ArchSec.AI: Towards a LLM-based assistant to support analysis of security architectural strategies

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

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

Abstract

Large Language Models (LLMs) have been widely used informally by software developers and architects for tasks that require specialized knowledge. When designing software architecture, it is necessary to consider quality aspects, such as security. The most common way to incorporate security is to use patterns, which are reusable solutions for recurring architectural problems. However, given the variety of existing patterns and their different purposes, architects with less expertise may have difficulty selecting appropriate patterns. This highlights the need to provide tools to support architects' decision-making process in identifying and applying security patterns in an informed manner. To facilitate information retrieval, a corpus was built that unifies existing information in the literature on architectural patterns for security. The experiments were conducted using Gemini, DeepSeek, and NotebookLM. Preliminary results indicate that general-purpose LLMs frequently generate hallucinated information when addressing domain-specific and constrained problems. However, their responses are elevated when enriched with a highly specialized corpus. This evidence supports the development of *ArchSec.AI*, an AI assistant grounded in RAG (Retrieval-Augmented Generation) and a corpus that enables accurate and traceable decision support for the design of secure architectures.

Keywords

Large Language Models, Software Architecture, Patterns, Security

1 Introduction

Modern software systems exhibit unprecedented levels of architectural complexity, primarily driven by pervasive network connectivity. This interconnectedness expands the attack surface of applications that manage highly sensitive and valuable data [15]. The real-world consequences of these attacks are severe; global news outlets report daily security breaches targeting applications, resulting in millions of dollars in direct and indirect financial losses [6, 7]. Incidents of this nature underscore a challenge: secure software cannot be treated as an afterthought. To mitigate such risks, software architects must comprehensively understand and systematically deploy proven architectural security strategies [5]. These strategies synthesize approximately three decades of knowledge, codified into security architectural patterns, tactics and principles, in order to guide the robust design, construction, and evaluation of software systems. Concurrently, the rapid evolution of Large Language Models (LLMs) has introduced powerful tools capable of generating sophisticated natural language responses [1]. In certain design scenarios, it can reliably assist architects. Recent studies have already proposed LLM-based solutions to identify and apply object-oriented design patterns [8]. Going further, works published in the literature show a prominent interest in investigating Retrieval-Augmented Generation (RAG) powered tools to recommend architectural patterns and generate relevant artifacts for the architectural design process, such as Architectural Decision Records (ADRs) [10–12]. This highlights the immense potential of artificial intelligence in paving new pathways for industry development.

While these works achieved remarkable efficiency across diverse software architecture tasks, their utility within highly specialized domains is constrained by the boundaries of the training data that composes their underlying corpus. In the context of LLMs, a corpus is the dataset used in the model's training, comprising a curated collection of documents, web pages, books, code, or other text sources [9]. The knowledge corpus underlying these RAG-based solutions are limited; some do not modify the corpora of standard LLMs [8], while others rely on data obtained from a single system (HDFS [12]) or a single data type (e.g., ADRs [10, 11]). Despite the advances these works incorporate — such as prompt engineering, fine-tuning, and benchmarking against various models — *ArchSec.AI*'s novelty lies primarily in its use of formal, reliable methods to build its corpus. Furthermore, there is a lack of existing work that bridges the architectural knowledge found in the literature with solutions for the security domain.

This research gap motivated the conception of a RAG-powered assistant grounded in a curated, domain-specific corpus. Using this specialized repository, *ArchSec.AI* is designed to recommend optimal architectural security strategies tailored directly to the specific threat landscapes and design challenges identified by the practitioner. To highlight the importance of this approach, our results have yielded two primary contributions: (i) preliminary evidence exposing deficiencies and knowledge boundaries of general-purpose LLMs when tasked with suggesting specific security architectural patterns, and (ii) a robust corpus of architectural security decisions curated and validated through a rigorous systematic mapping process. Our forward-looking idea capitalizes on these two main contributions: the development and evaluation of an assistant that is capable of helping the decision-making process of architects, researchers, and practitioners in consulting architectural knowledge created specifically for solving security problems.

The remainder of this paper is organized as follows. Section 2 presents the theoretical background. Section 3 presents the related work. Section 4 presents the method and results from the exploratory study. Section 5 discusses the method and results from the corpus construction and assessment. Section 8 discusses the

main limitations of the study. Finally, Section 9 concludes the paper with final remarks.

2 Background

2.1 Security

According to ISO/IEC 25010 standard, *security is a product quality characteristic representing the degree to which a product or system protects information and data so that persons or other products or systems have data access appropriate to their types and levels of authorization* [23]. To guarantee the security since early-steps of software development, we need implement certain defenses against attacks: **Accountability**, which is the capability of a product to enable actions of an entity to be traced uniquely to the entity [23]; **Authentication**, the capability of a product to prove that the identity of a subject or resource is the one claimed [23]; **Authorization**, which includes the granting of access based on access rights [22]. Besides these attributes, security can also be grounded in the CIA triad [5, 21]. The CIA triad represents the three pillars of information security: **Confidentiality**: The property that data are protected from unauthorized access, including means for protecting personal privacy and proprietary information; **Integrity**: The property that guard data against improper information modification or destruction and ensuring information non-repudiation and authenticity; and **Availability**: The property that ensures timely and reliable access to and use of information.

2.2 Architectural Patterns for Security

Architectural patterns provide a mechanism to capture domain experience and knowledge to allow it to be reapplied when a new architectural problem is found [28]. A pattern can be considered as a package of design decisions that are found repeatedly in practice [19]. Such patterns have known properties that permit reuse, besides that they describe a class of architectures [5]. According to Schumacher et al. [29], a pattern usually includes elements such as the *pattern name, alternative names, motivating examples, context, problem description, solution, structural and behavioral specifications, implementation guidelines, solved examples, variants, known uses, consequences, and references to related patterns.* Together, these sections provide both conceptual and practical guidance for applying the pattern in real-world systems. However, depending on the application scenario and the level of detail required, some sections may be adapted or omitted.

Similarly, security patterns describes a solution to a recurrent problem in order to stop or mitigate a set of specific threats through the implementation of a security mechanism, defined in a given context [29]. In short, security patterns are architectural patterns that implement security attributes [5]. As in the case of Guard Check [4] (authorization, authentication and integrity), Alignment-based Translation (availability) [17], Secure Channel (confidentiality) [14], Security needs identification for assets (confidentiality, integrity, availability and accountability) [3], and Full View With Errors (authorization and access control) [34].

2.3 Retrieval-Augmented Generation

LLMs still face significant limitations, especially in domain-specific tasks [25], notably producing "hallucinations" [35], fabricating answers instead of acknowledging that they don't know the right answer. To overcome challenges, RAG [26] enhances LLMs by retrieving relevant document chunks from external knowledge base, giving it the context it needs to give correct and relevant answers [16]. These documents are stored into a vector database, optimized for storing and retrieving textual data for immediate use [16]. Like any other LLM, RAG applications operates from prompts — text in natural language with instructions that guide its generation of responses. The content and clarity of these prompts can directly influence the quality and accuracy of the generated responses, and variations in formulation, level of detail, and use of specific linguistic cues can significantly alter the results [2].

3 Related Work

There is a recent interest in leveraging LLMs to support various activities within the software engineering activities, and more recently to assist software design process. Regarding concrete design-level support, Colombo *et al.* (2025) [8] investigated the potential of LLMs to assist designers in applying object-oriented design patterns. The core of their evaluation centered on whether LLM-based tools can effectively guide practitioners in selecting and implementing design patterns during the early stages of a software project. Through a comparative analysis of experiments across different LLM platforms, the authors in [8], highlighted the models' capacity to support architectural design. Notably, they found that consensus among LLMs is a strong indicator of a suggestion's validity, thereby offering a viable filtering mechanism for software architects in practice. In the context of software architecture design, Dhar *et al.* (2024) [11] conducted an exploratory study to investigate the efficacy of LLMs in generating Architectural Decision Records (ADRs), documents that encapsulate the context, rationale, and implications of architectural choices. Their findings indicate that while LLM-generated decisions are accurate and contextually relevant, they do not yet match human performance. Complementing, Díaz-Pace *et al.* (2024) [12] proposed ArchMind, an LLM-based assistant powered by Retrieval-Augmented Generation (RAG). ArchMind is designed to assist architects by recommending alternative design patterns for specific requirements, evaluating and ranking these options based on their trade-offs, auditing selected decisions, and automatically structuring the resulting information into an ADR template. Results showed that the generated ADRs were concrete and well-justified in their design rationale, although they tended to miss system-specific details.

While these foundational studies demonstrate that equipping LLMs with domain-specific knowledge yields significant advantages, e.g., streamlining pattern retrieval [8], automating artifact generation (ADRs) [11], and enhancing architectural reasoning (trade-off analysis of recommended patterns) [12], two critical limitations remain unresolved: (1) Limitations regarding the knowledge corpus used for the solutions and tools proposed in these studies—such as relying on a single data source, using only the default datasets inherent to base LLMs, or focusing on information about

a single system—without accounting for the diversity of information that other data sources could provide. (2) Lack of reliability in the methods used to construct the corpora. Furthermore, this research highlights an important boundary condition in model capability: while pure LLMs perform adequately when recommending ubiquitous, highly documented architectural patterns, their performance degrades sharply when confronted with specialized domains. For nuanced scenarios requiring domain-specific security patterns, such as those tailored for cloud, microservices, blockchain, robotics, aeronautics, or IoT ecosystems, the models exhibit constrained knowledge boundaries due to the limitations of their underlying training data. Crucially, to the best of our understanding, there is a prominent research gap in the state of the art: a lack of consolidated knowledge on the efficacy of LLMs in retrieving and consulting accurate architectural design strategies for software security concerns.

4 Exploratory Study

The primary **objective** of this exploratory study is to evaluate the capability and feasibility of LLMs in recommending appropriate architectural design patterns when presented with distinct software security contexts and problem scenarios. This section details the empirical setup structured around its core research questions, target objects, experimental factors, hypotheses, and variable definitions, as suggested in [33].

4.1 Experimental Design

Questions: To guide our empirical evaluation, we formulate the following research questions: ***RQ1.*** *How accurate are LLMs in recommending architectural patterns to mitigate specific software security problems?; and* ***RQ2.*** *How diverse are the responses provided by LLMs when asked about academic sources for suggested security patterns?*

The **experimental objects** consist of two state-of-the-art, general-purpose Large Language Models (LLMs): Google Gemini (v. 2.5-flash) and DeepSeek (v. V4-flash). These models were selected for their availability via web-based interfaces and their representation of diverse architectures and providers. By focusing on general-purpose, commercially available LLMs, we aimed to simulate real-world usage conditions [31]. This study's researchers acted as principal **subjects** in this study, which executed the intervention. The **intervention** involves executing security pattern recommendation requests across both models using the structured *Prompt Template #1 and #2*. During all prompt executions, the **controlled variables** were: models versions, prompt templates, none hyper-parameters use (e.g., temperature, top-k, top-p, etc.), and model subscriptions (basic for both models). As **factors** we used the variables: person of interest, technical domain, application domain, security requirement, and security problem, which are highlighted in blue color in both prompt templates. Such variables were selected by being common part of any architecture pattern documentation. The **treatments** (or factors' values) were obtained from descriptions of five distinct security architectural patterns: *Guard check, alignment-based translation, secure channels, security needs identification for assets, and full view with errors.* Such patterns were randomly selected from secondary studies [4, 18, 20], using a Python script available in a Zenodo repository (Section 9). Factors' information were extracted from those patterns and imputed as variables values in prompt templates. As **dependent variables** we observed the LLMs outcomes, i.e., security pattern suggestion as response for prompts requests. **Control group:** Specifications of five security patterns were used as control group. Pattern descriptions contain information listed in Section 2.2.

Prompt Template #1

I am a <person of interest> and I need to design a <technical domain> software system for <application domain> that implements <security> requirement. Can you recommend an architectural security pattern reported in the literature to solve <security problem>?

Prompt Template #2

I am a <person of interest> and I need to design a <technical domain> software system for <application domain> that implements <security> requirement. Can you recommend an architectural security pattern reported in the literature to solve <security problem>? *Cite the bibliographic references that support your suggestion.*

4.2 Experiment Execution

Table 1 shows the results of executing the five treatments in this exploratory study. Each treatment corresponds to adaptations of both prompt templates with information of an expected security patterns. The adapted prompts were executed in both models Gemini and DeepSeek. Suggested security patterns (outputs) by each LLM were compared with information from the expected pattern (from control group) given the context and problem prompted to them. LLMs answers were summarized using the coding method [30] for comparison intents. This activity was carried out by two researchers with experience in architecture patterns and security. Complete LLMs outputs and control group (five patterns) descriptions are available in [32].

4.3 Results Analysis and Discussion

The use of LLMs for suggesting security architectural patterns presents several limitations related to reliability and trustworthiness. Common issues include **hallucinations**, such as invented references, nonexistent terms, and even fictitious security patterns, which may mislead users and compromise decision-making. To determine whether a response was a hallucination, comparisons were made with the information contained in the provided references, alongside individual searches to understand the information supplied by the LLM—essentially comparing the content of the original source material that established the pattern against the responses provided. In addition, LLMs often produce **inconsistent and imprecise results**, where the same question asked multiple times can generate different recommendations. This lack of consistency requires users to continuously verify and validate the generated content, increasing the effort needed to use these systems safely.

Table 1: Exploratory study results.

Treatment	LLM	Output of prompt #1	Observation	Output of prompt #2	Observation	Expected Pattern
T1	Gemini	'Design by Contract'.	The architectural style exists, but hallucinated description.	'Design by Contract'.	Hallucinated pattern and references.	Guard Check [4]
	Deepseek	'Guardian', 'Secure Access Control', and 'Authorization-Integrity Wrapper'.	Hallucinated patterns.	'Checks-Effects-Interactions'.	Existing pattern, hallucinated references and description.	Guard Check [4]
T2	Gemini	'Secure Semantic Gateway', 'Semantic Mediation Gateway', 'Trusted Semantic Intermediary'.	Hallucinated patterns.	'Secure Semantic Mediator', 'Trusted Semantic Gateway'.	Hallucinated patterns and references.	Alignment-based Translation [17]
	Deepseek	'Secure Gateway', 'Semantic-aware Gateway with Policy Enforcement', 'Semantic Firewall', 'Ontology-based Security Gateway', 'Ontology-based Mediation Gateway' and 'Semantic Security Mediator'.	Hallucinated patterns.	'Semantic Gateway as a Service', 'Trusted Third-Party Mediation', 'Semantic Gateway with Secure Mediation'.	Hallucinated patterns and references.	Alignment-based Translation [17]
T3	Gemini	'Secure Channels'.	Correct pattern, partially hallucinatory description.	Answer: 'Secure Tunnel', 'Encrypted Tunnel', 'VPN Gateway'.	Hallucinated patterns and references.	Secure Channels [14]
	Deepseek	'Secure Channel', 'Protected Communications', 'Message Confidentiality', 'End-to-End Encrypted Media & Secured Signaling'.	Partially hallucinated patterns.	'Datagram Transport Layer Security – Secure Real-time Transport Protocol', 'Secure Session Layer', 'Encrypted Channel'.	Existing protocols, but hallucinated patterns and references.	Secure Channels [14]
T4	Gemini	Sherwood Applied Business Security Architecture Framework.	The framework exists, the description is hallucinated.	'Risk Assessment'.	Existing concept, but hallucinatory patterns and references.	Security needs identification for assets [3]
	Deepseek	'Security Requirements Elicitation', 'Security Decision Making', 'Security Strategy', 'Protection Poker' and 'Security Quality Requirements Engineering'.	Hallucinated patterns.	'Security Requirements Decomposition and Misuse Case', 'Misuse-Driven Security', 'Liability-Driven Security', 'Business-Oriented Security', 'Business-Case-First Security Misuse Case'.	Hallucinated patterns and references.	Security needs identification for assets [3]
T5	Gemini	'Policy Enforcement Point', 'Policy Decision Point'.	Pattern 1 exists, hallucinatory description.	'Policy Enforcement Point', 'Policy Decision Point', 'Externalized Authorization'.	Pattern 1 exists, description and references are partially hallucinatory.	Full View With Errors [34]
	Deepseek	'Attribute-Based Access Control (ABAC)'.	Existing pattern, but hallucinatory description.	'Attribute-Based Access Control (ABAC)'.	Existing pattern, but description and references are hallucinated.	Full View With Errors [34]

Importantly, these limitations are not exclusive to the recommendation of security patterns, but are broadly observed in the use of conversational AI systems in general. Mitigating these problems requires greater control over the LLM architecture, the use of curated and attached knowledge sources, and the application of optimized prompting strategies, often supported by Retrieval-Augmented Generation (RAG) approaches.

5 Corpus Construction

To overcome the limitations of using pure LLMs to suggest security architecture patterns, we propose establishing a corpus of security-related strategies for software architectures. This section aims to describe the methods and results of constructing such a corpus.

Firstly, a systematic mapping, combined with the snowballing technique, was conducted to establish the state of the art of architectural strategies for security and, consequently, to establish a corpus. The guidelines of [27] were followed. As a result, a final set of 112 primary studies was selected, each assembling different strategies, including 98 security patterns, 10 tactics, and 4 principles. Three researchers with expertise in software architecture and security strategies were responsible for executing this step. Results of this mapping are available in a Zenodo repository(Section 9).

For the data extraction phase, a form was created and the following information was collected: *Study ID, Year of Publication, Title, type of security architectural strategy reported, Security attributes related to each reported strategy, Application Domain, Technical domain, Detail of the techniques and methods used to identify architectural strategies, Detail of the methods for assessment of the classification, Limitations of the study, Contributions of the study, Relationship mapping between strategies, Strategies Aliases', and information about the quality of the study.* After this activity, the information was tabulated in a spreadsheet (Section 9).

Once information extraction from the primary studies was complete, the next step was to collect detailed information on individual strategies proposed in each study. This information was sometimes contained in the work itself that proposed the strategies (e.g., [34])

or in other referenced sources (e.g., [17]). The data extracted from each strategy (i.e., pattern, tactic, principle) were manually formatted, organized into editable text documents, and stored in a private cloud repository. Duplicate strategies were handled by removing strategies that belonged to the same bibliographic source. As a result, the security architectural strategies corpus was created.

5.1 Experimental Design

The main **goal** of this new experiment was to determine whether LLMs, when used with the corpus, improve the accuracy of security strategy recommendations compared with general LLMs (without a specialized corpus). The design of this second experiment is the same as the one defined in Section 4, with the only difference being that the **experimental object** was *NotebookLM* enhanced with the corpus. NotebookLM [1] was selected for this exploratory study because it has RAG capabilities and is a specialized tool for corpus analysis, offering a closed-loop, source-grounded framework optimized for empirical accuracy and rigorous documentation.

5.2 Experiment Execution

The execution phase of this study was operationalized through a two-stage process: querying and data extraction. The corpus with 98 different security patterns was ingested into the Google NotebookLM Pro platform. Following the environment setup, the experimental Prompt Template #1 was executed. The identical set of five treatments (contextual problems) was formulated. Because NotebookLM natively handles session context and grounding, each query was processed under uniform environment settings. Finally, for each treatment, the generated security patterns recommendations (LLM output) were extracted and systematically archived in an external spreadsheet to facilitate subsequent qualitative analysis and side-by-side comparison with the Gemini and Deepseek (general-purpose LLM) outputs. NotebookLM outputs are available in [32].

6 LLM with Corpus Assessment Results

The evaluation of the responses from Corpus-augmented NotebookLM was conducted qualitatively. This activity was carried out by two experienced individuals in the areas of architectural patterns and security. The results were summarized using the coding method [30], resulting in Table 2.

The comparative evaluation between general-purpose LLMs (Gemini and DeepSeek) and Corpus-enhanced LLM (NotebookLM + security strategies corpus) indicates that the accuracy of LLMs fluctuates based on the abstraction level of the prompt. Specifically, general models can perform well when presented with generic problems that are ubiquitously documented in their training datasets [8]. For instance, in the treatment 5 scenario, although the LLMs failed to recommend the exact target pattern (*Full View With Errors*), their alternative suggestions (*Policy Enforcement Point* and *Attribute-Based Access Control*) are related patterns but did not offer a solution to the intended problem. Compared with an LLM's assessment on a corpus, constraining it within a specialized knowledge repository yields enhanced suggestions and greater verifiability (i.e., reference traceability) than general-purpose LLMs. Thus, by anchoring the LLM's generative process to a validated corpus with RAG strategies, the solution successfully mitigated the risk of hallucinations and provided actionable architectural recommendations.

[1] https://workspace.google.com/products/notebooklm/

7 The *ArchSec.AI* Proposal

Figure 1 depicts the *ArchSec.AI*'s intended structure. The data sources (security patterns, tactics, and principles documents) are passed to an ingestion pipeline that cleans and chunks the documents and feeds such data into a vector database, i.e., the knowledge corpus construction. Furthermore, the appropriate LLM will be selected to compose *ArchSec.AI*. It is expected *ArchSec.AI* operates by: (1) Receiving a user query, i.e., security requirement, context, or problem; (2) Retrieving relevant information from the security architectural strategies corpus; (3) Incorporating this retrieved knowledge into the prompt sent to a language model; (4) Generating recommendations that provide insights into suitable architectural strategies for security.

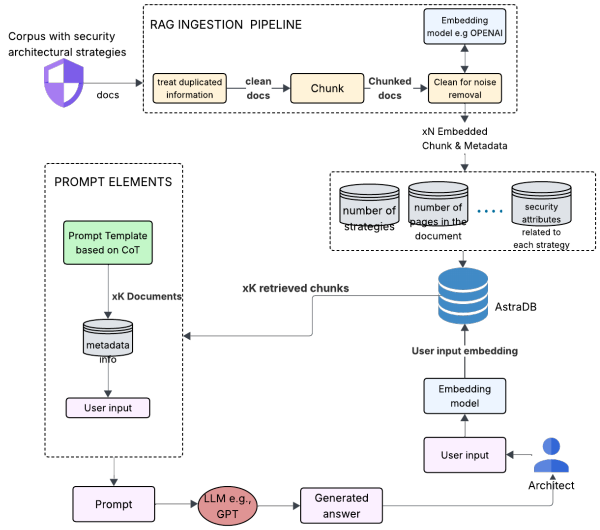

Figure 1: *ArchSec.AI* internal elements. Adapted from [24]

Future research will focus primarily on the full implementation, deployment, and empirical validation of *ArchSec.AI*, using the LangChain[2] framework. To assess the performance of the *ArchSec.AI*, we will conduct a statistical evaluation using quantitative data from the RAGAS framework (LLM-as-a-judge) [13]. RAGAS would evaluate results along the metrics: *faithfulness, answer relevance, context precision and context recall.*

8 Threats to Validity

This section discusses threats to the validity of this study, categorized into construct, internal, and external validity [33], along with mitigation strategies. **Construct Validity:** We identified two primary threats in this category: *(i) Subjectivity in Qualitative Assessment:* The assessment of the models' recommendations, across

[2] https://www.langchain.com/

Table 2: Experiment Results with NotebookLM and the Proposed Corpus

Treatment	LLM	Output	Observation	Expected Pattern
T1	NotebookLM + Corpus	'Guard Check'.	Correct answer and references.	Guard Check [4]
T2	NotebookLM + Corpus	'Alignment-based Translation'.	Correct answer and references.	Alignment-based Translation [17]
T3	NotebookLM + Corpus	'Secure Channels', 'Secure VoIP call','VoIP Tunneling'.	Correct answer and references. Additional patterns fit the problem outlined in the question and can be considered candidate patterns.	Secure Channels [14]
T4	NotebookLM + Corpus	'Security needs identification for assets'.	Correct answer and references.	Security needs identification for assets [3]
T5	NotebookLM + Corpus	'Full View With Errors', 'Limited View', 'Reference Monitor'.	Correct answer and references. Additional patterns fit the problem outlined in the question and can be considered candidate patterns.	Full View With Errors [34]

both the LLM experiment and the corpus-grounded NotebookLM iteration, was initially conducted by a single researcher. To mitigate this threat and reduce potential bias, a senior researcher with extensive expertise in software architecture and security independently reviewed and validated the qualitative results; and *(ii) Corpus Comprehensiveness and Representativeness:* Our corpus may not encapsulate every security pattern existing in the literature, as non-peer-reviewed or gray literature was excluded. While capturing all sources would require a multivocal review methodology, we intentionally prioritized peer-reviewed articles to ensure high scientific validity. To mitigate the risk of unrepresentative sampling, we employed an exhaustive search strategy, performing two consecutive iterations of forward and backward snowballing during our systematic mapping process to maximize corpus coverage. **Internal Validity:** The following factors were identified: *Prompt Engineering Constraints and Model Hyperparameters:* The design of prompt templates represents a critical variable, as alternative methods could yield different architectural recommendations. Our study relied strictly on zero-shot prompting, omitting techniques such as few-shot prompting or Chain-of-Thought prompting. Consequently, the generated outputs lack the rigid structural fields typical of architectural patterns (e.g., explicit context, problem, solution, consequences, and related patterns). We also did not manipulate internal model hyperparameters, such as temperature or top-p sampling. To mitigate these threats, during the development of *ArchSec.AI*, we will test different prompting strategies as well as parameter adjustments. **External Validity:** The following conditions restrict the scope of our results: *(i) Scale of Objects and Instruments:* The abstraction of our conclusions is constrained by the scale of the experiment, which evaluates five specific security patterns across three distinct AI instruments (Google Gemini, DeepSeek, and NotebookLM Pro with Corpus). However, to avoid selection bias, the security patterns were randomly selected from the whole set of 98 patterns obtained in our systematic mapping; *(ii) Domain Specificity:* This research focuses exclusively on consulting architectural knowledge tailored for software architecture security. Generalizations regarding applicability to other software engineering phases (e.g., requirements engineering, testing) and other attributes (e.g., performance, maintainability) were not planned and remain unexplored, as they are outside the scope of this research. However, while this study focused solely on security patterns, the complete corpus already includes information about tactics, principles, styles, and a range of other security-related quality attributes (e.g., privacy, authentication, authorization). In addition, it covers various applications and technical domains, as well as security-specific topics such as threat patterns, anti-patterns, and weaknesses.

9 Conclusion

Our results demonstrate that general-purpose LLMs frequently hallucinate under specialized architectural and security constraints. However, grounding LLMs with RAG capabilities on a dedicated knowledge corpus improves recommendation accuracy. These findings support the idea that an integrated assistant like *ArchSec.AI* is required to unify and query the corpus. As future work, the *ArchSec.AI* will be developed following a standard process to create AI assistants based on RAG and specialized knowledge corpus [24].

Artifact Availability

Artifacts are available as a Zenodo repository [32]: https://doi.org/10.5281/zenodo.22116001. It contains all the code scripts developed to conduct the exploratory study, the resulting experimental data, as well as the body of knowledge regarding architectural security patterns. The artifact is structured to support inspection and reuse.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, the PRPI-USP (Grant number: 22.1.09345.01.2), and the CNPq (Grant nº 406941/2025-4).

References

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

[2] Jay Alammar and Maarten Grootendorst. 2024. *Hands-on large language models: language understanding and generation.* " O'Reilly Media, Inc.".

[3] Aleem Khalid Alvi and Mohammad Zulkernine. 2011. A natural classification scheme for software security patterns. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 113–120.

[4] Sadaf Azimi, Ali Golzari, Naghmeh Ivaki, and Nuno Laranjeiro. 2025. A systematic review on smart contracts security design patterns. *Empirical Software Engineering* 30, 4 (2025), 95.

[5] Len Bass, Paul Clements, and Rick Kazman. 2012. *Software Architecture in Practice: Software Architect Practice_c3.* Addison-Wesley.

[6] Cyber Security Brazil. 2026. Clientes do Google Cloud relatam cobranças de até US$ 17 mil após abuso de chaves API vazadas — cybersecbrazil.com.br. https://www.cybersecbrazil.com.br/post/clientes-do-google-cloud-relatam-cobran%{C}3%{A}7as-de-at%{C}3%{A}9-us-17-mil-ap%{C}3%{B}3s-abuso-de-chaves-api-vazadas. [Accessed 21-05-2026].

[7] Cyber Security Brazil. 2026. Estudante em Taiwan é acusado de usar equipamento de rádio para paralisar trens-bala — cybersecbrazil.com.br. https://www.cybersecbrazil.com.br/post/estudante-em-taiwan-%{C}3%{A}9-acusado-de-usar-equipamento-de-r%{C}3%{A}1dio-para-paralisar-trens-bala. [Accessed 21-05-2026].

[8] Vitor Monteiro Colombo, Weslen Schiavon de Souza, Lisane Brisolara de Brisolara, Brenda Salenave Santana, and Tatiana Aires Tavares. 2025. Avaliação do uso de LLMs para Suporte à Tomada de Decisão na Adoção de Padrões de Projeto em Software Orientado a Objetos. In *Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS)*. SBC, 101–111.

[9] Niladri Sekhar Dash and S. Arulmozi. 2018. *Definition of 'Corpus'*. Springer Singapore, Singapore, 1–15. doi:10.1007/978-981-10-7458-5_1

[10] 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).

[11] Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. 2024. Can llms generate architectural design decisions?-an exploratory empirical study. In *2024 IEEE 21st International Conference on Software Architecture (ICSA)*. IEEE, 79–89.

[12] 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.

[13] 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.

[14] Eduardo B Fernandez, Juan C Pelaez, and Maria M Larrondo-Petrie. 2007. Security patterns for voice over ip networks. In *2007 International Multi-Conference on Computing in the Global Information Technology (ICCGI'07)*. IEEE, 33–33.

[15] Eduardo Fernandez-Buglioni. 2013. *Security patterns in practice: designing secure architectures using software patterns.* John Wiley & Sons.

[16] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* 2, 1 (2023).

[17] Emiliia Geloczi, Felix Klement, Eva Gründinger, and Stefan Katzenbeisser. 2023. Secure Stitch: Unveiling the Fabric of Security Patterns for the Internet of Things. In *International Workshop on Security and Trust Management*. Springer, 65–84.

[18] Munawar Hafiz, Paul Adamczyk, and Ralph Johnson. 2011. Patterns transform architectures. In *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*. IEEE, 242–251.

[19] Neil B Harrison, Paris Avgeriou, and Uwe Zdun. 2007. Using patterns to capture architectural decisions. *IEEE software* 24, 4 (2007), 38–45.

[20] Atsuo Hazeyama. 2012. Survey on body of knowledge regarding software security. In *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE, 536–541.

[21] National Institute if Standards and Technology (NIST). 2020. Executive Summary - NIST SP 1800-25 documentation. *NIST SP 1800-25* (2020).

[22] ISO. 1989. Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 2: Security Architecture. *ISO 7498-2:1989* (1989).

[23] ISO. 2023. Systems and software engineering Systems and software Quality Requirements and Evaluation (SQuaRE)- Product quality model. *ISO/IEC 25010:2023(E) (Revision of ISO/IEC 25010:2011)* (2023).

[24] Paul Iusztin, Maxime Labonne, and Alex Vesa. 2024. *LLM Engineer's Handbook: Master the Art of Engineering Large Language Models from Concept to Production.* Packt Publishing Limited.

[25] Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large language models struggle to learn long-tail knowledge. In *International conference on machine learning*. PMLR, 15696–15707.

[26] 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.

[27] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic mapping studies in software engineering. In *12th international conference on evaluation and assessment in software engineering (EASE)*. BCS Learning & Development.

[28] Roger S Pressman. 2005. *Software engineering: a practitioner's approach.* Palgrave macmillan.

[29] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. 2013. *Security Patterns: Integrating security and systems engineering.* John Wiley & Sons.

[30] Mai Skjott Linneberg and Steffen Korsgaard. 2019. Coding qualitative data: A synthesis guiding the novice. *Qualitative research journal* 19, 3 (2019), 259–270.

[31] Mohamed Soliman, Elia Ashraf, Kamel MK Abdelsalam, Jan Keim, and Ashwin Prasad Shivarpatna Venkatesh. 2025. LLMs for Software Architecture Knowledge: A Comparative Analysis Among Seven LLMs. In *European Conference on Software Architecture*. Springer, 99–115.

[32] Rebeca M. P. Sombra and Lina Garcés. 2026. *ArchSec.AI: Towards a LLM-based assistant to support analysis of security architectural strategies.* doi:10.5281/zenodo.22116001

[33] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2012. *Experimentation in software engineering*. Vol. 236. Springer.

[34] Joseph Yoder and Jeffrey Barcalow. 1997. Architectural patterns for enabling application security. In *Proceedings of the 4th Conference on Patterns Language of Programming (PLoP'97)*, Vol. 2. Citeseer, 30.

[35] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2025. Siren's song in the ai ocean: A survey on hallucination in large language models. *Computational Linguistics* (2025), 1–45.