Do Large Language Models Contain Software Architectural Knowledge?

An Exploratory Case Study with GPT

Mohamed Soliman
Heinz Nixdorf Institut, Universität Paderborn, Germany
mohamed.soliman@uni-paderborn.de

Jan Keim
Karlsruhe Institute of Technology (KIT), Germany
jan.keim@kit.edu

***Abstract*—Architectural knowledge (AK) of existing systems is essential for software engineers to make design decisions. Recently, Large Language Models (LLMs) trained on large-scale datasets, including software repositories, have shown promise in embedding knowledge and answering questions. However, LLMs have not been evaluated for their abilities to answer questions about AK, leaving doubts about their accuracy. This paper assesses GPT, a leading LLM, by evaluating its responses' accuracy, quality, and trustworthiness on the AK of the large-scale open-source system HDFS. We conducted an exploratory case study with 14 software engineers who posed questions to GPT and compared its responses to a predefined ground truth. The engineers rated GPT's answers with moderate quality and trustworthiness. Our findings on GPT´s accuracy indicates moderate recall but lower precision, especially in identifying quality attribute solutions and design rationales. These results suggest that while GPT and similar models can provide initial insights into AK, expert validation remains necessary for reliability. This study underscores LLMs' potential and limitations to discover software AK.**

***Index Terms*—Architecture knowledge, Architecture design decisions, Large Language Models, GPT**

I. Introduction

Software engineers frequently ask questions about the *architectural knowledge (AK)* [1] of existing software systems. The questions on AK involve various *AK concepts*, like *components and connectors* (e.g., microservices or messaging) [2], implemented *quality solutions* (e.g., patterns [3], tactics [2]) for achieving attributes like performance or security, and the *rationale of design decisions* [4]. Each concept supports engineers differently: AK on *components and connectors* helps in adding features or fixing bugs, AK on *quality solutions* aids in analyzing and improving quality attributes, and *design rationale* [4] supports system evolution and decision-making.

While AK is important for software engineers, determining certain AK concepts for specific parts of the system is complex. One reason behind this complexity is that the AK of an existing system is distributed among different sources. One source is software repositories, such as version control systems (e.g., Git) [5] and issue trackers [6], [7]. Another source is developer communities on the web, such as forums (e.g., Stack Overflow [8]) and blog articles [9]. Each source contains part of the AK, and each source is required to fully and effectively understand the AK of existing software systems.

In the current state of the art, software architectural researchers propose methods to extract AK from open-source systems (e.g., [10], [11]), focusing on specific sources and AK concepts. For instance, some approaches analyze source code to identify components and connectors [5] and quality solutions (e.g., tactics [10]). Others analyze issue trackers [6], [7], mailing lists [11], and blogs [9] to extract design rationale. However, recent research shows that design decisions span multiple sources, such as issue trackers, mailing lists [11], and version control systems (e.g., Git). Thus, even when using current approaches, engineers must manually integrate AK from different sources to fully understand the architecture of a software system. Manual AK integration is challenging, time-consuming, and remains unaddressed by current approaches.

To address the challenges of manual AK extraction and integration, software architectural researchers (e.g., [12], [13]) currently explore the recent technology Large Language Models (LLMs), such as GPT [14] and LlaMa [15], which demonstrate strong capabilities in natural language processing, including text analysis, code generation, and documentation [16]–[18]. Their effectiveness stems from advanced model architectures and extensive training on large, diverse datasets. For example, GPT-3 [19] and its successor GPT-3.5, with 175 billion parameters, were trained on over 1 TB of filtered internet data, including open-source repositories (e.g., GitHub, Jira) and posts from forums like Stack Overflow [19]. This extensive training enables LLMs to integrate knowledge from various sources into a unified model.

Given the promising abilities and availability of LLMs, software engineers can easily utilize LLMs to ask questions about the AK of existing software systems, e.g., using chatGPT to ask questions about the AK of open-source systems. However, we know little about the abilities of LLMs to answer questions about the AK of existing software systems that are embedded in their models and part of their datasets. This raises doubts about the accuracy, quality, and trustworthiness of LLMs' generated answers. Therefore, in this paper, we aim to *evaluate the accuracy, quality, and trustworthiness of GPT's answers to questions about the architectural knowledge of an open source system embedded in its model*. We have decided on GPT because it is the most popular and well-known LLM, making our results more relevant for practitioners.

To achieve our aim, we conducted a case study with 14 software engineers, who asked GPT questions about the AK concepts of the open-source system, Hadoop HDFS. We justify our choice of HDFS in Section II. To evaluate the accuracy of GPT, we analyzed the answers of GPT and compared them to the ground truth of HDFS. Furthermore, the software engineers in our study evaluated the quality and trustworthiness of GPT's answers. In summary, we achieved the following contributions:

1) *An empirical evaluation on the accuracy of GPT to answer questions about the AK of an open-source system, embedded in its model*. We specifically evaluated the accuracy of GPT-3.5[1] to answer questions about different *AK concepts*: components and connectors, quality solutions, and rationale of design decisions. Furthermore, we analyzed the confusion of GPT-3.5 when providing wrong answers. Our evaluation sets the first baseline on the accuracy of LLMs to answer questions about the AK concepts for software systems embedded in their model.
2) *An empirical evaluation on the quality and trustworthiness of GPT's answers according to software engineers*. Our contribution also involves analyzing the reasons for poorly evaluated answers to determine the quality and trustworthiness issues identified by practitioners.

With our contributions, we aim to determine how far LLMs could actually be used by practitioners and researchers to answer questions on the AK of an existing system. The rest of the paper is organized as follows: Section II explains our study design, Section III and Section IV present our results. Section V, Section VI, Section VII discuss our results, threats to validity, and related work. Finally, Section VIII concludes the paper.

II. Study Design

A. Research questions

To achieve our aim (see Section I), we ask two research questions (RQs) that we explain in the following.

(RQ1) *How accurate is GPT to answer questions about the architectural knowledge of an open-source system that is embedded in its model?*

The accuracy of GPT, as one popular LLM, will provide a baseline evaluation for the accuracy of LLMs in answering questions about the AK of existing systems. By answering RQ1, we can determine the strengths and weaknesses of LLMs to answer questions about different AK concepts. On the one hand, researchers can develop new approaches that overcome the weaknesses of LLMs. On the other hand, practitioners will be aware of the strengths and weaknesses of LLMs, which will guide them in adopting this technology in their organizations.

(RQ2) *How do software engineers evaluate the quality and trustworthiness of GPT's responses regarding the architectural knowledge of an open-source system that is embedded in its model?*

In answering RQ2, we determine important practical aspects regarding the quality of answers produced by LLMs as question-answering systems and the extent to which software engineers can trust these systems. These two aspects (quality and trustworthiness) are important in determining whether using LLMs is sufficient for software engineers' practical use.

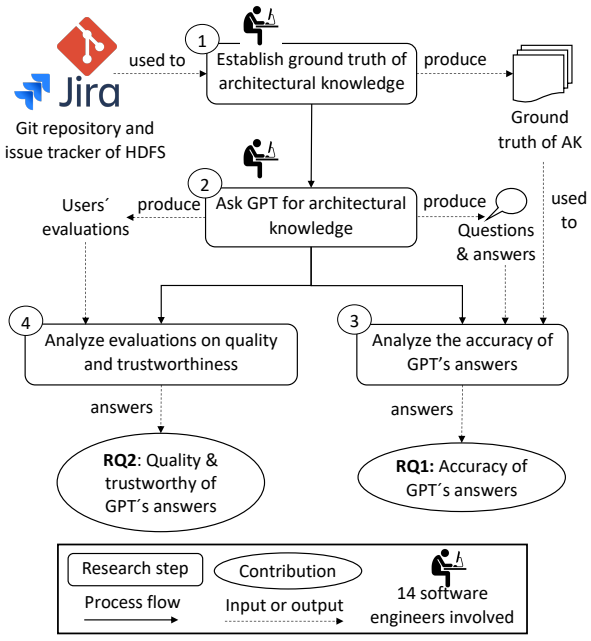

Fig. 1: Research process

To answer the aforementioned RQs, we conducted an exploratory case study with 14 software developers. All 14 *participants of our study* have at least a BSc in computer science or a related field, and have industrial experiences that range between 1 to 5 years in software development, with an average of 3 years of industrial experience in software development, as well as an average of 1 year of experience with software architecture. We provide all details on the participants' backgrounds online [20].

The participants of our study utilized GPT-3.5 to ask questions about the AK of Apache HDFS[2]. We decided on HDFS due to the availability of its architectural documents [21] and sufficient architectural issues in its issue trackers [7], which facilitates determining the ground truth of its AK. Furthermore, HDFS is a large-scale system with a big community and many online resources that are presumably part of GPT's training data. To validate that HDFS is part of GPT's training dataset, we asked GPT-3.5 questions about the sources of knowledge on HDFS, such as the locations of source code, names of classes, and issues in issue trackers. The answers of GPT-3.5 show that GPT's training dataset involves many resources from HDFS, including source code, issue trackers, and documentation. Accordingly, we decided on HDFS as a suitable system to achieve the aim of this study.

For the design of our case study, we follow the guidelines by Runeson and Höst [22]. Our case study is a single-embedded case study. Our case is HDFS, and the units of analysis are

[1]At the time of the study, GPT-3.5 was the most recent stable version.

[2]https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, last accessed 15-11-2024.

software engineers who aim to extract AK from the HDFS subsystems. The case study was conducted in the context of a MSc course on software architecture at an European university. Below, we explain the steps of the study that are also depicted in Figure 1.

(1) *Establish ground truth of architectural knowledge*

To evaluate the accuracy of GPT's answers, we need to determine the ground truth on the AK of HDFS. Thus, we performed the following steps:

1) *Collect architectural issues and documents*: Based on current literature [7], [21], when the developers of HDFS make architectural changes, they create an issue in the project's issue tracker, document their architectural changes in a document, attach this document to the issue, and discuss design decisions in issue comments. Thus, to establish the ground truth of AK in HDFS, we need to find issues containing design decisions, especially those with attached architectural documents. To find issues with design decisions and their respective architectural documents, we analyzed an existing dataset of architectural issues [23]. The dataset involves 403 manually classified architectural issues from HDFS. Thus, we first filtered out issues from this dataset, which contains the attached documents. Then, we manually inspected the filtered issues to ensure that the attached documents were actually architectural. In total, we collected 46 architectural documents with their respective issues, which are available online [20].

2) *Select sub-systems of HDFS for analysis*: HDFS is a large-scale system, which is challenging to analyze regarding all of its AK concepts. Thus, we decided to focus on specific sub-systems of HDFS that allow us to delve into many of their AK concepts. To select sub-systems, we checked the collected architectural documents (from the previous step) regarding their quality and richness of AK. We selected in total seven sub-systems. We provide detailed descriptions for each subsystem of HDFS online.

3) *Form groups to analyze sub-systems*: To analyze the sub-systems of HDFS, we formed six groups from the 14 participants, with each group focusing on one or two subsystems. Forming groups is needed to ensure the reliability of the ground truth because multiple participants will analyze the AK of each subsystem. In the following three steps, we establish the ground truth for each AK concept.

4) *Analyze AK concepts*: Each group analyzed AK concepts relevant to their assigned sub-systems. In detail, each group explored three main AK concepts: *components and connectors*, *quality solutions*, and *rationale of design decisions*. To find each AK concept, groups used different methods and resources, which we explain below:

- *For components and connectors*, each group extracted *explicitly* documented components and connectors in the architectural documents. These could be either textually or graphically documented. Because HDFS is a fast-evolving system and architectural documents might be outdated, each group explored the components and connectors for their subsystems from the source code of the recent version of HDFS 3.4.0. To analyze source code, groups used the source code analysis tool Structure101[3] to visualize the components and connectors and compare them with the ones from the architectural documents. As a result, each group determined a list of components and connectors with their description in each sub-system.
- *For quality solutions*, each group searched the architectural documents and source code of their respective sub-systems for solutions to quality attributes, such as tactics or patterns. For architectural documents, groups manually analyzed and annotated explicitly mentioned tactics or patterns. In source code, groups used keywords-searches of names of well-known tactics and patterns [2]. Furthermore, groups used Archie [24] to search for tactics on security and availability. Each retrieved tactic has been manually verified for its correctness. As a result, each group determined a list of applied quality solutions in each sub-system.
- *For the rationale of design decisions*, groups manually analyzed architectural documents, and annotated prominent rationale concepts: assumptions [25], trade-offs [26], risks [26], alternatives [27], and benefits and drawbacks [27]. Because architectural documents might lack the rationale of decisions, groups also analyzed design discussions in the comments of their respective issues in issue trackers to determine the rationale of design decisions.

5) *Review and verify ground truth*: The paper's first author verified each identified AK concept from the previous step to ensure alignment with their classified AK concept and that the ground truth is supported with sufficient evidence. We share our analyzed ground truth of HDFS online [20].

(2) *Ask GPT for architectural knowledge*

To evaluate GPT as used by software engineers, we requested the participants to ask GPT questions about the AK concepts in HDFS. Table I shows the number of asked questions, and Table II presents examples of questions asked by the participants to GPT, Figure 2 shows an example interaction with a question and GPT's answer. All other questions are available online [20]. Each participant asked GPT about the AK concepts in their respective subsystems independently of each other to ensure a diversity of questions and mitigate bias regarding the influence of prompts on the GPT results. Participants could freely phrase their questions; they specifically formulated direct questions (i.e., zero-shot) without training data or examples, which aligns with software developers' common use of LLMs in their daily work.

After each question posed by a participant, we asked participants to evaluate the answer generated by GPT regarding

TABLE I: The number of questions for the different types of solution

Type of Solution	Count
Component and Connectors	81
Quality Attribute Solutions	87
Rationale of Decisions	68

[3]https://structure101.com/

TABLE II: Example questions for the different types of solution

Type of Solution	Examples
Component and Connectors	• In hdfs, in ReplicasWritable component, what components are involved in selecting in-RAM tmpfs? • What are the architectural components of fsdataset in Hadoop HDFS?
Quality Attribute Solutions	• What architectural patterns were utilized to optimize the performance of QJournal in Hadoop HDFS? • What tactics have been implemented to improve the security of the Storage Policy Satisfier in HDFS?
Rationale of Decisions	• In Hadoop HDFS using the NFS interface, why does the user need to read data from the NFS buffer instead of the DFS client? • What alternative solutions does QJournal offer?

Question: *How the components of fsdataset in HDFS interact?*
Answer: The components of the fsdataset in Hadoop HDFS interact with each other through a coordinated process. [..]
• *Client*: The client interacts with the NameNode [..]. It sends [..] requests to the appropriate DataNodes [..].
• *NameNode*: The NameNode maintains the metadata [..]. It receives [..] requests from clients [..].
• *DataNode*: The DataNodes [..] communicate with the NameNode and other DataNodes to perform data read/write [..]
• *Block Scanner*: The Block Scanner periodically scans the fsdataset [..]. It sends the block reports to the NameNode for further processing.
• *Replication*: The NameNode monitors the health and availability of the DataNodes. If a DataNode fails [..] the NameNode [..] selects new DataNodes [..] and coordinates the replication.
• *Balancer*: The Balancer is responsible for balancing the data across the DataNodes in the cluster. It periodically monitors the distribution of data blocks and moves them between DataNodes.
Overall, the components of the fsdataset interact through a combination of client-server communication, metadata coordination, and data transfer operations [..].

Fig. 2: Example question with GPT's answer

quality and *trustworthiness*, using two separate Likert scales. For quality, we designed the following Likert scale:

- *Very high quality (5)*: The whole answer is well structured, and contains conscience information without repetition.
- *High quality (4)*: The whole answer is well structured, and part of the answer is conscience, but another part involves repetition of information.
- *Medium quality (3)*: The answer is partially well structured, but more than 50% of the answer is a repetition.
- *Low quality (2)*: The answer is poorly structured, and most of the answer is a repetition.
- *Very low quality (1)*: The whole answer has no clear structure and is a repetition.

For trustworthiness, we designed the following Likert scale:

- *Very high trustworthiness (5)*: The whole answer contains reasonable information, supported by clear arguments and evidence.
- *High trustworthiness (4)*: Most of the answer contains reasonable information. However, part of the answer requires validation.
- *Medium trustworthiness (3)*: Part of the answer contains reasonable information. However, most of the answer requires validation.
- *Low trustworthiness (2)*: A small part of the answer contains reasonable information. And, most of the answer lacks arguments and evidence.
- *Very low trustworthiness (1)*: The whole answer provides insufficient arguments and evidence that cannot be trusted.

In addition to the Likert scale evaluation, we asked participants to briefly justify their evaluation in a few sentences. To facilitate the collection and storage of questions, answers, and evaluations, we developed a custom tool that calls the GPT API for question answering, and stores the data in a structured format. As a result of this step, we collected the following data: 236 questions and answers about the AK concepts and the participants' evaluations regarding the quality and trustworthiness of GPT's answers. All data of the study are available online [20]. We analyzed the collected data to answer the RQs, which we will explain in the following two sections.

(3) *Analyze the accuracy of GPT's answers*

To evaluate the accuracy of GPT (i.e., answer RQ1), we compare GPT's answers with the correct answers based on our ground truth from Step (1). To prevent bias, this step was conducted by an independent researcher who did not participate in establishing the ground truth of AK from HDFS. Furthermore, the analysis has been additionally verified by the paper's first author to ensure reliability.

For each question to GPT, we first determined the correct answer from our ground truth (see Section II-A). We then analyzed each answer generated by GPT to determine the correct and false parts of the answer. Fortunately, GPT generates answers as a list of bullet points (see Figure 2), which facilitates our analysis of the answer to determine its correct and false points. Specifically, we determined the following:

- *True positive (TP)*: Correct points in an answer that align with our ground truth.
- *False positive (FP)*: False points in an answer that do not align with our ground truth.
- *False negative (FN)*: GPT's missed correct answers according to our ground truth.

To ensure the validity of our analysis, we extra-checked all false positives within the HDFS resources (i.e., documents, issues, and source code) to ensure that these points are certainly false positives. Based on the numbers of TP, FP, and FN, we calculated standard metrics to measure the accuracy of GPT's answers: Precision, recall, and F1-score. We measured these metrics separately for each AK concept, which allowed us to compare the accuracy of GPT for questions about different AK concepts. For summarization, we calculate the average and the micro average for all metrics. Furthermore, we executed significance tests, i.e., the T-test for normally distributed data and the Mann-Whitney U Test for other skewed data.

In addition to the quantitative accuracy analysis, we qualitatively analyzed the false positive points generated by GPT

AK concept	Precision	Recall	F_1
Component & Connectors	0.508	0.574	0.539
Quality Attribute Solutions	0.333	0.588	0.425
Rationale of Decisions	0.345	0.620	0.444
Average	0.395	0.594	0.469
Query Average	0.389	0.588	0.469

TABLE III: Accuracy of GPT's answers, micro averages

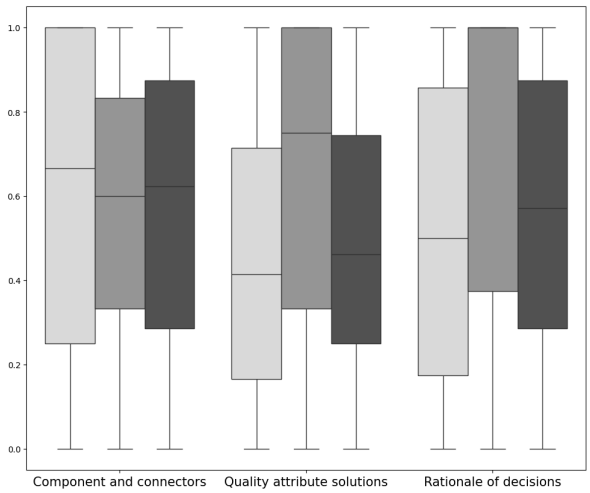

Fig. 3: Average performance for the queries. Light gray is precision, medium gray is recall, and dark gray is F_1-score.

to determine *confusions* of GPT. For each false positive, we determined its AK concept (e.g., whether it is a quality attribute, a tactic, or a requirement), and associated it with the goal of a question. The analysis has been conducted by the first author and an independent researcher. Based on our analysis of false positive points, we understand the reasons for poor precision and confusions of GPT. We present the results of RQ1 in Section III.

④ *Analyze evaluations on quality and trustworthiness*

The study participants evaluated the quality and trustworthiness of GPT's answers on a Likert scale as explained in Step ②, and justified their evaluation. We analyzed this data to answer RQ2. For the evaluation on the Likert scale, we used descriptive statistics and executed suitable significant tests. For the textual justifications, two independent researchers manually analyzed the justifications of each answer using qualitative content analysis [28] to determine reasons for low ($\leq$3) evaluations. We present the results of RQ2 in Section IV.

III. RQ1 – Accuracy of GPT's answers on architectural knowledge

In this section, we present our results of RQ1 regarding the accuracy of GPT to answer questions about the AK of our case Hadoop HDFS. We present in sub-section III-A an overview of the accuracy of GPT for the three broad categories of AK concepts: components and connectors, quality attribute solutions, and the rationale of design decisions. We then delve into the accuracy details for each AK concept in the following sub-sections III-B, III-C, and III-D.

A. Overview on GPT's accuracy

We present the micro average precision, recall, and F_1-score in Table III, and their distributions in Figure 3 for the three main categories of AK concepts: components and connectors, quality attribute solutions, and the rationale of design decisions. In the following paragraphs, we discuss our observations from Table III and Figure 3.

For components and connectors, we can observe that the accuracy of GPT shows a moderate balance between precision and recall. Slightly more than half of the identified components and connectors are correct, as the precision indicates in Table III. However, precision shows a wide range of values as in Figure 3, indicating instability. Similarly, the recall in Table III suggests that GPT can identify more than half of the relevant components and connectors. However, recall has a narrower distribution than precision, as visible in Figure 3, suggesting that recall is more stable than precision. The trade-off between precision and recall results in an F_1-score of (0.539). These results mean that, when software engineers ask GPT about the components and connectors of a sub-system, they would probably get answers with nearly half of the components and connectors correctly identified.

For quality attribute solutions, GPT struggles more with precision. There is a relatively high rate of false positives, and many irrelevant elements are included in the answers, resulting in only a third of the returned points in an answer being relevant quality attribute solutions as shown in Table III and Figure 3. Different from components and connectors, the recall of GPT to identify quality attribute solutions has a higher micro-average (0.588) as presented in Table III. However, recall shows to have a wide range of values as in Figure 3, which indicates instability. Overall, the results indicate that GPT captures the majority of relevant quality attribute solutions, but the lack of specificity reduces precision. Also, the F_1-score of 0.425 highlights GPT has difficulties in accurately identifying quality attribute solutions. Thus, if software engineers ask GPT about quality solutions in a sub-system, they would probably get answers with less than half of the solutions correctly identified.

For the rationale of design decisions, similar to quality attribute solutions, GPT struggles with a high rate of false positives, resulting in a low precision of 0.345 as presented in Table III. Furthermore, the recall is quite high (0.620) as presented in Table III, showing that most relevant rationales are captured, but with considerable noise. Furthermore, the values of precision and recall have a wide range of values as in Figure 3, which indicates instability. This imbalance between precision and recall is then represented in the F_1-score (0.444). Thus, if software engineers ask GPT about the rationale of design decisions, they probably get answers with less than half of the rationale correctly identified.

Overall, GPT tends to have a better recall than precision across the three AK concepts. The average precision is relatively low (0.395), while the recall is considerably higher (0.594), leading to an average F_1-score of 0.469 as presented

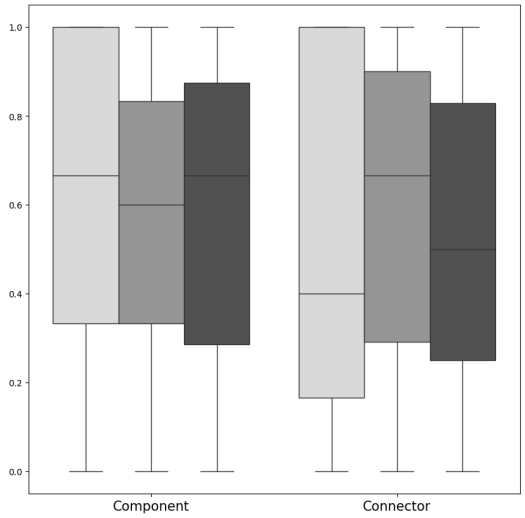

Fig. 4: Average performance for the queries about the components and connectors. Light gray is precision, medium gray is recall, and dark gray is F_1-score.

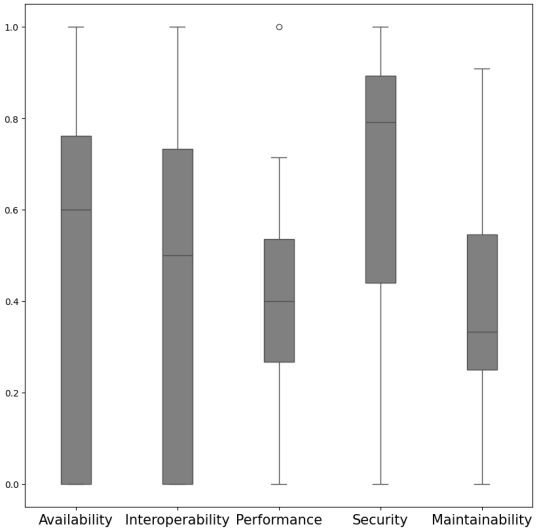

Fig. 5: F_1-scores of the detailed quality attributes

in Table III. This suggests that GPT is more inclined to retrieve a broad set of potential solutions, many of which may be incorrect. Using the Mann-Whitney U significance test, we can state that the precision for components and connectors is significantly higher than the precision for quality attribute solutions and decision rationales. All other performance values were not significantly different. Thus, software engineers could expect more accurate answers from GPT when they ask questions about components and connectors than questions on quality attribute solutions or rationales for design decisions.

B. GPT's accuracy to identify components and connectors

In Figure 4, we distinguish the accuracy of GPT based on the questions that aim to identify either components or connectors. We can observe that components have relatively higher precision; however, it is not significant according to

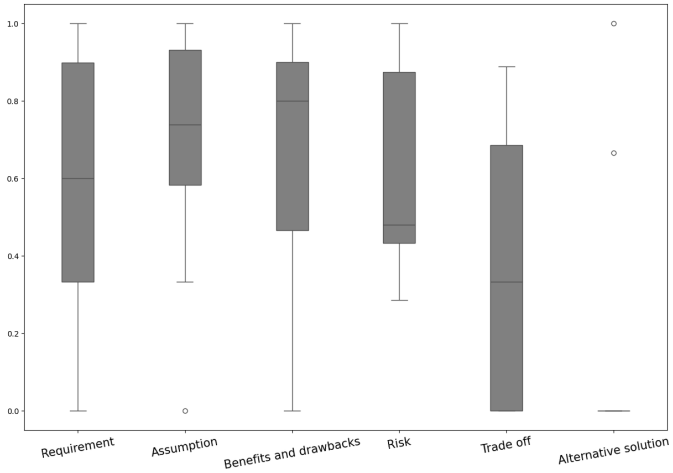

Fig. 6: F_1-scores of the detailed rationales

our significant test. In contrast, the precision of GPT to identify connectors has a wide range of values, which indicates instability. We can also observe a balance between precision and recall for both components and connectors. Thus, if software engineers ask about components or connectors, they would get nearly the same accuracy of GPT's answers.

To understand the confusion of GPT for components and connectors, we analyzed GPT's wrong answers using qualitative analysis (see Section II). Examples of false answers are shown in Table IV. We found that in 42% of the questions on components or connectors, GPT gets confused and retrieves components or connectors from other subsystems. Thus, for instance, if a software developer asks about the components for a specific sub-system, GPT would probably bring components from other sub-systems. The second-highest confusion is the confusion between components and connectors, with 34%. Thus, if a software developer asks about components, GPT might answer with connectors, and vice versa. Other less common confusions are the confusion between components, quality attributes, and tactics. This means that GPT can differentiate the main AK concepts but makes mistakes when distinguishing specific elements with the same AK concept.

C. GPT's accuracy in extracting quality attribute solutions

In Figure 5, we distinguish the F_1-score of GPT to answer questions about different quality attributes. We note that we decided on the quality attributes in Figure 5, as these were explicitly found in HDFS's ground truth. We can observe that questions on security solutions (e.g., security tactics) have the highest F_1-score, while questions on availability and interoperability solutions have the second-best median of F_1-score but a wide distribution, which indicates instability. Finally, both performance and maintainability have the least F_1-score, with a narrow distribution, which verifies their limited F_1-score. Based on our significant tests (see Section II), GPT has a significantly better F_1-score to answer questions about security solutions compared to performance and maintainability solutions. In all other cases, there are no significant differ-

TABLE IV: Example false answers for questions about Components & Connectors (C&C), Quality Attributes (Q), and the Rationale (R).

Type	Question	Ground Truth	Falsely Found	Reason
C&C	What are the connectors of Disk Balancer in Hadoop HDFS?	Json Connector, Null Connector, DBNameNode Connector	HDFS Connector, Disk Connector, Balancer Connector	They are connectors, but not for Disk Balancer.
C&C	What are the main components of StoragePolicySatisfier in HDFS?	BlockMovementTask, BlockDispatcher, BlockStorageMovementTracker	StoragePolicySatisfier Daemon, NameNode, BlockPlacementPolicy, Block Movement Coordinator, Balancer, PolicyMonitor	The listed components are general components of Hadoop HDFS unrelated to StoragePolicySatisfier.
Q	What are the architectural patterns used to improve maintainability in the implementation of Block Management in Hadoop HDFS?	Layered Architecture	Strategy pattern, Observer pattern	The architectural patterns improve maintainability, but this does not pertain to the implementation of Block Management.
Q	What quality attributes does the component writable replicas fulfill in Hadoop HDFS?	High availability, Data Replication, Distributed Storage, High Throughput	Data Reliability, Data Availability, Fault Tolerance, Scalability, Consistency, Data Integrity	Confusing quality attributes
Q	What architectural patterns were put into effect to enhance the security of QJournal in HDFS?	Secure communication, authentication, authorization, data encryption, auditing, logging, secure config.	Fault tolerance design	Fault tolerance design is a tactic rather than an architectural pattern.
R	In HDFS, the component Volume implements the performance tactic of reducing overhead. What assumptions did the developers consider?	Read-heavy workloads	Large File System, Data Integrity	Mixing up assumptions.
R	In HDFS, the CLI implements the Discover Service Tactic. What alternatives were considered?	Distributed Key-Value Store (e.g., ZooKeeper), Static Configuration	DNS-based Service Discovery, Service Registry	Confusing alternatives.

ences in F_1-score among the different quality attributes. Thus, software engineers can expect answers with higher accuracy when asking questions about security solutions compared to performance or maintainability solutions.

To understand the confusion of GPT for quality attribute questions, and similar to components and connectors, we analyzed GPT's wrong answers using qualitative analysis (see Section II). Table IV shows examples of false answers. We found that GPT gets confused in 87% of the wrong questions and retrieves other solutions for the same quality attribute. For example, if a software developer asks about the performance tactics implemented in a specific subsystem, GPT might probably bring other performance tactics that were not implemented in this specific subsystem. In fewer cases, GPT confuses solutions among quality attributes, such as confusion between tactics and design patterns or tactics and components. These results indicate that GPT grasps the differences between the different quality attributes but makes mistakes when identifying solutions for a specific one.

D. GPT's accuracy on the rationale of design decisions

In Figure 6, we distinguish the F_1-score of GPT to answer questions about different AK concepts on the rationale of design decisions. Overall, GPT has surprisingly good and quite similar F_1-score (median between 0.5 to 0.8) to answer questions about the requirements of components, assumptions behind decisions, benefits and drawbacks of solutions, and risks. While GPT has a lower and varying F_1-score to answer questions about trade-offs. In contrast, based on our significance test, GPT has significantly the lowest F_1-score to determine alternative solutions, which have not been implemented in source code, but considered and discussed by developers in issue trackers or documents.

To understand the confusion of GPT for the rationale of design decisions, we analyzed its wrong answers qualitatively (see also Section II and Table IV). In 67% of cases, GPT confused rationales within the same AK concept, such as misidentifying unrelated quality attributes for a software component. In 33% of cases, it confused AK concepts entirely, like mistaking functional requirements for tactics. This indicates GPT can distinguish between AK concepts but struggles to pinpoint specific rationales accurately.

RQ1 key takeaways:
- *Overall*, GPT has a higher recall than precision. But the precision of questions on components and connectors is significantly higher than other AK concepts.
- *For components and connectors*, GPT can correctly identify more than half of the components of a subsystem but is less precise in identifying connectors.
- *For quality solutions*, GPT is different among quality attributes. Security questions have the best accuracy, while maintainability questions have the worst.
- *For the rationale of decisions*, GPT answers rationale questions with similar accuracy but fails in questions about alternative solutions.
- *The reason for low precision* is that GPT gets confused between instances of the same AK concept among subsystems or components or decisions. Still, GPT can successfully differentiate between different AK concepts.

IV. RQ2 – Quality and Trustworthiness of GPT's Answers on Architectural Knowledge

Table V and Figure 7 present the evaluation of the quality and trustworthiness of GPT's answers as evaluated by the participants of our study. We note that because both quality and trustworthiness have the same distributions, we provide a single diagram (Figure 7) for both. *Components and connectors* show to have the highest quality and trustworthiness scores with averages of 2.969 and 2.835. This is followed by

TABLE V: Assessment of quality and trustworthiness, micro averages

AK concepts	Quality	Trustworthiness
Component & Connectors	2.969	2.835
Quality Attribute Sol.	2.290	2.177
Rationale of Decisions	2.662	2.708
Average	2.640	2.573

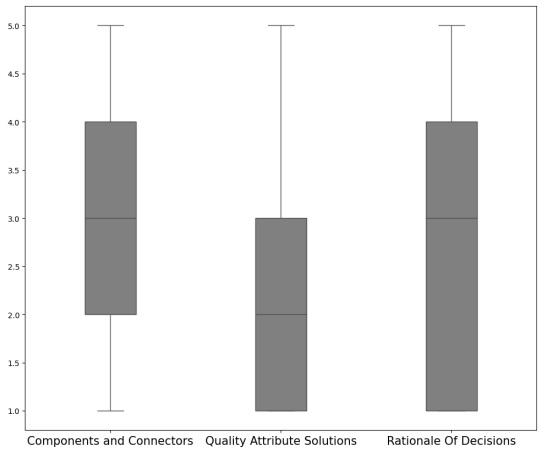

Fig. 7: Assessment of the quality and trustworthiness of the answers

rationale of design decisions with average scores of 2.662 and 2.708. The lowest scores were given to questions about *quality attribute solutions* with average scores of 2.290 and 2.177. Our significance test shows that the quality and trustworthiness of *components and connectors* is significantly better than that of *quality attribute solutions*. These results mean that GPT produces higher-quality responses and sufficient content for questions on components and connectors that software engineers can trust. In contrast, GPT struggles to produce high-quality and reliable answers for quality attribute solutions, likely due to complex patterns and the tactics' nuanced and context-dependent nature.

In general, GPT provides moderately good (quality: 2.640) and trustworthiness (2.573). However, the scores suggest room for improvement, particularly in ensuring higher-quality responses and more consistent reliability. As explained in Section II, the participants of our study justified their evaluations. We analyzed these justifications from the participants for answers with low quality and low trustworthiness (i.e., with evaluations < 3). Below, we present the results of our analysis and their percentages from low-evaluated answers. We note that the sum of percentages can exceed 100%, as participants mentioned multiple reasons in their justifications.

The participants mentioned three reasons for low quality:

- *Unclear and generic answers (58%)* in GPT's answers were the biggest reason for low-quality evaluations. For example, one participant wrote, "*The answers are not that clear It can be seen that CLI is discussed but it is discussed more generally and It cannot be seen what can we do with CLI*".
- *Obviously wrong answers (37%)*. For example, one participant mentioned, "*The rating is 1 because not even a single answer is a connector*".
- *Repetition of knowledge (11%)*, when answers involve them.

Participants mentioned two reasons for low trustworthiness:

- *Unclear arguments (59%)* as the generated answers lack clear arguments on their explanations. For example, one participant justified his low trustworthiness with "*It is not clearly instructed how the connectors work with their components, and some connectors don't seem to be connectors*".
- *Lack of evidence (49%)* as the generated answers do not explain the source of knowledge. For example, one participant said, "*I think the generated text is good, but it does not have any evidence for trustworthiness*".

RQ2 key takeaways:
- *Answers on components and connectors* have the best quality and trustworthiness; this is followed by questions on the rationale of design decisions.
- *Answers on quality attribute solutions* have significantly the worst quality and trustworthiness.
- *Reasons for low quality* answers of GPT are unclear, generic and wrong answers, as well as repetitions.
- *Reasons for low trustworthiness* on GPT's answers are unclear arguments and lack of evidence.
- *Overall*, the quality and trustworthiness of GPT's answers are moderate and require software engineers to review and verify answers.

V. Discussion

A. Implications for researchers

The overall accuracy in Section III suggests that, while GPT is moderately capable of answering relevant AK concepts, they are prone to include irrelevant information, as indicated by the lower precision scores. Furthermore, GPT tends to err on the side of inclusiveness, as indicated by the higher recall. However, this leads to less reliable outputs that require expensive human validation. These results should motivate researchers to develop approaches that validate the results of LLMs. One option could be using the results of LLMs and searching in software repositories (e.g., version control systems or issue tracker) for AK concepts (e.g., components or tactics) to verify the outputs of LLMs.

The results in Section III also show that GPT can correctly identify more than half of the architectural components and over a third of quality solutions. While this accuracy is far from ideal, it is surprising, given the little information provided to GPT in the questions (i.e., zero-shot). These results are comparable to the accuracy of traditional architectural recovery algorithms (e.g., [5]) that detect components from source code. For example, the ACDC had the highest accuracy for detecting components in Hadoop with 62%. In contrast, approaches to identify quality solutions (e.g. tactics) from source code achieve better accuracy than LLMs [29]. For example, Archi [24], [29] can detect availability and security tactics with an F_1-score of 0.8. However, one advantage of LLMs over source code analysis is that LLMs do not require developing special techniques to analyze specific architectures (e.g., micro-services [30]). While LLMs and source code

analysis have entirely different mechanisms, there is a big potential for combining them to improve the accuracy of detecting architectural components and quality solutions.

The results in Section III-D about the accuracy of GPT regarding the rationale of design decisions show that GPT can differentiate between most of the rationale concepts, but struggles to identify alternative solutions and trade-offs. These results should direct researchers to utilize LLMs to detect rationale concepts from software repositories and support LLMs with further approaches to detect missing concepts, such as alternative solutions and trade-offs.

The results of RQ2 in Section IV show that LLMs show promise in helping to answer architectural questions. Yet, there is still considerable room for improvement, particularly in ensuring higher-quality and more trustworthy answers in the more complex and abstract aspects of architectural analysis. Our identified reasons for low quality and trustworthiness can guide researchers to develop approaches that improve quality and trustworthiness. One idea to improve the trustworthiness of the answers is to provide traceability links between the answers of LLMs and their evidence in software repositories. Another way to improve the quality of answers is to support them with contextual details from repositories and use visual diagrams to support their precision. For example, lower scores on quality attribute solutions (e.g., tactics) could be supported using architectural models to show how tactics are implemented in source code.

B. Implications for practitioners

The accuracy results of RQ1 imply that LLMs do contain AK. Thus, they could be useful for generating initial drafts of architectural documentation or identifying potential architectural components and decisions in existing systems, helping practitioners understand the architecture of existing software systems. However, further refinement and validation by experts would still be necessary. Thus, given the required resources of high-performance computers, large organizations could pre-train LLMs using repositories from the software systems in these organizations to create a customized version of LLMs that contain unified knowledge of their software systems. Practitioners could use these LLMs to gain an initial architectural understanding of existing systems or generate a first version of structured architectural documentation. However, given the lower precision for some AK concepts (see Section III), LLMs are not yet reliable enough to replace expert analysis.

Practitioners could use current LLMs (e.g., GPT) to re-use AK from existing large-scale open-source systems. Most open-source systems lack documentation, and it is hard to understand their architecture. Our results in Section III show that LLMs could correctly identify many of the AK concepts of HDFS. Thus, practitioners could use current LLMs as a learning source to understand and possibly re-use AK from these systems. Our accuracy results can also guide practitioners on which AK concepts can be trusted and which require extra validation from the system's software repositories.

The results of RQ2 on the quality and trustworthiness of architectural questions inform practitioners about the expected quality and trustworthiness of the different AK concepts. Thus, if software engineers posed questions regarding quality attribute solutions, such as patterns or tactics, they could expect lower quality and less evidence in their answers. In these questions, practitioners need to verify the answers of GPT or support them with evidence from software repositories.

VI. Threats to validity

In this section, we discuss the threats to validity based on the guidelines by Runeson & Höst [22] and by Sallou et al. [31].

a) ***Construct validity****:* One threat to construct validity is the formulation of questions by the study's participants, which might have biased GPT's answers. To mitigate this threat, we formed groups and asked members of each group to formulate their questions independently. This way, participants asked different forms for questions, reducing this bias.

A second threat to construct validity happened when establishing the ground truth from HDFS, in which participants analyzed different resources, documentation, source code, and issues, for AK concepts. This process requires mapping theoretical AK concepts to their respective instances in the HDFS resources. This mapping process is prone to mistakes and bias because participants might misunderstand the contents of resources or map AK concepts incorrectly. We focused on HDFS's sub-systems with the best and richest architectural documents to mitigate this threat. This allows us to provide sufficient information about the AK of this subsystem. Furthermore, participants analyzed multiple resources: documents, source code, and issues. This supports the validation of the retrieved AK through data triangulation because the same AK concepts can appear in multiple resources.

A third thread to construct validity might have occurred when asking questions to GPT because developers have also established the ground truth of AK beforehand. Their knowledge of the system's architecture might have influenced how the developers formulated the questions or evaluated the quality and trustworthiness of the answers. However, requiring software developers to specialize in specific subsystems helped them pose specific questions, which developers would ask if they became involved in a specific subsystem.

b) ***Internal validity****:* One threat to internal validity occurs when study participants evaluate the quality and trustworthiness of GPT's answers. These evaluations could be impacted by other factors, such as tiredness and distractions. Nevertheless, we asked participants to justify their evaluations, allowing them to think about their justifications. We also gave them sufficient time to ask GPT and evaluate its answers.

Another possible threat to internal validity is due to GPT's closed-source nature, which means we do not know exactly if there are other internal factors that may have influenced our results. These could be possible updates to the GPT-3.5 model or possible undiscovered faults. To ensure we used the same version, we used a dedicated tool that called the GPT-3.5 API directly, and we did not depend on GPT's user interface.

*c) **External validity**:* One threat to the external validity is that we analyzed the AK for a single software system, Hadoop HDFS. This might not generalize our results to all other open-source systems in GPT's dataset. Furthermore, it is unclear if and to what extent there is a similar amount of knowledge about other open-source systems contained in the model of GPT. If other systems contain significantly less AK in GPT's model, the results may be negatively affected. However, Hadoop HDFS has been commonly analyzed in other software architectural research efforts [7], [21] and has been selected due to its richness of resources with AK. Thus, we argue that our results could be generalized to other open-source systems with similar amounts of resources about their AK.

Another threat to external validity is that we evaluated one LLM (i.e., GPT-3.5), which has a specific size and training dataset. Our results might not be generalized to other models of different sizes and datasets. However, our results could provide the first baseline and hypothesis on LLMs' abilities to answer questions about the AK of existing systems.

*d) **Reliability**:* One major threat to reliability stems from the nature of LLMs and their variability. LLMs usually have a random component for their output to make text more diversified. This can influence the results, affecting the reliability.

Another threat to reliability comes from our applied qualitative research methods to establish the ground truth and analyze GPT's answers. To mitigate this threat, we asked multiple developers to work together in groups to establish the ground truths and agree with each other. Furthermore, two researchers analyzed the GPT answers and agreed upon each answer to ensure the analysis's agreement and replicability. Furthermore, we described our research steps in Section II, and shared our data online to facilitate replication. With this setup that uses multiple people, we aim to reduce individual bias and increase reliability.

VII. Related Work

We are not aware of other similar studies that empirically evaluate LLMs to answer questions about an existing system's AK. This section discusses related work on AK and related work that utilizes language models for software architecture.

1) Architectural knowledge: Early studies on AK (e.g., [1], [4]) established its foundational concepts, and created manual documentation tools (e.g., [32]). Recent work has investigated AK in software repositories, such as issue trackers [7], [33] and mailing lists [11], as well as on the Web through forums [34], technical documentation [35], and blogs [9]. Domain-specific AK has also been explored for quantum software [36] and machine learning systems [37]. However, these work do not utilize or evaluate LLMs, and thus different from the goal of this study.

Machine learning approaches have been applied to extract AK from various sources. For instance, Bhat et al. [33] classified issue tracker entries, while Fu et al. [38] categorized decisions in open-source mailing lists. However, these approaches do not incorporate or evaluate LLMs, which distinguishes them from this work.

2) Language Models for Software Architecture: While LLMs have been applied to various software engineering tasks [39], their use in software architecture is barely researched.

BERT, a pre-trained language model, has been employed in architectural research for tasks like detecting patterns in code [40], [41] and classifying design decisions in documentation [42] and mailing lists [11]. These works extract specific AK concepts from targeted sources, but do not use or evaluate more advanced LLMs like GPT, which offer significantly greater capabilities than BERT.

Recent studies have begun exploring LLMs for software architecture. Dhar et al. [12], [43] used GPT and T5 in zero-shot, few-shot, and fine-tuning setups to recommend design decisions based on context. Similarly, Díaz-Pace et al. [13] use GPT with retrieval-augmented generation to recommend, evaluate, and document design decisions. These works aim to support making new decisions, whereas our study utilizes GPT's ability to answer questions about the AK of existing systems.

LLMs have also been explored for generating architectural components. Eisenreich et al. [44] outlined a vision for generating architectures from requirements, and White et al. [45] proposed prompt patterns for deriving architectures. However, unlike these works, we focus on analyzing the AK of existing systems, rather than generating new architectures.

Rukmono et al. [46] proposed using LLMs with chain-of-thought prompts to perform architectural conformance checks, mapping code to architectural components and verifying alignment with specified designs. While related, their work is narrower in scope, focusing on conformance checks, whereas our study broadly explores AK in existing systems without predefined design concepts.

VIII. Conclusion and Future Work

In this study, we evaluated the capability of LLMs to answer questions about the architectural knowledge (AK) of an existing system, using the model's embedded knowledge. We conducted an exploratory case study focusing on GPT and the Hadoop HDFS system to assess the accuracy, quality, and trustworthiness of GPT's responses regarding AK of HDFS. Our results show that GPT is moderately accurate, making it a potentially valuable tool for providing an initial understanding of system architecture. However, it also exhibits notable limitations, including frequent errors that hinder its reliability. The evaluation of GPT's response quality and trustworthiness revealed that its answers are generally of moderate quality, often characterized by generic content with limited evidence or supporting arguments. Future work will extend this evaluation to other LLMs, such as GPT-4.0 and Llama, to explore whether factors such as model size or training dataset influence accuracy, quality, and trustworthiness. Additionally, we aim to develop methods and tools to enhance LLM accuracy in answering architecture-related questions and addressing the identified shortcomings.

REFERENCES

[1] P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning About Architectural Knowledge," in *Quality of Software Architectures*, vol. 4214. Springer Berlin Heidelberg, 2006, pp. 43–58.

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley, 2012.

[3] F. Buschmann, R. Meunier, H. Rohnert *et al.*, {*Pattern-Oriented*} *Software Architecture: A System of Patterns*, 1st ed. John Wiley & Sons, 7 1996.

[4] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *Journal of Systems and Software*, vol. 80, no. 6, pp. 918–934, 6 2007.

[5] J. Garcia, I. Ivkovic, and N. Medvidovic, "A comparative analysis of software architecture recovery techniques," in *International Conference on Automated Software Engineering*, 2013, pp. 486–496.

[6] M. Bhat, K. Shumaiev, K. Koch *et al.*, "An Expert Recommendation System for Design Decision Making: Who Should be Involved in Making a Design Decision?" *Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018*, pp. 85–94, 7 2018.

[7] M. Soliman, M. Galster, and P. Avgeriou, "An Exploratory Study on Architectural Knowledge in Issue Tracking Systems," *European Conf. on Software Architecture*, 9 2021. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-86044-8_8

[8] M. J. de Dieu, P. Liang, M. Shahin, and A. A. Khan, "Characterizing architecture related posts and their usefulness in stack overflow," *Journal of Systems and Software*, vol. 198, p. 111608, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223000031

[9] M. Soliman, K. Gericke, and P. Avgeriou, "Where and what do software architects blog? : An exploratory study on architectural knowledge in blogs, and their relevance to design steps," in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, 2023, pp. 129–140.

[10] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar, "A tactic-centric approach for automating traceability of quality concerns," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 639–649.

[11] M. Soliman, "Exploring architectural design decisions in mailing lists and their traceability to issue trackers," in *Software Architecture*, M. Galster, P. Scandurra, T. Mikkonen *et al.*, Eds. Cham: Springer Nature Switzerland, 2024, pp. 307–323.

[12] R. Dhar, K. Vaidhyanathan, and V. Varma, " Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical Study ," in *2024 IEEE 21st International Conference on Software Architecture (ICSA)*. Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2024, pp. 79–89. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSA59870.2024.00016

[13] J. A. Díaz-Pace, A. Tommasel, and R. Capilla, "Helping novice architects to make quality design decisions using an llm-based assistant," in *Software Architecture*, M. Galster, P. Scandurra, T. Mikkonen *et al.*, Eds. Cham: Springer Nature Switzerland, 2024, pp. 324–332.

[14] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding with unsupervised learning," 2018. [Online]. Available: https://web.archive.org/web/20230318210736/https://openai.com/research/language-unsupervised

[15] H. Touvron, T. Lavril, G. Izacard *et al.*, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[16] N. Xuanfan and L. Piji, "A systematic evaluation of large language models for natural language generation tasks," in *Proceedings of the 22nd Chinese National Conference on Computational Linguistics (Volume 2: Frontier Forum)*, J. Zhang, Ed. Harbin, China: Chinese Information Processing Society of China, Aug. 2023, pp. 40–56. [Online]. Available: https://aclanthology.org/2023.ccl-2.4

[17] L. Chen, Q. Guo, H. Jia *et al.*, "A survey on evaluating large language models in code generation tasks," 2024. [Online]. Available: https://arxiv.org/abs/2408.16498

[18] H. Naveed, A. U. Khan, S. Qiu *et al.*, "A comprehensive overview of large language models," 2024. [Online]. Available: https://arxiv.org/abs/2307.06435

[19] T. Brown, B. Mann, N. Ryder *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell *et al.*, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[20] M. Soliman and J. Keim, "Supplementary material for 'Do Large Language Models Contain Software Architectural Knowledge? An Exploratory Case Study with GPT'." [Online]. Available: https://doi.org/10.5281/zenodo.14512761

[21] R. Kazman, D. Goldenson, I. Monarch *et al.*, "Evaluating the effects of architectural documentation: A case study of a large scale open source project," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 220–260, 2016.

[22] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir Software Eng*, vol. 14, no. 2, p. 131, Dec. 2008.

[23] J. Maarleveld, A. Dekker, S. Druyts, and M. Soliman, "Maestro: A deep learning based tool to find and explore architectural design decisions in issue tracking systems," in *Software Architecture. ECSA 2023 Tracks, Workshops, and Doctoral Symposium*, B. Tekinerdoğan, R. Spalazzese, H. Sözer *et al.*, Eds. Cham: Springer Nature Switzerland, 2024, pp. 390–405.

[24] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar, "A tactic-centric approach for automating traceability of quality concerns," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 639–649.

[25] C. Yang, P. Liang, P. Avgeriou *et al.*, "An industrial case study on an architectural assumption documentation framework," *Journal of Systems and Software*, vol. 134, pp. 190–210, 12 2017.

[26] H. van Vliet and A. Tang, "Decision making in software architecture," *Journal of Systems and Software*, vol. 117, pp. 638–644, 7 2016.

[27] M. Soliman, M. Riebisch, and U. Zdun, "Enriching Architecture Knowledge with Technology Design Decisions," in *Working International Conf. on Software Architecture*, 5 2015.

[28] P. Mayring, *Qualitative Content Analysis. Theoretical Foundation, Basic Procedures and Software Solution*. Beltz, 2014.

[29] J. Keim, A. Kaplan, A. Koziolek, and M. Mirakhorli, "Does bert understand code? – an exploratory study on the detection of architectural tactics in code," in *Software Architecture*, A. Jansen, I. Malavolta, H. Muccini *et al.*, Eds. Cham: Springer International Publishing, 2020, pp. 220–228.

[30] N. Alshuqayran, N. Ali, and R. Evans, "Towards micro service architecture recovery: An empirical study," in *2018 IEEE International Conference on Software Architecture (ICSA)*, 2018, pp. 47–4709.

[31] J. Sallou, T. Durieux, and A. Panichella, "Breaking the Silence: the Threats of Using LLMs in Software Engineering," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER'24. New York, NY, USA: Association for Computing Machinery, 2024, p. 102–106. [Online]. Available: https://doi.org/10.1145/3639476.3639764

[32] S. Gerdes, M. Soliman, and M. Riebisch, "Decision buddy: Tool support for constraint-based design decisions during system evolution," in *Proceedings of the 1st International Workshop on Future of Software Architecture Design Assistants*, ser. FoSADA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 13–18. [Online]. Available: https://doi.org/10.1145/2751491.2751495

[33] M. Bhat, K. Shumaiev, A. Biesdorf *et al.*, "Automatic extraction of design decisions from issue management systems: A machine learning based approach," in *Lecture Notes in Computer Science*, vol. 10475 LNCS. Springer Verlag, 2017, pp. 138–154.

[34] T. Bi, P. Liang, A. Tang, and X. Xia, "Mining architecture tactics and quality attributes knowledge in Stack Overflow," *Journal of Systems and Software*, p. 111005, 5 2021.

[35] I. Gorton, R. Xu, Y. Yang *et al.*, "Experiments in Curation: Towards Machine-Assisted Construction of Software Architecture Knowledge Bases," in *IEEE/IFIP ICSA 2017*, 4 2017, pp. 79–88.

[36] M. S. Aktar, P. Liang, M. Waseem *et al.*, "Architecture decisions in quantum software systems: An empirical study on stack exchange and github," *Information and Software Technology*, vol. 177, p. 107587, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584924001927

[37] S. J. Warnett and U. Zdun, "Architectural design decisions for machine learning deployment," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*, 2022, pp. 90–100.

[38] L. Fu, P. Liang, X. Li, and C. Yang, "A machine learning based ensemble method for automatic multiclass classification of decisions," *ACM*

International Conference on Evaluation and Assessment in Software Engineering (EASE 2021), pp. 40–49, 6 2021.

[39] X. Hou, Y. Zhao, Y. Liu *et al.*, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, Sep. 2024, just Accepted. [Online]. Available: https://doi.org/10.1145/3695988

[40] J. Keim, A. Kaplan, A. Koziolek, and M. Mirakhorli, "Does bert understand code? – an exploratory study on the detection of architectural tactics in code," in *Software Architecture*, A. Jansen, I. Malavolta, H. Muccini *et al.*, Eds. Cham: Springer International Publishing, 2020, pp. 220–228.

[41] ——, "Using bert for the detection of architectural tactics in code," Karlsruher Institut für Technologie (KIT), Tech. Rep. 2, 2020.

[42] J. Keim, T. Hey, B. Sauer, and A. Koziolek, "A taxonomy for design decisions in software architecture documentation," in *Software Architecture. ECSA 2022 Tracks and Workshops*, T. Batista, T. Bureš, C. Raibulet, and H. Muccini, Eds. Cham: Springer International Publishing, 2023, pp. 439–454.

[43] R. Dhar, K. Vaidhyanathan, and V. Varma, "Leveraging generative ai for architecture knowledge management," in *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*, 2024, pp. 163–166.

[44] T. Eisenreich, S. Speth, and S. Wagner, "From requirements to architecture: An ai-based journey to semi-automatically generate software architectures," in *Proceedings of the 1st International Workshop on Designing Software*, ser. Designing '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 52–55. [Online]. Available: https://doi.org/10.1145/3643660.3643942

[45] J. White, S. Hays, Q. Fu *et al.*, *ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design*. Cham: Springer Nature Switzerland, 2024, pp. 71–108. [Online]. Available: https://doi.org/10.1007/978-3-031-55642-5_4

[46] S. A. Rukmono, L. Ochoa, and M. Chaudron, "Deductive software architecture recovery via chain-of-thought prompting," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER'24. New York, NY, USA: Association for Computing Machinery, 2024, p. 92–96. [Online]. Available: https://doi.org/10.1145/3639476.3639776