LLMs for Software Architecture Knowledge: A Comparative Analysis Among Seven LLMs

Mohamed Soliman[1(✉)], Elia Ashraf[1], Kamel M. K. Abdelsalam[3], Jan Keim[2], and Ashwin Prasad Shivarpatna Venkatesh[1]

[1] Universität Paderborn, Paderborn, Germany
{mohamed.soliman,elia.ashraf,ashwin.prasad}@uni-paderborn.de
[2] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
jan.keim@kit.edu
[3] Ain Shams University, Cairo, Egypt
kamelmk@sci.asu.edu.eg

Abstract. Software developers require extensive architectural knowledge (AK) to effectively maintain and extend existing software systems. Recently, Large Language Models (LLMs) have demonstrated promising capabilities in learning from vast datasets, including software repositories, to provide insightful answers about existing software systems. With the development of various LLMs, each characterized by distinct sizes, architectures, and vendors, there is potential for these models to assist developers in addressing architectural queries related to AK. Despite the advancements, there is a limited understanding of the comparative performance of different LLMs, particularly regarding the accuracy and similarity of their responses to architectural questions. This paper aims to bridge this gap by evaluating seven diverse LLMs, including GPT, Mistral, LLaMA, and DeepSeek, focusing on their ability to accurately and consistently respond to queries about the AK of the open-source system, Hadoop HDFS. Our study reveals significant variations in the performance of these LLMs, highlighting differences in the accuracy and similarity of their responses. These findings provide valuable insights for software developers and researchers, guiding the selection and utilization of LLMs for architectural knowledge tasks in software development.

Keywords: Software architecture · Architectural design decisions · Architectural knowledge · LLMs · GPT · DeepSeek · LLaMA

1 Introduction

Software engineers need *Architectural knowledge (AK)* to develop and maintain software systems. This encompasses three *AK concepts* [27]: (i) *Components and connectors* (e.g., layers, remote calls) [3], (ii) *Quality solutions* (e.g., patterns [8], tactics [3]) that ensure attributes like maintainability and availability, and

V. Andrikopoulos et al. (Eds.): ECSA 2025, LNCS 15929, pp. 99–115, 2026.
https://doi.org/10.1007/978-3-032-02138-0_7

(iii) *Rationale behind design decisions* [38]. Each *AK concept* is important for software engineers: AK on *components and connectors* assists in understanding the structure and behavior of a system, AK on *quality solutions* supports quality improvements, and *design rationale* [38] facilitates making new design decisions.

Despite the importance of AK, retrieving AK for existing software systems remains a challenge due to its distribution. No single source holds complete AK, requiring software engineers to integrate information from various repositories (e.g., Git) [20], issue trackers [5,32], mailing lists [33], forums (e.g., Stack Overflow [11]), and blog articles [35]. This manual aggregation from these sources is complex and time-consuming and remains an open challenge.

To tackle this challenge, software architecture researchers (e.g., [13,15,18,19]) are now exploring the potential of Large Language Models (LLMs), such as GPT [30] and LLaMA [39], to automate AK extraction and integration. Due to advanced architectures and large-scale training on diverse datasets, LLMs demonstrate strong capabilities in text processing, code analysis, and documentation understanding [9,29,42]. For example, GPT-3 [7] was trained on over 1 TB of filtered internet data, including open-source repositories (e.g., GitHub, Jira) and developer discussions from forums like Stack Overflow. This extensive training allows LLMs to consolidate knowledge from multiple sources into a single model.

Given their accessibility and effectiveness, LLMs offer software engineers a promising tool to efficiently retrieve AK. For instance, engineers can directly query LLMs for AK about open-source systems, reducing the need for manual integration and improving the overall understanding of software architectures. However, the number of LLMs, their vendors, and sizes have increased significantly in the last year, showing many options for software engineers to select from. Furthermore, we know little about the differences between the different LLMs, especially regarding the accuracy of their answers and similarities among their responses to questions about the AK of existing software systems. In our prior state-of-the-art work [37], we evaluated only on GPT-3.5, without considering more recent LLMs such as DeepSeek-R1, LLaMA 3.3, or GPT-4o. Therefore, in this paper, we **aim** to *compare different LLMs regarding the accuracy and similarity of their responses to questions about the architectural knowledge of an open-source system embedded in their models.*

To achieve our goal, we selected seven prominent LLMs with different sizes and vendors. We then evaluated the accuracy of the selected LLMs using our existing dataset from previous work [36] consisting of 229 questions written by software engineers on the AK of the open-source system, Hadoop HDFS. Furthermore, we compared the responses of the different LLMs with each other to determine which LLMs provide similar responses. In summary, we have the following contributions:

1. *An empirical evaluation on the accuracy of different LLMs to answer questions about the AK of an open-source system embedded in their models.* We focus on assessing the precision and recall in responding to queries related to various *AK concepts*, including components and connectors, quality solu-

tions, and the rationale of decisions. Our experiments compare various LLMs in terms of their accuracy in addressing questions about the AK concepts embedded within software systems.
2. *An empirical evaluation regarding the similarities of responses among the different LLMs.* Our work identifies which LLMs provide similar or distinct responses to architectural questions. Additionally, we determine which LLMs share correct or false answers.

With our contributions, we aim to determine if LLMs have differences regarding their accuracy and responses and if these differences are related to the size or vendor of the LLMs. This knowledge can support practitioners and researchers in deciding on suitable LLMs to answer questions on the AK of an existing system.

The remainder of the paper is structured as follows: Sect. 2 details our study design, while Sect. 3 and Sect. 4 present our findings. In Sect. 5, we discuss our results, and we discuss threats to validity in Sect. 6. In Sect. 7, we tackle related work. Finally, Sect. 8 concludes the paper.

2 Study Design

To achieve our aim (see Sect. 1), we ask the following research questions (RQs).
(***RQ1***) *How accurate are the answers provided by different LLMs regarding the architectural knowledge of an existing system embedded in their models?*

With the increasing number of LLMs and lack of information about their accuracy, researchers and practitioners struggle to select one or more LLMs to answer architectural questions. This RQ aims to determine accuracy variations among LLMs across different AK concepts. We seek to determine if certain LLMs excel in specific AK concepts, such as components, connectors, or quality solutions. Accordingly, researchers and practitioners could better decide on the most accurate LLM for their questions.
(***RQ2***) *How similar are the responses of the different LLMs to questions about the architectural knowledge of an existing system embedded in their models?*

Different LLMs may produce similar or divergent responses to the same architectural questions, potentially sharing correct or incorrect answers. This RQ explores whether LLMs provide similar responses and if they share the same correct or incorrect answers. Response similarity could benefit researchers and practitioners combining multiple LLMs to query AK of existing systems, supporting effective usage of various LLMs for architectural questions.

To answer the research questions (RQs), we followed three steps, depicted in Fig. 1 and detailed in the following sections. In Step 1, we selected seven prominent LLMs and submitted 229 architectural questions from an existing dataset, which served as our ground truth. In Step 2, we analyzed the LLMs' responses, extracting answers and comparing them to the ideal answers in the ground truth to evaluate the accuracy of LLMs for RQ1. In Step 3, we measured the similarity between LLMs' responses, including correct and incorrect answers, to address RQ2.

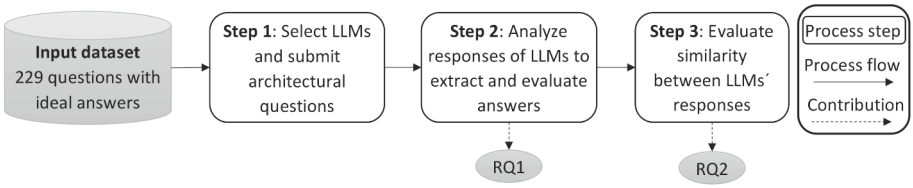

Fig. 1. Overview on the research process

Input Dataset

For the input dataset, we use the dataset from our previous work [36,37] with questions about the AK concepts and answers about the AK of Hadoop HDFS that we can use for our evaluation. Furthermore, the questions in the dataset are written independently by software engineers, along with their ground truth answers. The participants could freely articulate the questions and use direct questions (i.e., zero-shot) without providing examples or applying other prompting techniques, which aligns with how software engineers would commonly use LLMs. The dataset overall consists of 229 questions, distributed as follows: 80 questions about components and connectors, 81 questions about quality attribute solutions, and 68 questions about the rationale of design decisions. Sample questions include: (i) How do the architectural components in SPS (Storage Policy Satisfier) depend on each other? (ii) What are the security tactics used to improve security in the implementation of the SPS? (iii) For the SPS, what alternative solution was used prior or considered as an alternative solution?

Step 1: Select LLMs and Submit Architectural Questions

Table 1. The selected LLMs, their vendors, parameters, and training sizes. B stands for billions, and T stands for trillions tokens.

ID	Vendor	Model Name	Parameters	Training Size	Ref.
Q	Alibaba	Qwen2-Instruct	72B	7T	[43]
DS	DeepSeek	DeepSeek-R1-Distill-Llama	70B	14T	[12]
G4	OpenAI	gpt-4o	–	–	–
GM		gpt-4o-mini	–	–	–
LL	Meta	Llama-3.3-Instruct	70B	15T	[23]
TL		TinyLlama-Chat-v1.0	1.1B	3T	
M	Mistral	Mistral-Large-Instruct-2407	123B	–	[1]

To address our RQs, we selected various LLMs from different vendors with distinct sizes of parameters and training datasets. Table 1 lists each LLM's ID,

vendor, model name, parameters, and training size. The IDs of the LLMs will be referenced in Sect. 3 and Sect. 4. We chose these prominent models due to their widespread application in software engineering tasks [16]. While all LLMs use the decoder architecture, some are tailored for specific purposes or employ unique training techniques. For example, DeepSeek-R1-Distill undergoes a distillation process, fine-tuning the model from the larger DeepSeek-R1.

To retrieve responses for the architectural questions, we submitted the same questions from the input dataset to each LLM, ensuring a fair comparison of accuracy and answers. We used the Hugging Face transformers Python interface to run open LLMs on our hardware, providing a flexible and efficient environment for managing inference tasks across multiple models. All models were run in 4-bit quantization mode to accommodate hardware constraints, ensuring efficient inference without significant performance loss. Responses from all LLMs, except those from OpenAI, were retrieved using Hugging Face transformers. For OpenAI models, we used the OpenAI interface with the Langchain Python module. All experiments were conducted on the following hardware: 1xNVIDIA H100-80C GPU, 16xIntel(R) Xeon(R) Platinum 8462Y+ CPUs, and 78 GB RAM. This process returned responses from all LLMs in Table 1 to all 229 questions, totaling 1603 responses. All responses and scripts are available online [34].

Step 2: Analyze Responses of LLMs to Extract and Evaluate Answers

Table 2. Example for a question and responses of LLMs, answers are in *italic*

	Question: How has the Performance of QJournal within Hadoop HDFS been improved through implemented tactics ?
LL	...several tactics have been implemented: 1. ***Data Striping***: QJournal uses a striping approach to store data across multiple DataNodes... 2. ***Async Writes***: QJournal uses asynchronous writes... 3. ***Batching***: ... 4. ***Caching***:... 5. ***Data Compression***: ... 6. ***Parallelization***: QJournal uses parallelization techniques, such as multi-threading and parallel processing, to improve performance... 10. ***Monitoring and Tuning***:...
DS	<think> Okay, so I'm trying to understand how the performance of QJournal... has been improved through implemented tactics...</think>The performance of QJournal...has been enhanced through several strategic implementations...: 1. ***Data Writing Efficiency***: - ***Batching Writes***: Aggregating multiple log entries into a single write... - ***Efficient Serialization***: Utilizing formats like Avro or Protobuf... 2. ***File Management***: - ***Concatenation ...***:...
M	Here are some key strategies that have been implemented to enhance QJournal performance: 1. ***Efficient Data Replication***: - ***Pipeline Optimization***: ...- ***Parallel Replication***: ... 2. ***Improved Write Path***: - ***Batch Writes***:...- ***Asynchronous Writes***:... 3. ***Metadata Management***: - ***Efficient Metadata Storage***:...
TL	here are some tactics...: 1. *Implementing a distributed journaling mechanism:* qjournal uses a distributed journaling mechanism... 2. *Implementing a write-ahead log (wal)*: qjournal uses a write-ahead log (wal) to write data to the disk...

In this step, we analyze the responses of the LLMs to extract answers and compare them to the ideal answers in the ground truth. We note that each response involves multiple answers for the submitted question. Table 2 provides

examples for different responses of LLMs and the *answers* in each response. When comparing answers to the ground truth, we classify answers among:

- *True positive (TP)* generated answers that match the ground truth.
- *False positive (FP)* generated answers that do not match the ground truth.
- *False negative (FN)* answers from the ground truth that are not generated.

Given the large number of responses, manually analyzing all 1603 responses to extract answers and compare them to the ground truth was not feasible. We opted for semi-automatic analysis as the analysis is time-consuming, as also reported in our previous work [37]. Initially, we aimed for a fully automated analysis but encountered obstacles in accurately extracting and classifying answers from the LLMs' responses. Specifically, we followed three steps:

1. **Determine formats of responses and extract answers**: One obstacle in extracting answers from LLMs' responses is their diverse formats. Table 2 illustrates these differences. For instance, LLaMA provides a numerical list using the format ***Answer***, while DeepSeek and Mistral use hierarchical lists with numbers and dashes. To address this, we identified common formats in each LLM's responses and developed a script (available online [34]) to extract answers.
2. **Compare answers to the ground truth**: To evaluate the extracted answers, we compared them with the ground truth. One challenge was the varied terminology used by LLMs for the same answer. For instance, concurrency appears as *parallelization* in LLaMA and *parallel replication* in Mistral (Table 2). Exact string matching would miss these variations. To overcome this, we applied fuzzy string matching [10] using the difflib Python library[1]. This method allowed us to classify answers as TP, FP, or FN. Fuzzy matching requires a threshold to determine similarity. We experimented with thresholds between 0.9 and 0.5 on a random sample of 400 responses, finding 0.65 to be the most precise with a precision of 0.92. Using fuzzy matching, we compared all extracted answers to the ground truth and stored them for manual verification.
3. **Manual verification and corrections**: Certain answers contained synonymous that fuzzy matching could not identify correctly, such as *parallel processing* as a synonym for *parallelization*. To ensure accurate classification, two authors manually reviewed all FP answers to detect and correct misclassified answers. This step required significantly less effort than manually checking full responses, as we focused on specific answers. All answers are shared online [34].
4. **Calculate precision and recall**: Using TP, FP, and FN counts, we calculated $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$ for each LLM response to answer RQ1. To identify significant differences in $Precision$ and $Recall$ among LLMs, we conducted significance tests on their distributions across all questions and pairs of LLMs. Specifically, we used the t-test for normalized data and the Mann-Whitney-U-Test for skewed data. The results are presented in Sect. 3. Also, all values of precision, recall, and significance test results are online [34].

[1] https://docs.python.org/3/library/difflib.html.

Step 3: Evaluate Similarity Between LLMs' Responses
To answer RQ2, we compared responses to the same questions across all pairs of LLMs using the following two approaches:

• **Compare full responses of LLMs using BERTScore**: In this approach, we measure the similarity between full responses from different LLMs to the same questions using BERTScore [44]. We chose BERTScore over other methods (e.g., BLEU) because it captures semantic similarity. Using the BERTScore library[2], we evaluated the similarity among responses of all combinations of LLM pairs (21 pairs) from the 7 LLMs in Table 1. The results include precision, recall, and F-score values, indicating the similarity between two responses. The average F-score values are presented in Table 4, and further values are online [34].

• **Compare answers using fuzzy matching and manual analysis**: BERTScore does not differentiate between true and false answers in a response. Therefore, in this approach, we separately compare the identified TP and FP answers (from Step 2) to determine which LLMs provide similar correct or incorrect answers for the same questions. We applied fuzzy matching to the TP and FP answers separately, using the steps described in Step 2 to determine the fuzzy threshold (0.6) with the best precision. Evaluating the results, we found that matching similar TP answers had a high precision of 0.95, while matching FP answers had a low precision of 0.5. This low precision is due to the hallucination behavior of LLMs, which produce FP answers with varied terms that fuzzy matching cannot capture. To ensure accurate similarity between FP answers, two authors manually verified all FP answers among the different LLMs. After determining similar answers, we calculated the *ratio of similar answers* in a response with each of the other LLMs. The average ratios are presented in Sect. 4, and ratios for all questions are available online [34].

To determine significant similarities, we executed the same significance tests as RQ1. Tests were executed separately among the BERTScores and ratios. Detailed results of the significance tests are available online [34].

3 RQ1: Accuracy of LLMs

Figure 2 shows the precision and recall distributions for each LLM and AK concept. Generally, all LLMs achieve low precision for AK concepts, with quality attribute solutions having the lowest precision. Conversely, most LLMs exhibit higher recall for all AK concepts. These findings align with our previous work [37] on GPT-3.5, indicating that LLMs tend to provide all possible answers for architectural questions, resulting in high false positives. Therefore, researchers and practitioners must filter out false positives, especially for quality attribute solutions like patterns or tactics.

Regarding the differences in precision and recall among LLMs, Table 3 presents the results of our significance tests. We observe the following:

[2] https://pypi.org/project/bert-score/.

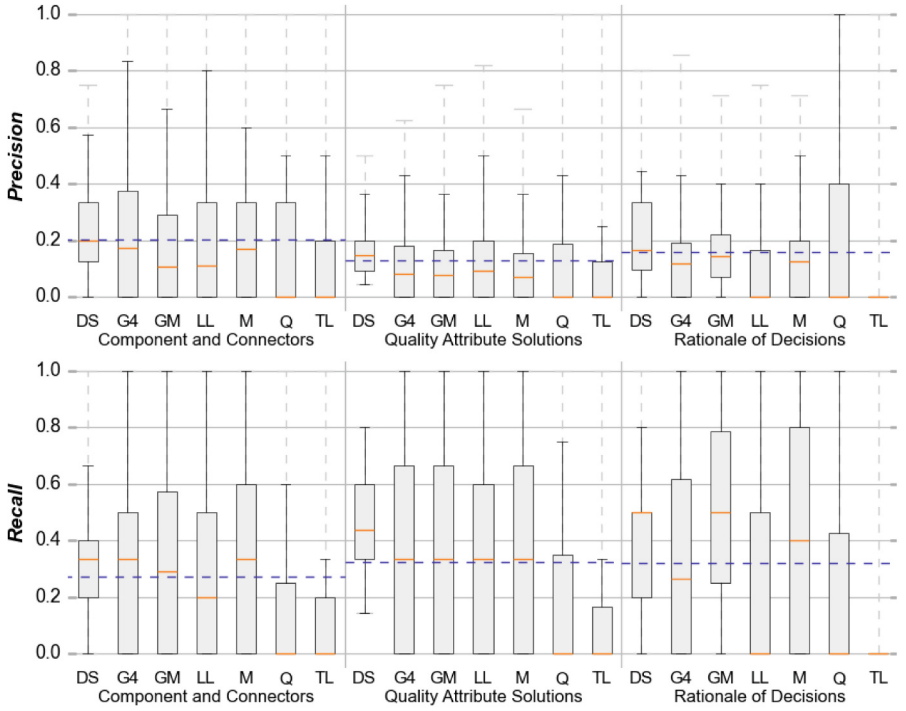

Fig. 2. Precision and recall of different LLMs and AK concepts

- For *components and connectors*, most LLMs show similar precision and recall but with different distributions. TinyLLaMA has significantly lower precision and recall, and Qwen has significantly lower recall than other LLMs. Therefore, for questions on components and connectors, practitioners and researchers can choose among the LLMs in Table 1, except for TinyLLaMA and Qwen.
- For *quality attribute solutions*, most LLMs show very low precision, with DeepSeek having significantly better precision than all other LLMs except Large LLaMA. TinyLLaMA has the significantly worst precision. Most LLMs show high recall, except for TinyLLaMA and Qwen. DeepSeek has the best recall, but it is only significantly better than GPT-4o and GPT-4 mini. Thus, for questions on quality attribute solutions, DeepSeek is recommended.
- For *rationale of decisions*, LLMs show varying precision and recall. DeepSeek and GPT-mini achieve the best precision, significantly outperforming all other LLMs. For recall, GPT-mini leads significantly better than others, except DeepSeek and Mistral. TinyLLaMA and Qwen have the worst precision and recall. Thus, for questions on the rationale of design decisions, practitioners and researchers should use DeepSeek and GPT-mini for their superior precision and recall.

Table 3. Results of significance tests for precision and recall, and among all pairs of LLMs. Given the set of LLMs in Table 1, we specify significance. We use $X >> Y$ to indicate that the LLM X is significantly better than the LLM Y for the metric, and $X << Y$ to show that X is significantly worse than Y

AK concept	Metric	Significant results
Components and connectors	Precision	$TL << x, \quad \forall x \in LLMs \setminus \{Q\}, \quad G4 >> Q$
	Recall	$x >> TL, Q \quad \forall x \in LLMs \setminus \{TL, Q\}$
Quality attribute solutions	Precision	$DS >> x \quad \forall x \in LLMs \setminus \{LL\}, \quad TL << x \quad \forall x \in LLMs \setminus \{Q\}, \quad LL >> Q$
	Recall	$TL, Q << x \quad \forall x \in LLMs, \quad DS >> G4, GM$
Rationale of design decisions	Precision	$DS >> x \quad \forall x \in LLMs \setminus \{GM\} \quad GM >> LL, TL << x \quad \forall x \in LLMs$
	Recall	$TL << x \quad \forall x \in LLMs, \quad GM >> Q, LL, G4, \; DS, M >> Q, LL$

RQ1 key takeaways:

- LLMs achieve *low precision but higher recall* for architectural questions.
- *DeepSeek* achieves better results than larger models like GPT-4 and Mistral for questions on quality solutions and rationale of decisions.
- *TinyLLaMA and Qwen* achieve the worst precision and recall overall.

4 RQ2: Similarities Among LLMs

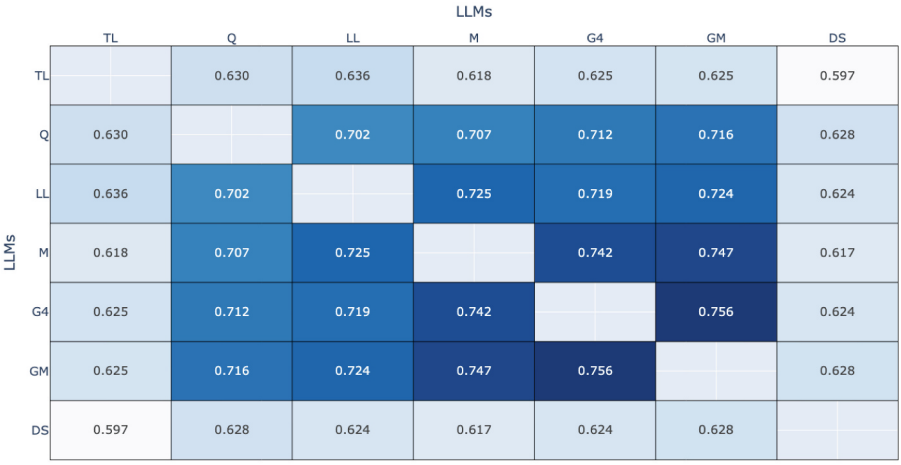

Fig. 3. Average BERTScore (F_{BERT}) for the LLMs' response similarities

Figure 3 shows a heatmap with the average F-score from the BERTScore (F_{BERT}) among the used LLMs (see Fig. 1) for all AK concepts combined. There

were no significant differences in similarities between different AK concepts. The LLMs exhibit high and close similarities in their responses, ranging from 0.597 to 0.756. The highest similarities are among GPT-4o, GPT-4o mini, and Mistral, followed by LLaMA and Qwen. In contrast, DeepSeek and TinyLLaMA have the lowest similarities with other LLMs. Therefore, from Fig. 3, when asking multiple LLMs, more than half of a response of an LLM will be similar to other LLMs.

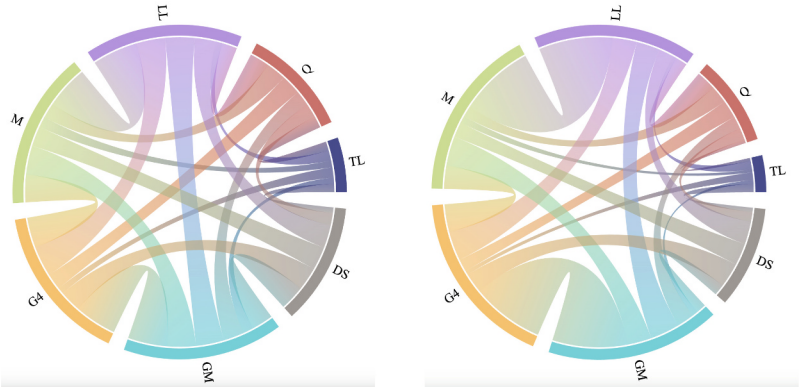

Fig. 4. Similarity ratios of true positive and false positive answers

The BERTScore in Fig. 3 does not differentiate between true and false positive answers. Figure 4 shows a chord diagram illustrating the similarity ratios in true and false positive answers among the LLMs in Fig. 1 for all AK concepts combined. Each segment represents an LLM, and the connections (chords) indicate their similarities with other LLMs. The cords' thickness represents the average similarity ratio of an LLM's answers to another LLM in the chord. There were no significant differences in similarities for the different AK concepts (e.g., components and connectors vs. quality solutions). Thus, we combined AK concepts in Fig. 4, with further figures and exact ratios available in our supplementary material [34].

We observe that all LLMs share both true and false answers. GPT-4o, GPT-4o mini, LLaMA, and Mistral have the highest similarities, while DeepSeek, Qwen, and TinyLLaMA have the lowest. For example, if engineers ask LLMs about system components, LLaMA and Mistral will likely share components, whereas DeepSeek, Qwen, and TinyLLaMA will provide more different ones.

Comparing the similarities among true and false positive answers in Fig. 4a and Fig. 4b, we see that false positive answers have higher similarities than true positive answers, especially between GPT-4o and GPT-4o mini and between Mistral and LLaMA. DeepSeek, Qwen, and TinyLLaMA show even smaller similarities for false positives than true positives. These observations may be due to similarities in the training datasets or architectures of LLMs, causing their models to produce similar or different hallucinations of false positives. For example,

if software engineers ask GPT-4o and GPT-4o mini about security tactics, they might receive responses with similar false security tactics.

RQ2 key takeaways:

- *All LLMs share at least half of their responses with other LLMs.* This includes true and false positive answers to architectural questions.
- *GPT-4o, GPT-4o mini, LLaMA, and Mistral* have the biggest similarity among their responses and answers, while *DeepSeek, Qwen, and TinyLLaMA* have the smallest similarities among their responses and answers.

5 Discussion

Implications for Researchers
The results of RQ1 confirm our previous findings [37] that zero-shot LLMs achieve low precision and high recall, providing many false positives. This suggests that standard LLMs, with their current architecture and training datasets, are not yet suitable for architectural questions without further customization. Researchers should explore approaches to customize LLMs, such as using advanced prompting techniques (e.g., Chain-of-Thought) or fine-tuning.

RQ1 results also show that the parameters size, training dataset and methodology of LLMs significantly impact their precision and recall. TinyLLaMA and Qwen, the smallest model and smallest training datasets, have the worst precision and recall. Very large models like GPT-4o and Mistral show saturation, with no significant improvement over mid-sized models like GPT-4 mini and LLaMA. Interestingly, DeepSeek, a mid-sized LLM with a different training methodology through distillation, outperforms larger models, indicating that methodologies of training play a crucial role in achieving better precision and recall. Thus, researchers should focus on improving the methodologies of LLM training and architectures over expanding parameter sizes.

RQ2 results can guide researchers in combining LLMs to answer architectural questions more accurately. For instance, a voting mechanism could be used to find the correct answers. However, since some LLMs share false positive answers, researchers should select LLMs with the least similarity in false positives for this approach. DeepSeek and Mistral, which have the least similarity ratios of false positives, could be good candidates for a voting mechanism.

Implications for Practitioners
The results of RQ1 indicate that LLMs should be used cautiously for architectural questions due to the high number of false positives. Practitioners can use LLMs to create initial architectural documentation and provide preliminary answers, but these answers should be filtered and verified for accuracy.

For components and connectors, different LLMs (except TinyLLaMA and Qwen) have similar performance, giving practitioners flexibility in their choice.

For other architectural questions, DeepSeek or GPT-4 mini are recommended for their superior performance for quality solutions and rationale of decisions.

The results of RQ2 guide practitioners on which combinations of LLMs to use for architectural questions. To obtain a higher variety of answers and information, practitioners should ask LLMs with less similar responses. For example, DeepSeek and GPT-4o have lower similarity and high accuracy, making them good choices to cover the most correct and varied answers.

6 Threats to Validity

External Validity
One threat to external validity is that our evaluation of LLMs was based on the AK of a single software system, Hadoop HDFS, as has been provided in the ground truth [37]. This may limit the generalizability of our findings to other open-source systems, which might have more or less resources about their software architecture in the training datasets of LLMs. However, researchers frequently analyzed Hadoop HDFS for its software architecture [25,32], due to its extensive resources on AK. Therefore, we believe our results could apply to other open-source systems with similar AK resources.

Another threat to external validity is that we evaluated seven specific LLMs. Our findings may not be generalizable to other models of varying sizes and datasets. Nevertheless, our results can serve as a baseline and hypothesis regarding the accuracy and similarities of LLMs to answer architectural questions related to existing systems.

Construct Validity
One threat to construct validity is due to the applied textual comparison technique (see Sect. 2) to compare the responses of the LLMs to the ground truth, and among responses, which might involve mistakes. However, we have utilized fuzzy string matching to cover the most variations and adjusted the threshold of fuzzy matching to achieve the best accuracy. Furthermore, we followed a semi-automated approach with manual verification.

Reliability
One threat to reliability arises from the variability of LLMs. These models typically incorporate a random element in their output to enhance text diversity. For instance, when the same question is posed multiple times, an LLM might generate slightly different responses each time due to this stochastic nature. This randomness can make it challenging to replicate results. However, we share the responses of LLMs and scripts [34] to facilitate replication of the research process.

Another threat to reliability is using a semi-automated approach to analyze responses and extract answers. The manual analysis might involve bias. However, two authors analyzed and discussed the data. Also, we limited the manual analysis to specific answers (e.g., only false positives), where each answer has few terms. The small size of manual analysis reduced the chance for disagreements.

7 Related Work

This section discusses related work on AK and other related work that utilizes LLMs for software architecture.

7.1 Architectural Knowledge

Early work (e.g., [27,38]) explored AK concepts and proposed manual documentation tools (e.g., [21]). More recent studies have explored AK within software repositories, such as issue trackers [4,28,32] and mailing lists [33], as well as on the Web through forums [6], technical documentation [22], and blogs [35]. Additionally, researchers investigated design decisions for specific domains, such as quantum [2] and machine learning systems [41]. However, these studies do not utilize or evaluate LLMs, which sets them apart from our research aim.

Researchers applied machine learning techniques to classify design decisions from issue trackers [4] and mailing lists [17]. However, these approaches do not incorporate or evaluate LLMs, setting them apart from this study.

7.2 Language Models for Software Architecture

Software engineers started to use LLMs for different tasks [24,40], and software architecture researchers also started to evaluate LLMs.

Some work utilizes BERT for different architectural tasks, such as classifying design decisions in documentation [26] and mailing lists [33]. However, these works do not use or evaluate advanced LLMs like GPT and LLaMA, which offer different capabilities (e.g., prompting) than BERT.

Dhar et al. [13,14] use GPT and T5 to recommend design decisions. The authors evaluate different prompting techniques and fine-tuning. Similarly, Díaz-Pace et al. [15] propose approaches to recommend and document design decisions, in which the authors use GPT combined with retrieval-augmented generation. However, the goal of these works is different, as they aim to propose and document new design decisions, whereas our study evaluate seven LLMs to answer questions about the AK of existing systems.

Rukmono et al. [31] propose an approach to perform conformance checks between code and architectural components. The proposed approach use prompting techniques in LLMs, and specifically chain-of-thoughts. However, this work is different than ours, because it focus on conformance checks, whereas our study explore AK in existing systems without having a design specification.

Fuchß et al. [18] apply LLMs for traceability between architecture documentation and code. They also explore in another work [19] the usage of LLMs to generate simple architecture models that can be used as intermediate artifacts in transitive traceability. However, these works are different as they focus on traceability rather AK.

In our previous work [36,37], we evaluated the accuracy, quality and trustworthiness of GPT-3.5 to answer architectural questions on Hadoop HDFS. However, in this paper, we evaluate seven different LLMs and analyze the similarity of their generated answers.

8 Conclusion

In this study, we compared the accuracy and similarity of various LLMs in addressing architectural questions. Our findings show that all LLMs achieve higher recall than precision. Additionally, we observed significant differences in accuracy among the LLMs, with certain models performing better on specific architectural knowledge concepts. Despite these differences, LLMs show high similarity in their responses, including both correct and incorrect answers.

Future work involves extending this evaluation to other software systems. Furthermore, we plan to develop new approaches to improve the accuracy of LLMs for answering questions about the architectural knowledge of an existing system. Our planned approaches will use recent prompting techniques, fine tuning and combining the results of LLMs to improve accuracy.

Acknowledgements. This work was funded by the Topic Engineering Secure Systems of the Helmholtz Association (HGF) and supported by KASTEL Security Research Labs, Karlsruhe.

Data Availibility. All data and artifacts are available online [34].

References

1. Large Enough — Mistral AI. https://mistral.ai/news/mistral-large-2407
2. Aktar, M.S., Liang, P., et al.: Architecture decisions in quantum software systems: an empirical study on stack exchange and GitHub. Inf. Softw. Technol. **177**, 107587 (2025). https://doi.org/10.1016/j.infsof.2024.107587
3. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 3rd edn. Addison-Wesley (2012)
4. Bhat, M., Shumaiev, K., Biesdorf, A., Hohenstein, U., Matthes, F.: Automatic extraction of design decisions from issue management systems: a machine learning based approach. In: Lopes, A., de Lemos, R. (eds.) ECSA 2017. LNCS, vol. 10475, pp. 138–154. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65831-5_10
5. Bhat, M., et al.: An expert recommendation system for design decision making: who should be involved in making a design decision? IEEE 15th ICSA, pp. 85–94 (2018). https://doi.org/10.1109/ICSA.2018.00018
6. Bi, T., Liang, P., Tang, A., Xia, X.: Mining architecture tactics and quality attributes knowledge in Stack Overflow. J. Syst. Softw. (2021). https://doi.org/10.1016/j.jss.2021.111005
7. Brown, T., Mann, B., Ryder, N., et al.: Language models are few-shot learners. In: Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020). https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf
8. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, 1 edn. Wiley (1996)

9. Chen, L., et al.: A survey on evaluating large language models in code generation tasks (2024). arXiv:2408.16498
10. Crochemore, M., Rytter, W.: Jewels of stringology: text algorithms. World Sci. (2002). https://doi.org/10.1142/4838
11. de Dieu, M.J., Liang, P., et al.: Characterizing architecture related posts and their usefulness in stack overflow. J. Syst. Softw. **198**, 111608 (2023). https://doi.org/10.1016/j.jss.2023.111608
12. DeepSeek-AI, Guo, D., Yang, D., Zhang, H., et al.: DeepSeek-R1: incentivizing reasoning capability in LLMs via reinforcement learning (2025). https://doi.org/10.48550/arXiv.2501.12948
13. Dhar, R., Vaidhyanathan, K., Varma, V.: Can LLMs generate architectural design decisions? - an exploratory empirical study. In: IEEE 21st ICSA, pp. 79–89. IEEE (2024). https://doi.org/10.1109/ICSA59870.2024.00016
14. Dhar, R., Vaidhyanathan, K., Varma, V.: Leveraging generative AI for architecture knowledge management. In: IEEE 21st ICSA Companion, pp. 163–166 (2024). https://doi.org/10.1109/ICSA-C63560.2024.00034
15. Díaz-Pace, J.A., Tommasel, A., Capilla, R.: Helping novice architects to make quality design decisions using an LLM-based assistant. In: Galster, M., Scandurra, P., Mikkonen, T., Oliveira Antonino, P., Nakagawa, E.Y., Navarro, E. (eds.) ECSA 2024. LNCS, vol. 14889, pp. 324–332. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-70797-1_21
16. Fan, A., et al.: Large language models for software engineering: survey and open problems. In: IEEE/ACM ICSE: Future of Software Engineering, pp. 31–53 (2023). https://doi.org/10.1109/ICSE-FoSE59343.2023.00008
17. Fu, L., Liang, P., Li, X., Yang, C.: A machine learning based ensemble method for automatic multiclass classification of decisions. In: ACM EASE, pp. 40–49 (2021). https://doi.org/10.1145/3463274.3463325
18. Fuchß, D., et al.: LiSSA: toward generic traceability link recovery through retrieval-augmented generation. In: ICSE (2025). https://doi.org/10.1109/ICSE55347.2025.00186
19. Fuchß, D., Liu, H., Hey, T., Keim, J., Koziolek, A.: Enabling architecture traceability by LLM-based architecture component name extraction. In: 22nd IEEE ICSA (2025). https://doi.org/10.5445/IR/1000179830
20. Garcia, J., Ivkovic, I., Medvidovic, N.: A comparative analysis of software architecture recovery techniques. In: IEEE/ACM ASE, pp. 486–496 (2013). https://doi.org/10.1109/ASE.2013.6693106
21. Gerdes, S., Soliman, M., Riebisch, M.: Decision buddy: tool support for constraint-based design decisions during system evolution. In: International WS on FoSADA, pp. 13–18. ACM, New York (2015). https://doi.org/10.1145/2751491.2751495
22. Gorton, I., Xu, R., Yang, Y., Liu, H., Zheng, G.: Experiments in curation: towards machine-assisted construction of software architecture knowledge bases. In: IEEE/IFIP ICSA 2017, pp. 79–88 (2017)
23. Grattafiori, A., Dubey, A., Jauhri, A., et al.: The Llama 3 Herd of Models (2024). https://doi.org/10.48550/arXiv.2407.21783
24. Hou, X., et al.: Large language models for software engineering: a systematic literature review. TOSEM (2024). https://doi.org/10.1145/3695988
25. Kazman, R., Goldenson, D., Monarch, I., Nichols, W., Valetto, G.: Evaluating the effects of architectural documentation: a case study of a large scale open source project. IEEE Trans. Software Eng. **42**(3), 220–260 (2016). https://doi.org/10.1109/TSE.2015.2465387

26. Keim, J., Hey, T., Sauer, B., Koziolek, A.: A taxonomy for design decisions in software architecture documentation. In: Batista, T., Bureš, T., Raibulet, C., Muccini, H. (eds.) ECSA 2022. LNCS, vol. 13928, pp. 439–454. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-36889-9_29
27. Kruchten, P., Lago, P., van Vliet, H.: Building up and reasoning about architectural knowledge. In: Hofmeister, C., Crnkovic, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 43–58. Springer, Heidelberg (2006). https://doi.org/10.1007/11921998_8
28. Maarleveld, J., Dekker, A., Druyts, S., Soliman, M.: Maestro: a deep learning based tool to find and explore architectural design decisions in issue tracking systems. In: ECSA Tools Tracks (2023)
29. Naveed, H., et al.: A comprehensive overview of large language models (2024). arXiv:2307.06435
30. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding with unsupervised learning (2018). https://web.archive.org/web/20230318210736/https://openai.com/research/language-unsupervised
31. Rukmono, S.A., Ochoa, L., Chaudron, M.: Deductive software architecture recovery via chain-of-thought prompting. In: ICSE-NIER (2024). https://doi.org/10.1145/3639476.3639776
32. Soliman, M., Galster, M., Avgeriou, P.: An exploratory study on architectural knowledge in issue tracking systems. ECSA (2021). https://doi.org/10.1007/978-3-030-86044-8_8
33. Soliman, M.: Exploring architectural design decisions in mailing lists and their traceability to issue trackers. In: Galster, M., Scandurra, P., Mikkonen, T., Oliveira Antonino, P., Nakagawa, E.Y., Navarro, E. (eds.) ECSA 2024. LNCS, vol. 14889, pp. 307–323. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-70797-1_20
34. Soliman, M., Ashraf, E., Abdelsalam, K.M.K., Keim, J., Venkatesh, A.P.S.: Supplementary material for 'LLMs for Software Architecture Knowledge: A Comparative Analysis Among Seven LLMs'. https://doi.org/10.5281/zenodo.15584019
35. Soliman, M., Gericke, K., Avgeriou, P.: Where and what do software architects blog?: An exploratory study on architectural knowledge in blogs, and their relevance to design steps. In: IEEE ICSA, pp. 129–140 (2023). https://doi.org/10.1109/ICSA56044.2023.00020
36. Soliman, M., Keim, J.: Supplementary material for 'Do Large Language Models Contain Software Architectural Knowledge? An Exploratory Case Study with GPT'. https://doi.org/10.5281/zenodo.14512761
37. Soliman, M., Keim, J.: Do large language models contain software architectural knowledge? An exploratory case study with GPT. In: IEEE ICSA. IEEE (2025). https://doi.org/10.5445/IR/1000178724
38. Tang, A., Jin, Y., Han, J.: A rationale-based architecture model for design traceability and reasoning. JSS **80**(6), 918–934 (2007). https://doi.org/10.1016/j.jss.2006.08.040
39. Touvron, H., et al.: Llama: open and efficient foundation language models (2023). arXiv:2302.13971
40. Venkatesh, A.P.S., Sabu, S., Mir, A.M., Reis, S., Bodden, E.: The emergence of large language models in static analysis: a first look through micro-benchmarks. In: FORGE (2024). https://doi.org/10.1145/3650105.3652288
41. Warnett, S.J., Zdun, U.: Bridging the gap between MLOps and RLOps: an industry 4.0 case study on architectural design decisions in practice. In: IEEE ICSA (2025). http://eprints.cs.univie.ac.at/8343/

42. Xuanfan, N., Piji, L.: A systematic evaluation of large language models for natural language generation tasks. In: 22nd Chin. Nat. Conf. on Comp. Linguistics, Harbin, China, pp. 40–56 (2023). https://aclanthology.org/2023.ccl-2.4
43. Yang, A., Yang, B., Hui, B., et al.: Qwen2 Technical Report (2024). https://doi.org/10.48550/arXiv.2407.10671
44. Zhang, T., Kishore, V., Wu, F., Weinberger, K.Q., Artzi, Y.: BERTScore: evaluating text generation with BERT (2020). arXiv:1904.09675