




# Helping novice architects to make quality design decisions using an LLM-based assistant

J. Andrés Díaz-Pace<sup>1</sup>, Antonela Tommasel<sup>1</sup>, and Rafael Capilla<sup>3</sup>

<sup>1</sup> ISISTAN, CONICET/UNICEN, Tandil, Buenos Aires, Argentina  
{andres.diazpace,antonela.tommasel}@isistan.unicen.edu.ar

<sup>2</sup> Rey Juan Carlos University, Madrid, Spain  
rafael.capilla@urjc.es

**Abstract.** Architectural knowledge and specifically design decisions have become first-class entities to be captured routinely in a design process. However, the quality of the decisions captured is often low. Part of the problem is that reflections intended to criticize, and thus improve, the decisions are seldom made in architecture teams, particularly when involving novice architects. To improve reflective practices and capture better decisions, we propose a design assistant approach based on generative AI techniques. Our assistant, called **ArchMind**, relies on two information sources: architectural knowledge about patterns, and information about the system under design. Furthermore, the assistant takes advantage of LLMs to progressively aid users in selecting and assessing alternative decisions, until capturing them using an Architecture Decision Record format. **ArchMind** mainly targets novice architects as discussed in an initial experiment using the assistant off-line for a classroom project. The generated ADRs were concrete and well-justified in their design rationale, although they tended to miss system-specific details.

**Keywords:** Design decisions · Architectural knowledge · Assistant · Large Language Models · Reflection · Architecture Decision Records.

## 1 Introduction

For over 20 years, the software architecture community has emphasized the need to document key design decisions in software projects. Diverse tools and techniques have been developed to capture, manage, and share architectural knowledge (AK) in academia and industry. However, architectural decision-making involves more than capturing AK. Architects often start by deriving decisions for software requirements and then evaluate alternative choices based on their pros and cons. Once discussed, decisions can be documented using templates such as the Architecture Decision Record (ADR).

Related to the deliberative process in which decisions are discussed, Razavian et al. [6] proposed the *Mind1-Mind2* model, in which some architects make decisions (*Mind1* stage) while others criticize those decisions (*Mind2* stage). The latter refers to as *cognitive architects* and their reflective tasks aim at validating decisions, identify overlooked aspects, and enhance the overall quality

of the decisions. In practice, less experienced architects tend to struggle with reflection about design decisions. Some challenges include identifying good alternative decisions (e.g., different architectural styles or design patterns), weighing quality-attribute trade-offs, and effective comparison of design alternatives. These situations also appear frequently in architecture training courses. In this context, we argue that generative AI techniques, like Large Language Models (LLMs) can assist by suggesting alternatives and evaluating their pros and cons. Additionally, once contextual information and potential decisions are gathered, an LLM could generate preliminary ADRs [3].

This paper investigates an approach to support novice architects in decision-making activities by relying on a set of LLM-based agents, called *copilots*, that act as assistive cognitive architects within the *Mind1-Mind2* model. We present a prototype called **ArchMind** that uses two information sources: *existing AK on software patterns*, and *the system under design* (e.g., context, requirements, past decisions). Using LLM techniques like prompting and Retrieval-Augmented Generation (RAG) [4], **ArchMind** assists architects with tasks such as suggesting alternative patterns for a requirement, evaluating and ranking them based on pros and cons, assessing about chosen decisions, and structuring the information using an ADR template. Our approach enables architects’ interventions to adjust the inputs for the tasks managed by the copilots. This process builds an ADR progressively, enabling the architect and copilots to focus on different aspects of the final decision document. We discuss initial findings from comparing ADRs generated by **ArchMind** with those produced in an architecture classroom.

## 2 Background

The *Mind1-Mind2* model has been used in literature to study the dynamics of the software architecture process [6]. Capilla et al. [2] implemented a model with undergraduate students in a software design course. The students assumed one of three roles: senior architects making decisions (*Mind1*), cognitive architects providing critique (*Mind2*), and junior architects documenting decisions. A typical exercise involved considering design patterns to meet a functional requirement. The study showed that reflective tasks somehow improved the final architecture, and the architects’ interactions contributed to a thorough decision analysis. However, in certain cases the students’ bias may have limited the exploration of alternatives, thus affecting decision quality. This factor motivated our proposal to use generative AI to enhance the reflective practices.

**AI-assisted decision-making** In software engineering, AI assistants and LLMs have been commonly used for implementation activities [5], but their application to architecture design remains limited [3]. For instance, White et al. [7] introduced a prompt pattern catalog for activities such as requirement elicitation, system design, and refactoring. Ozkaya [5] discusses how generative AI can help develop or refine system architectures, aligning with our goal of using an LLM tool to support better decision-making. However, crafting suitable prompts and other LLM abstractions for architectural tasks is not simple and depends on the knowledge base provided to the tool. Moreover, human oversight is crucial to mitigate “hallucinations” in the LLM outputs.

**LLMs and Retrieval Augmented Generation** In recent years, LLMs have been trained on extensive datasets and fine-tuned with human instructions [1], enabling them to understand and respond to natural language questions. LLMs are particularly useful for in-context learning. Interacting with LLMs typically involves instructions (prompts), which combine user-specific details and relevant context for the LLM to generate responses. In the architecture domain, a prompt might include functional requirements, system context, or preferences for solutions from a pattern catalog (e.g., the Gang-of-Four book). By processing this information, an LLM can infer a design decision and document it using an ADR.

LLMs often have limitations for dealing with factual information in tasks that require specific domain expertise. A promising technique here is retrieval-augmented generation (RAG), which integrates external domain-specific sources into the LLM generative process to improve response accuracy and relevance. In essence, a RAG helps ground the LLM, making it focus on specific contents for its response [4]. For instance, in our architectural example, the Gang-of-Four catalog could serve as a foundation for an architecture-centric RAG framework.

### 3 Study Design

This work evaluates whether generative AI can enhance human decision-making and reflective practices based on prior results of novice architects. In this context, we investigated the quality of ADRs generated by an LLM-based assistant. Unlike previous research for generating ADRs in one single step [3], we adopted an *incremental* approach. Our method integrates various information pieces (e.g., architectural patterns, system details, assessment results) and allows architects to interact with the assistant to refine the final decision. The assistant consists of a set of *copilots*, each responsible for a specific design task, as depicted in Fig. 1. This modular schema facilitates reasoning by breaking the ADR generation process into simpler steps.

As an initial benchmark for human design decisions, we used a student project from an undergraduate architecture course at a Spanish university. In this project, a team of students captured their decisions as ADRs. For the sake of the study, these students were considered as novice architects. We developed a prototype called **ArchMind** to support five tasks including reflective practices. Once **ArchMind** was loaded with information from the students’ project, we carried out design sessions based on the *Mind1-Mind2* model. In these sessions, a human (acting as a senior architect) submits a requirement, and a copilot (acting as a cognitive architect) provides an initial decision. This decision is progressively refined by other copilots, with human interventions if needed until the final ADR is generated. We evaluated the resulting ADRs and compared them qualitatively to the students’ decisions.

We used both zero-shot and RAG strategies for our experiment. Zero-shot learning involves performing a task without prior exposure to related training examples. On the other hand, RAG [4] is a practical way of adding knowledge to an LLM, even if the knowledge might be already known to the model (e.g., the pattern catalogs in our domain). First, RAG ensures that the AK being

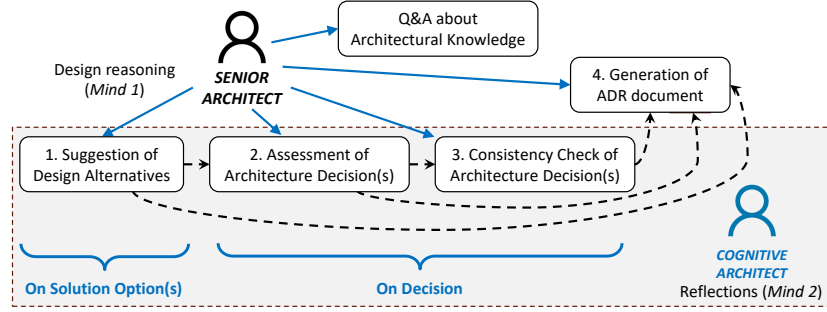


Fig. 1: Reflective tasks supporting architect’s in decision-making. The dotted arrows show the information dependencies between the tasks.

retrieved is relevant to the specific context or query, instead of relying on the model’s generalized knowledge, which might be too broad or inaccurate if the question is ambiguous. Second, in RAG, the information sources can be directly identified, providing greater traceability. Third, RAG directs the LLM to key pieces of information, improving the relevance and accuracy of the outputs.

### 3.1 Approach

To conduct a design session, we followed a four-step protocol involving a senior architect (the user<sup>3</sup>) and a cognitive architect (the copilots). Fig. 2 provides an example of the questions each copilot addresses. The user can intervene at specific points in the process to provide input or refine the copilots’ partial outputs. Initially, we fed the prototype with AK including architectural styles, design patterns, and microservice patterns from well-known sources<sup>4</sup>. This AK is required for the RAG but not for the zero-shot strategy. Additionally, we provided information about the system under design and its functional requirements. Given the system context  $S$  and a subset of requirements  $\{R\}$  (decision drivers), our goal was to yield a decision  $D$  through a series of interactions with the copilots, until producing an ADR for  $D$ . This process proceeded in iterations until all requirements were met. Each iteration involved 4 tasks:

**Task 1.** The user asks the *PatternSuggestion&Ranking* copilot to generate a list of design alternatives for  $\{R\}$  using any available patterns. Patterns are sourced either from the RAG knowledge base or the LLM (zero-shot). Afterwards, the copilot evaluates each alternative weighting its pros and cons, ranks the alternatives from best to worst (for  $S$  and  $\{R\}$ ), and discards any irrelevant pattern. The alternative ranked first is suggested as a candidate decision  $D$  for  $\{R\}$ .

**Task 2.** The *DecisionAssessment* copilot analyzes the previous  $D$  in more detail, focusing on quality attributes, risks, and subsequent decisions, among others.

<sup>3</sup> In our experiment, this role was played by two of the authors.

<sup>4</sup> We included *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma, Helm, Johnson, Vlissides), *Microservices Patterns: With examples in Java* (Richardson), *Design It!: From Programmer to Software Architect* (Keeling) and *Software Architecture in Practice* (Bass, Clements, Kazman, Safari).

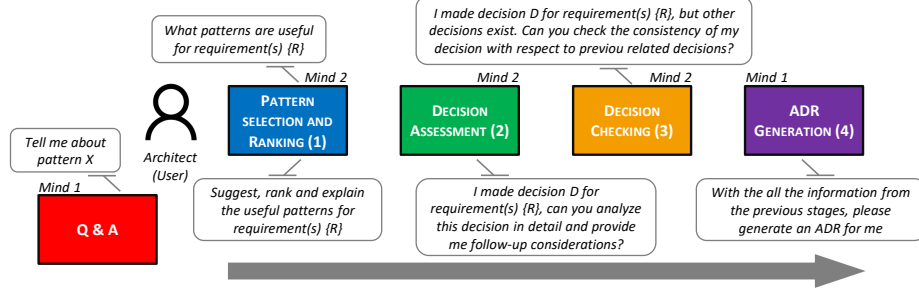


Fig. 2: Example of an architect’s journey and questions supported by each copilot (numbered boxes). The copilots have a one-to-one mapping with the tasks.

**Task 3.** The *DecisionChecking* copilot expands the analysis of  $D$  by ensuring its consistency with other related or committed decisions for  $\{R\}$ .

**Task 4.** The *ADRGeneration* copilot consolidates the analyses from tasks 1 – 3 and produces an ADR. Although one could attempt ADR generation after tasks 1 or 2, the copilot may lack enough information for a comprehensive ADR.

Finally, we included a *Q&A* task which allows users to ask questions about any architectural topic at any time. This task aims to assist about doubts or clarifications, but it is not mandatory for the generation of the ADR.

### 3.2 Target system

In our study, we chose a student project about a system whose requirements and decisions provide enough detail to compare it with ArchMind’s results. Details of the project materials and tool are provided in Section 5. The student team was composed by 6 participants, according to the two roles of the *Mind1-Mind2* model plus the role of *junior architect*, who was responsible for modeling the architecture in UML based on the decisions resulting from the senior and cognitive architects. Before making their decisions, participants elicited a list of functional requirements from the problem description. The target problem was about the migration of a monolithic system to a microservices solution. Subsequently, participants identified design issues hinted by the requirements and explored decisions for them. The decisions, originally written in Spanish, were translated into English to align with the AK sources and also for anonymization.

### 3.3 The ArchMind tool

The assistant comprises five copilots: *Q&A*, *PatternSuggestion&Ranking*, *DecisionAssessment*, *DecisionChecking* and *ADRGeneration*. Each copilot is accessible as a separate tab in the tool’s GUI, allowing users to navigate between them freely. Intermediate results are shared across tabs. Each copilot uses parametrized prompts for its LLM, summarized in Table 1. The tool is implemented using *Langchain*, *OpenAI GPT-3.5*, *Chroma* and *Streamlit*.

The tool allows configuring a RAG strategy for the first three copilots. In RAG, a copilot first uses similarity search to match its prompts to pieces from the knowledge base, called vectors, and then asks the LLM to generate a response. If

Table 1: Prompt instructions and parameters for the LLM in each copilot.

Copilot	Summary of instructions	Parameters
Q&A	<i>You are an architecture expert, answer question Q concisely</i>	Q: system-agnostic question
1. Pattern Suggestion	<i>Given a system context S and requirements R, provide a list of candidate patterns for S and R. For each pattern P, assess pros and cons. Create a ranking of the patterns and remove irrelevant patterns. For the first ranked pattern P, suggest decision D = P and instantiate it for requirements R and system context S</i>	S: system context. R: target requirement(s). List of candidate patterns (intermediate result)
2. Decision Assessment	<i>I want to better understand the implications of my decision D for requirements R in the provided context S. Please assess: appropriateness of D, clarifying questions, follow-up decisions, assumptions and constraints, consequences on quality attributes, potential risks and trade-offs</i>	S: system context. R: requirement(s). D: decision being assessed
3. Decision Checking	<i>Search for related decisions for my decision D for requirement(s) R, and check if past/related decisions lead to conflicts or inconsistencies</i>	S: System context. R: requirement(s). D: decision being assessed. Related decisions
4. ADR Generation	<i>Generate ADR for requirement(s) R based on initial decision D and using a MADR template. Leverage on prior assessment and consistency checking of D</i>	S: system context. R: requirement(s). D: decision being assessed. Assessment of candidates (pros & cons). Assessment of consistency of D with related decisions

no vectors can be retrieved for a prompt, the LLM call is skipped, and a default answer is returned. Conversely, with a zero-shot strategy, all prompts are directly sent to the LLM for responses. **ArchMind** has two repositories: one with pattern catalogs as the AK for the RAG and another containing information about the target system. To ingest the patterns as documents, they are divided into chunks and converted into embeddings, which are numeric vector representations of the text. These embeddings enable similarity search across documents in the RAG strategy. Users can select a specific pattern catalog or opt for a zero-shot mode.

The system repository provides descriptions (e.g., requirements, decisions) as input for the copilots. For example, in *Task 1*, a user can select a requirement and ask the *PatternSuggestion&Rank* copilot to explore and rank design options based on their pros and cons. A system summary provides context for the LLM to identify candidate options. Once a list of possible patterns is returned, the top-ranked pattern is suggested as the initial decision. The user can then change the patterns, the decision, or even write a new one if they disagree with the suggestion. Once an initial decision for the requirement is made, the user can ask the *DecisionAssessment* copilot for a more in-depth assessment. In *Task 2*, this task includes identifying assumptions, constraints, consequences over quality attributes, risks, trade-offs, and clarifying or follow-up questions for implementation. In *Task 3*, the *DecisionChecking* copilot retrieves semantically similar prior decisions from the system repository to check for potential conflicts or inconsistencies. Finally, the user can use the *ADRGeneration* copilot to request an ADR. In *Task 4*, the LLM employs a zero-shot strategy, combining the outputs from the previous copilots into an ADR document.

## 4 Initial Evaluation

In this research, our goal was to analyze the characteristics of the generated ADRs and the influence of the copilots' pipeline on them. As **ArchMind** is still

experimental, we did not aim to conduct a systematic comparison of prompting techniques or user studies. We started by exercising the tool on individual tasks to ensure each copilot could produce useful outputs, adjusting the prompts as needed. Once the AK repository was ready, we tested the *Q&A* copilot with various architectural questions. For the other copilots, we tested end-to-end generation scenarios for selected requirements. On the other hand, we implemented a batch procedure to take all the requirements from the target system through the copilot pipeline to generate 16 ADRs, using the RAG and zero-shot strategies.

We then analyzed the quality of these ADRs offline, evaluating the relevancy of the generated content compared to the ADRs produced by students<sup>5</sup>. We manually examined a sample of 5 key ADRs to identify similarities and differences. Based on our expertise, we judged most ADRs as correct in addressing decision drivers, with only a few misleading decisions. These issues arose because the LLM lacked sufficient system context compared to humans. For instance, students often derived a “problem statement” or design issue from the requirement before evaluating design options, which the current copilots did not capture. Regarding the search of the design options, a positive aspect of the copilots is that they suggested and justified more candidate patterns than humans. However, the copilots struggled to recognize when a functional module or a vendor-specific technology can be considered a convenient option.

For the RAG strategy, the LLM often provided no answers due to insufficient context, and when it did, the responses were specific to concrete patterns or system-related aspects. As a result, the ADRs from *ArchMind* were sometimes concise and stylistically different from the more verbose but less specific decisions and options considered by the students. In contrast, the zero-shot strategy always produced an output but with general comments. For example, several responses mentioned the broad “Microservices” pattern, which was not specific enough for our target system. Consequently, the generated ADRs had few design options and lacked detailed pros and cons of the patterns. One key lesson from this study is that the RAG strategy effectively focuses the design work on specific topics, such as those from our AK base. Thus, this strategy can benefit novice architects by improving their decision-making and assessment skills, and it can even be useful in real-life project scenarios. Our approach highlights a trade-off between specific (but constrained) copilots and more general ones, presenting a research challenge for using LLMs in the software architecture field. Another challenge involves accurately representing the system context and requirements as inputs for the copilots. During our experiments, a brief system context (about 10 lines of text) and short functional requirements (1-2 lines) were provided. However, further evaluation with more complex problems and human interactions is needed, considering variations in context (e.g., acceptance criteria, quality-attribute concerns) and AK sources.

---

<sup>5</sup> This does not mean that the ADRs produced by the students were considered as the ground truth but rather as a reference solution for comparison purposes.

## 5 Conclusions and Future Work

In this paper, we proposed an LLM-based approach and tool to assist architects in creating ADRs using zero-shot and RAG strategies. An initial offline study comparing human-generated and copilot-generated ADRs showed that a RAG strategy can yield more precise results than using a zero-shot approach. We conjecture that this advantage arises from using intermediate tasks and reflections, as facilitated by our copilots’ modular design. This approach holds potential for both architecture training and assessing design alternatives. We believe that LLM-based assistants can significantly enhance software architecture education. They can also automate decision-making and documentation tasks while also expanding human capabilities in exploring large spaces of design alternatives.

Related to our work, Dhar et al. [3] experimented with LLMs to generate design decisions. We differ from this approach in how we generate the ADRs, focusing on interactive and step-wise activities rather than on a single-shot effort. We provide more elaborate prompts and address a different research question.

For future work, we plan to conduct empirical experiments using ADRs for different projects and perform a deeper evaluation of the tool. This will include a qualitative, fine-grained analysis of human answers to refine the copilot inputs and prompts for the ADR sections. We will also investigate how copilots perform with alternative LLMs including multi-modal capabilities. Another research topic is the representation of AK for the RAG strategy.

**Data Availability** We provide a Github site with the source code and data used in the paper: <https://github.com/tommantonela/archmind>. A version of ArchMind can be run via *Streamlit*: <https://archmind.streamlit.app/>.

## Bibliography

- [1] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Nee-lakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. *Adv Neural Inf Process Sys* 33, 1877–1901 (2020)
- [2] Capilla, R., Zimmermann, O., Carrillo, C., Astudillo, H.: Teaching students software architecture decision making. In: *Software Architecture: 14th European Conference, ECSA 2020*. pp. 231–246. Springer (2020)
- [3] Dhar, R., Vaidhyanathan, K., Varma, V.: Can llms generate architectural design decisions?-an exploratory empirical study. *arXiv preprint arXiv:2403.01709* (2024)
- [4] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. *Adv Neural Inf Process Syst* 33, 9459–9474 (2020)
- [5] Ozkaya, I.: Can architecture knowledge guide software development with generative ai? *IEEE Software* 40(5), 4–8 (2023)
- [6] Razavian, M., Tang, A., Capilla, R., Lago, P.: In two minds: how reflections influence software design thinking. *J. Softw.: Evol. Process* 28(6), 394–426 (2016)
- [7] White, J., Hays, S., Fu, Q., Spencer-Smith, J., Schmidt, D.C.: Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. *CoRR abs/2303.07839* (2023)