For **question 5**, we obtained a Faithfulness score of 0.62, indicating that the generated response is inaccurate based on the provided context. We also received a Relevancy score of 0.89, suggesting that while our response does not follow the expected patterns outlined in the ground truth answer, the alternative patterns proposed still effectively meet the requirements and address the problem stated in the question. However, we recorded a Context Precision of zero, which means that the retrieved documents are not suitable for the question posed. Finally, we achieved a Context Recall score of 0.83, indicating that the retrieved documents cover all relevant aspects of the query, even though they do not align with the reference response. The overall mean of each metric is: **Faithfulness:** 0.7007; **Answer Relevance:** 0.6960; **Context Precision:** 0.000; **Context Recall:** 0.1667. These results indicate that the faithfulness and answer relevancy, comprising the generation pipeline, need significant improvement. The assistant is retrieving information that is not included in the context, which causes the answers to not directly address the questions. Possible solutions to this issue include using more restrictive prompts, lowering the temperature to enhance response precision, implementing better prompt engineering techniques in the prompt template, and incorporating few-shot examples. The retrieval pipeline, which comprises Context Precision and Context Recall metrics, shows that this is the main weakness of *ArchSec.AI*, with unsubstantial results. The best way to address this is to use more robust and sophisticated chunking methods, focusing on re-ranking. An initial search may yield multiple documents, not all of which are truly relevant to the user's question. Re-ranking is the technique used to reorder these results: it refines the initial list of retrieved documents by reorganizing and filtering sources to highlight those most relevant to the query. Also, implement hybrid search, which integrates keyword, semantic, and vector searches to cater to diverse queries, mitigating the weaknesses of individual retrieval methods while leveraging their strengths in a single pipeline. The results suggest that the alternative hypothesis H_1 is true, meaning that there are differences between the *ArchSec.AI* responses and the ground-truth answers, requiring new RAG engineering strategies, different from the traditional ones, to improve results for this type of system.

4 Threats to Validity

Limitations along with mitigation strategies are presented as follows. **Construct Validity:** The following factors were identified: *Scope of LLM-as-a-judge evaluation methods:* This study is limited to the selected framework to develop this evaluation, RAGAs. Among the various evaluation frameworks for LLM systems, RAGAs stands out due to its integrated dataset management and result-tracking mechanisms, as well as its integration with popular frameworks like LangChain - the primary framework used to develop *ArchSec.AI*. However, RAGAs may not encapsulate all aspects of RAG-based solutions. Other popular option among LLM-as-a-judge frameworks include DeepEval [12]. In future iterations of *ArchSec.AI*, evaluations using other frameworks will be conducted for comparative purposes. *Scope of RAG system evaluation methods: LLM-as-a-judge vs. Human evaluation:* By automating the evaluation process using LLM judges, researchers can rapidly assess the quality of outputs generated across large datasets. In the case of the *ArchSec.AI* evaluation, this method proved particularly useful given that the training corpus contains over 300 security architecture patterns; this highlights the method's potential to cover the entire knowledge corpus and, consequently, address the scalability issues inherent in human annotation. However, a significant limitation of the *LLM-as-a-judge* evaluation method is that LLMs typically rely exclusively on predefined evaluation criteria. In contrast, human evaluators can compare multiple responses while capturing greater nuance and variation in the answers. Human effort remains the golden standard for evaluation and curating reference answers in datasets; however, it is both slow and costly, thereby undermining the very purpose of

automated evaluation. To address the limitations of both evaluation methods, *ArchSec.AI* will employ a mixed approach, enriching the results obtained from LLM judges with findings from a future expert-led evaluation. **Internal Validity:** The following factors were identified: *Prompt engineering constraints :* Only zero-shot CoT promptings were considered while formulating the questions, omitting techniques such as few-shot and tree-of-thought prompting. In future evaluations, these techniques will be employed. *Evaluation setup:* There are also limitations regarding other aspects of the evaluation setup, like the chosen judge, the embedding model chosen for the judge, and hyperparameters. In future iterations of *ArchSec.AI*, different adjustments will be tested in order to obtain the optimal answer. From another standpoint, leveraging a jury of smaller LLMs can also reduce costs while increasing accuracy and mitigating intra-model favoritism. To improve evaluation reliability, future evaluations will adopt this strategy by selecting a single LLM judge. Future evaluations will adopt this strategy, using multiple LLMs as a jury to reduce bias and improve consistency. At last, regarding the **External Validity** of this study, the following factor was identified: *Scale of Artifacts and Instruments:* The scope of our conclusions is inherently bounded by the scale of the experiment, which evaluates five specific security patterns across one instrument, *ArchSec.AI*. To mitigate selection bias, the five evaluated security patterns were randomly selected from the comprehensive set of 97 security pattern classifications identified in our prior systematic mapping.

5 Conclusion and Future Work

The RAGAs evaluation revealed that improving *ArchSec.AI* depends not only on selecting stronger LLMs but also on engineering the entire RAG pipeline. Future work includes to expand the number of test cases in order to draw statistical relevant conclusions. In future versions of *ArchSec.AI* we will adopt Advanced RAG techniques [9], such as semantic and hybrid chunking, hybrid retrieval, re-ranking, and specialized vector databases (e.g., Qdrant and Weaviate). We also plan to expand the evaluation corpus, incorporate additional RAGAs metrics, compare stronger and different types of LLM judges in order to mitigate bias, design more challenging benchmark questions, and complement LLM-as-a-Judge with expert evaluation. Our experience also provides practical lessons for engineering domain-specific RAG systems. Open-source LLMs can achieve competitive results when supported by an effective retrieval pipeline, but performance depends on the joint calibration of chunking, embeddings, retrieval strategy, context size, prompting, and evaluation metrics. Therefore, RAG systems should be treated as continuously evolving software systems that require iterative tuning, monitoring, and validation rather than one-time deployment. We believe these lessons can guide the development of more reliable and maintainable RAG-based assistants for software engineering.

Artifact Availability

Artifacts are available as Zenodo repositories in [17].

Declaration on AI Use

AI tools were used only for language refinement and grammar correction. They were not involved in the study design, data collection, analysis, or interpretation. All scientific content and conclusions are solely the responsibility of the authors.

Acknowledgements

The authors gratefully acknowledge the PRPI-USP (Grant: 22.1.09345. 01.2) and Brazilian agencies for financial support: CAPES and CNPq (Grant: 406941/2025-4, 420025/2023-5).

References

[1] 2026. RAG Evaluation with RAGAS — Metrics, Pipelines & Benchmarks (2026). https://myengineeringpath.dev/genai-engineer/rag-evaluation/
[2] Len Bass. 2012. *Software architecture in practice*. Pearson Education India.
[3] Humberto Cervantes, Rick Kazman, and Yuanfang Cai. 2025. An LLM-assisted approach to designing software architectures using ADD. *arXiv preprint arXiv:2506.22688* (2025).
[4] Wenderson Júnio de Souza, Humberto Torres Marques Neto, and Henrique Cota de Freitas. 2025. Retrieval-Augmented Large Language Models for Computer Architecture Learning and Design Assistance. *International Journal of Computer Architecture Education* 14, 1 (2025), 12–18.
[5] Rudra Dhar, Adyansh Kakran, Amey Karan, Karthik Vaidhyanathan, and Vasudeva Varma. 2025. Draft-ing architectural design decisions using llms. *arXiv preprint arXiv:2504.08207* (2025).
[6] J Andrés Díaz-Pace, Antonela Tommasel, and Rafael Capilla. 2024. Helping novice architects to make quality design decisions using an llm-based assistant. In *European Conference on Software Architecture*. Springer, 324–332.
[7] Pramod Dutta. 2026. Ragas Faithfulness & Answer Relevancy: 2026 Guide | QASkills.sh. https://qaskills.sh/blog/ragas-faithfulness-answer-relevancy-guide
[8] Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. Ragas: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th conference of the european chapter of the association for computational linguistics: system demonstrations*. 150–158.
[9] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).
[10] Lazaros Gymnopoulos, Maria Karyda, Theodoros Balopoulos, Stelios Dritsas, Spyros Kokolakis, Costas Lambrinoudakis, and Stefanos Gritzalis. 2006. Developing a security patterns repository for secure applications design. In *Proceedings of the 5th European Conference on Information Warfare and Security*. 51–60.
[11] SALAR AZIMI HAREDASHT and HASSAN RASHIDI. [n. d.]. Evaluation of the Existing Security Patterns in Software Security. ([n. d.]).
[12] Jeffrey Ip and Kritin Vongthongsri. 2026. *deepeval*. https://github.com/confident-ai/deepeval
[13] Uday Kamath, Kevin Keenan, Garrett Somers, and Sarah Sorenson. 2024. Large language models: A deep dive. *Bridging Theory and Practice, Cham: Springer Nature* (2024).
[14] Eunsuk Kang and Daniel Jackson. 2010. Patterns for building dependable systems with trusted bases. In *Proceedings of the 17th Conference on Pattern Languages of Programs*. 1–14.
[15] Alan H Karp and Kevin Smathers. 2003. Three design patterns for secure distributed systems. *Information security bulletin* (2003), 49–54.
[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.
[17] Rebeca M. P. Sombra Lina Garcés and Domenico Amalfitano. 2026. *Security architectural pattern recommendation in a RAG-based assistant: Results of an LLM-as-a-judge evaluation*. doi:10.5281/zenodo.21340016
[18] P Salini and S Kanmani. 2012. A knowledge-oriented approach to security requirements engineering for e-voting system. *International Journal of Computer Applications* 49, 11 (2012).
[19] Rebeca M. P. Sombra and Lina Garcés. 2026. ArchSec.AI: Towards a LLM-based assistant to support analysis of security architectural strategiess. In *Anais do XXXX Simpósio Brasileiro de Engenharia de Software* (São Paulo/SP).
[20] Dean Wampler, Dave Nielson, and Alireza Seddighi. 2025. Engineering the RAG Stack: A Comprehensive Review of the Architecture and Trust Frameworks for Retrieval-Augmented Generation Systems. *arXiv preprint arXiv:2601.05264* (2025).
[21] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.