Software Architecture Meets LLMs: A Systematic Literature Review

Larissa Schmid[1⋆][0000−0002−3600−6899], Tobias Hey[2][0000−0003−0381−1020], Martin Armbruster[2][0000−0002−2554−4501], Sophie Corallo[2][0000−0002−1531−2977], Dominik Fuchß[2][0000−0001−6410−6769], Jan Keim[2][0000−0002−8899−7081], Haoyu Liu[2][0009−0002−7676−5010], and Anne Koziolek[2][0000−0002−1593−3394]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
lgschmid@kth.se
[2] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{hey,martin.armbruster,sophie.corallo,dominik.fuchss,
jan.keim,haoyu.liu,koziolek}@kit.edu

Abstract. Large Language Models (LLMs) are used for many different software engineering tasks. In software architecture, they have been applied to tasks such as classification of design decisions, detection of design patterns, and generation of software architecture design from requirements. However, there is little overview on how well they work, what challenges exist, and what open problems remain. In this paper, we present a systematic literature review on the use of LLMs in software architecture. We analyze 18 research articles to answer five research questions, such as which software architecture tasks LLMs are used for, how much automation they provide, which models and techniques are used, and how these approaches are evaluated. Our findings show that while LLMs are increasingly applied to a variety of software architecture tasks and often outperform baselines, some areas, such as generating source code from architectural design, cloud-native computing and architecture, and checking conformance remain underexplored. Although current approaches mostly use simple prompting techniques, we identify a growing research interest in refining LLM-based approaches by integrating advanced techniques.

Keywords: Systematic Literature Review · Software Architecture · Large Language Models · LLM4SA · Software Engineering · LLM4SE.

1 Introduction

Large language models (LLMs) are revolutionizing software engineering by offering exceptional capabilities in natural language understanding and generation. These models can significantly enhance productivity by, e.g., automating code generation [1], helping in the development of cyber-physical systems [2] and digital twins [23, 36], analyzing logs [21, 34], and answering developing questions [25].

⋆ Work was conducted while affiliated with Karlsruhe Institute of Technology (KIT).

Their ability to analyze vast amounts of data and identify patterns also helps in optimizing system performance and predicting potential issues early: LLMs can not only streamline workflows but also foster innovation and improve overall software quality.

Software architecture tasks also often require vast knowledge. Consequently, there is an emerging synergy between software architecture and LLMs. For example, LLMs have been explored for architecture tasks such as identifying design decisions [15, 27], generating architecture designs from requirements [13], and answering questions about architectural knowledge [28]. Moreover, software architecture can be applied to developing LLM-based systems by providing reference architectures for different use cases [7, 22, 26, 32].

Systematic literature reviews on the use of LLMs provide researchers and practitioners with comprehensive insights into current trends, challenges, and best practices. These reviews help identify gaps in existing knowledge, guide future research directions, and inform evidence-based decision-making in the development and application of LLMs. However, reviews on the use of LLMs in the scope of software engineering mainly focus on testing [31] or code generation [35]. Most articles in existing literature reviews on LLM usage in general software engineering [8, 11] are not related to software architecture: Hou *et al.* [11] do not include works from the software architecture community, as the search keys did not include relevant terms (only "software design" could be considered related). Even for design-related tasks, Fan *et al.* [8] report that they did not find much work on LLM-based software design. Different from code generation and tests, architecture tasks often affect higher level concerns and encounter data scarcity problems. These disparities highlight the value of a comprehensive review of software architecture and LLMs. To the best of our knowledge, no such review exists, making a systematic literature review particularly useful.

Therefore, in this paper, we conduct a systematic literature review of research papers at the intersection between LLMs and software architecture. We formulate our research questions to derive insights into the current state-of-the-art in this field and about what is working well, challenges, and open questions. From the software architecture side, we analyze the software architecture tasks targeted by these works and how the performance of LLMs is evaluated. From the LLMs' side, we explore which LLMs are used and how they are optimized. To provide insights into the path ahead for the synergy between LLMs and software architecture, we also analyze the discussed future work. Moreover, we give an initial overview on envisioned reference architectures for developing LLM systems. Following the methodology for systematic literature reviews on software engineering [18, 19], we initially found 119 with our search strategy, of which we identify and analyze 18 relevant papers about LLMs and architecture. We provide the complete data of this survey as supplementary material [29].

This literature review can benefit (i) software architecture researchers who want to apply LLMs in their architecture tasks and (ii) LLM-based systems developers who want to build their LLM systems with better architecture.

In the following, we first present our methodology to the review in Section 2. After that, Section 3 presents our findings on our RQs, and Section 4 discusses threats to validity. We further discuss our findings and outline future research directions in Section 5. Finally, Section 6 concludes the paper.

2 Methodology

This section describes the approach we followed to select, analyze, and evaluate relevant research on the intersection of software architecture and LLMs. We follow the methodology defined by Kitchenham et al. [18, 19]. Therefore, our review process consists of three main phases: 1. Planning the review by formulating research questions of interest (Section 2.1) and defining a search strategy (Section 2.2), 2. filtering the articles obtained by our search (Section 2.3), and 3. analyzing the remaining relevant articles (Section 2.4).

2.1 Research Questions

Our review aims to provide an overview of the current applications of LLMs to software architecture research and vice versa, i.e., how software architecture research is applied to LLMs. We want to provide insight into what works well, what does not, and what challenges remain.

First, we investigate which software architecture tasks LLMs are used for (**RQ1**) to understand which tasks are already being researched and potentially solved, and which remain an open challenge. To gain more detailed insight, we examine the degree of automation these approaches provide (**RQ1.1**), distinguishing between manual guidance, semi-automated, and fully automated methods. Additionally, we assess whether LLMs are applied end-to-end or only to specific sub-tasks within the broader software architecture process (**RQ1.2**).

Since the LLMs' capabilities can vary significantly, our goal is to identify which LLMs are used in the reviewed studies (**RQ2**). This research question provides insight into the most commonly applied models and whether there is a preference for general-purpose or domain-specific LLMs in software architecture.

To understand how researchers tune LLM performance, we examine the techniques used to improve effectiveness (**RQ3**). Specifically, we investigate the used tuning techniques (**RQ3.1**) and prompt engineering strategies (**RQ3.2**). RQ3 as well as RQ2 are based on the investigation by Hou *et al.* [11].

Evaluating the effectiveness of LLM-based approaches is crucial for understanding their practical applicability. Therefore, we explore how these approaches are evaluated (**RQ4**) by analyzing the evaluation methods used (**RQ4.1**) [20] and the specific metrics applied (**RQ4.2**). Furthermore, we examine whether these methods outperform existing baselines (**RQ4.3**) and assess whether supplementary materials are provided (**RQ4.4**) to support reproducibility.

Finally, to gain insights into the future directions of LLM research in software architecture, we analyze what future work the authors of the reviewed studies suggest (**RQ5**). Identifying open challenges and proposed research directions helps outline the next steps to advance LLM applications in this domain.

2.2 Search Strategy

We extracted a search query based on our goal to provide an overview of the applications of LLMs in software architecture and vice versa. Therefore, one part of our query is the keyword *software architecture* that has to be within the article for us to regard it as relevant. Moreover, the article has to contain a keyword related to LLMs. As not all articles may use the same keyword related to their usage of LLMs, we included several different terms the article has to contain at least one of, including the currently most popular models: *"LLM" OR "language model" or "language models" OR "generative AI" OR "bert" OR "GPT" OR "Llama" OR "Transformer"*.

We use this search string to search for articles in 25 top software engineering conferences and journals, such as ICSE, ASE, ICSA, ECSA, TSE, TOSEM, by using Google Scholar and defining them as sources of the publication. We provide the complete list of venues as part of our supplementary material [29]. For the most closely related conferences, namely ICSA and ECSA, we modify the search string not to need to contain the term *software architecture*, as the scope of the conference already implies that the article is related to software architecture. Moreover, we also include companion proceedings from these two conferences. This search leads to 119 articles.

2.3 Filtering of Results

Based on our initial search, resulting in 119 articles, we filter the results to make sure the articles are actually relevant to our survey. First, we check if they contain the term *software architecture* as part of the article and find that 44 articles only mention it as part of the references, e.g., when citing an article from a software architecture conference. Next, we assess if an article is a full article from the research track of the respective conference – except for ICSA and ECSA, where we also consider contributions from the companion proceedings – and if it conducts research on the topic of software architecture and LLMs. We distribute this step among the team of authors and refer to the ICSA scope of topics in the call for papers for determining if the article is in the scope of software architecture as *inclusion criteria*. Notably, this implies the *exclusion criteria* that domain and UML models like class and activity diagrams are excluded. Moreover, we *exclude* research related to only design patterns as opposed to architectural patterns. We filter 12 articles based on not being full articles, and 15 more because they are not related to software architecture and LLMs. We filter one additional article because it is a survey article; therefore, it only discusses existing research and does not propose a new approach. We notice that two more articles, while presenting slightly different ideas, show the same evaluation. Therefore, we subsume them into one article.

2.4 Analysis of articles

After filtering the results, we end up with **18 unique and relevant articles**. Figure 1 shows their distribution by venue and year of publication. While the first article was already published in 2020 at ECSA, there was only one publication in the following years, and there was no publication in 2023. However, 2024 shows a steep increase with 10 articles, five of them being part of companion proceedings. In 2025, five articles have already been published, further indicating that the upward trend in publications will continue. Most articles (14/18) are published at ICSA, ECSA, or in their companion proceedings. This is not surprising as both conferences are the most closely related to the topic of our review.

We then extracted relevant data from the articles to answer our research questions outlined above (cf. Section 2.1).

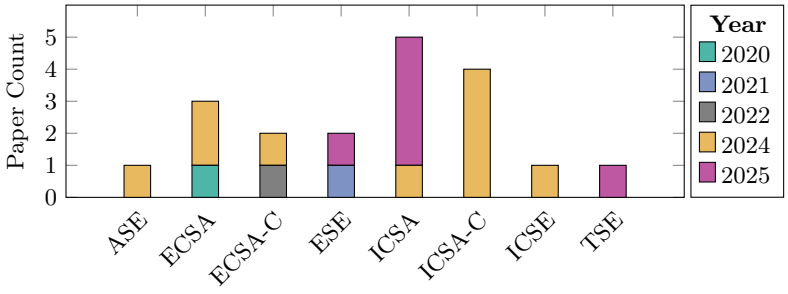


Fig. 1: Distribution of relevant articles regarding venue and year of publication.

3 Findings

In the following, we present our findings from the analysis of articles on our research questions.

3.1 RQ1: Software Architecture Tasks

Our first research question investigates the software architecture tasks, the degree of automation of the approaches, and how LLMs are utilized in these tasks.

We identified four main categories of software tasks related to software architecture that utilize LLMs: Reference Architectures, Classification & Detection, Extraction & Generation, and Assistants. We provide an overview of the distribution of these categories in Figure 2.

Reference architectures cover domains such as self-adaptive systems [7], chatbots with LLMs [32], and agents [22, 26]. We discuss them shortly in Section 5.

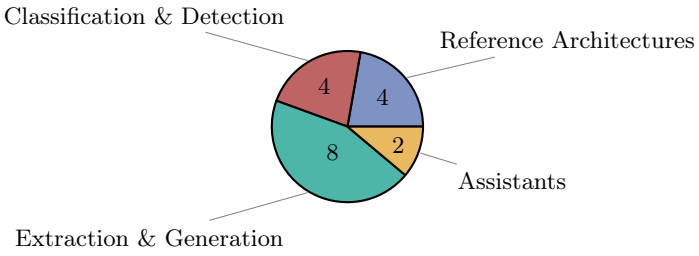


Fig. 2: Distribution of Tasks Utilizing LLMs (n=18)

Classification and detection tasks include classifying tactics in code [16], design decisions [15], and identifying design decisions in mailing lists [27]. LLMs are also used as classifiers in traceability link recovery tasks [17].

Extraction and generation tasks involve extracting design rationales [37], architecture component names [10], design structures from code [9], and mining design discussions [24]. Regarding generation, creating architecture decision records [4], software architecture designs from requirements [13], and architecture components for FaaS [3] are application scenarios for LLMs. Also, the generation of module descriptions and text embeddings for model-to-code mappings [14] are part of this category.

Assistant systems focus on question-answering about architectural knowledge [28] and aiding in selecting, assessing, and capturing better design decisions [5].

Most of the works (71 %) use LLMs in an automated fashion. The two approaches that build assistants or chatbots are semi-automated, as they require user interaction. In the remaining categories, only two further studies are classified as semi-automated, while the rest are fully automated. The semi-automation is related to either providing adaptable infrastructure components for identifying types of architectural design decisions rather than fully automation [27] or requiring the user to define and enter prompts themselves [13].

Whether the LLM is used to solve a subtask or the entire task is mixed across the studies. While 64 % of studies use LLMs end-to-end, 36 % of studies use them for subtasks. We observed the following subtasks for the non-assistant categories: Classification tasks [27], generation of descriptions or embeddings [14], extraction of component names [10], and generation of explanations [9]. Moreover, one of the assistants [5] uses LLMs for multiple subtasks like suggesting patterns, ranking, assessment of decisions, and generation of architecture decision records.

3.2 RQ2: Which Large Language Models are used?

To give a more detailed insight into the capabilities of the LLMs used in the studies, we analyzed the distribution of the respective models. This distribution is displayed in Figure 3. In total, 23 different models were used, which we grouped according to the base approach they derived from. However, all but ULMFiT are based on the Transformer architecture [30]. We included ULMFiT [12] anyways,

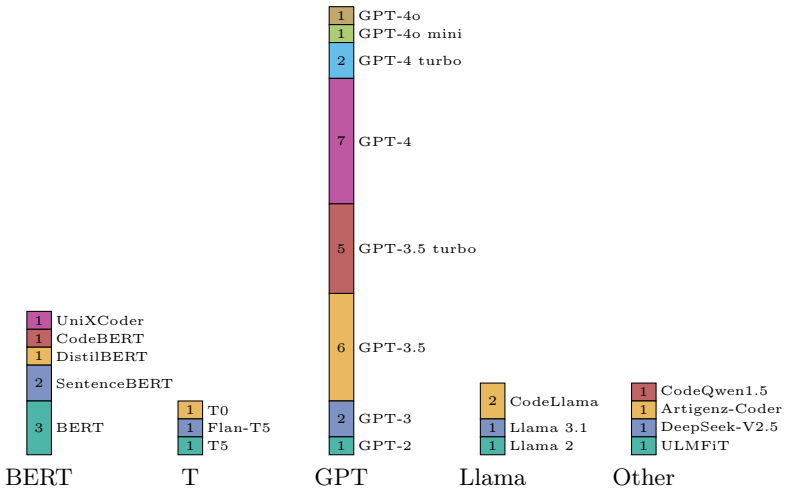
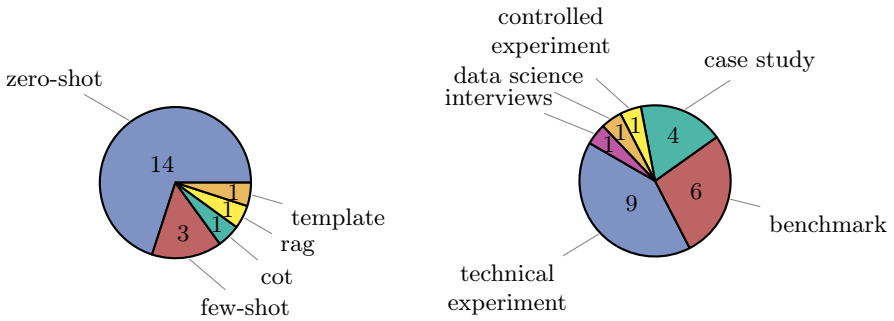


Fig. 3: Distribution of Used Models Grouped by Their Base Approach

as it is the first language model that introduces transfer learning with task-specific fine-tuning and can thus be seen as the direct predecessor of the current LLMs. The first observation one can make is that most of the used models (73 %) are only using the decoder-part of the Transformer architecture (GPT-, Llama-, and DeepSeek-based models). Encoder-only models (BERT-based and ULMFiT) are used in 21 % of the studies and encoder-decoder models (T-based) only in 7 % of the cases. This was expected, as since the release of GPT-3 the auto-regressive (decoder-only) LLMs surpassed the other variants in most SE tasks [11]. This also aligns with the distribution of the models over time: until 2024, solely encoder-only models were used (two times BERT [15, 16] and one time ULMFiT [24]). However, as GPT-3 was released in Mai 2020, the adoption of the decoder-only LLMs in the software architecture community was slower than in other SE areas [11, 31]. The most recent versions of GPT-based models (GPT-4o and later) and Llama-based models (Llama 3.1 and later), as well as the DeepSeek-based models (DeepSeek-V2.5 and Artigenz-Coder), were only used in the 2025 publications [3, 10]. This trend might be underestimated, as our study only includes data until March 2025.

3.3 RQ3: Optimization Techniques

We observed a clear distinction in tuning approaches based on the type of model. Encoder models were consistently fine-tuned across studies, emphasizing the need for task-specific adaptation due to their transformer-based masked language modeling pre-training. Fine-tuning allows researchers to tailor the model to software architecture tasks by training it on domain-specific data.



(a) Prompting Techniques (n=20) (b) Evaluation Methods (n=22)

Fig. 4: Overview of used prompting techniques and used evaluation methods

In contrast, decoder models like those from the GPT family were predominantly utilized through prompting techniques rather than fine-tuning, likely due to accessibility and cost constraints.

Figure 4a shows the overview of prompting techniques used. We found that researchers most commonly employed zero-shot prompting (70 % of used techniques). This aligns with the general usability of LLMs, as zero-shot prompting allows direct application without further training. This can also mean that the pre-trained LLMs encode enough knowledge for many software architecture tasks.

Few-shot prompting was used less frequently (15 %), suggesting that providing examples is not necessarily required for software architecture tasks.

More advanced prompt engineering strategies were rarely applied, which could indicate an area for future exploration. Chain-of-Thought prompting was used only in one study despite its potential to improve reasoning-based tasks. Retrieval-Augmented Generation was also applied only once, indicating that integrating external knowledge sources is not yet a common practice in this domain. Similarly, template-based prompting appeared in a single instance, suggesting that structured prompt design is underexplored for software architecture tasks.

3.4 RQ4: Evaluation of Approaches

For the evaluation of LLM-based approaches in software architecture tasks (RQ4.1), the most common methods were technical experiments and benchmarking, followed by case studies (cf. Figure 4). Technical experiments were the dominant evaluation method, used in 64 % of studies (cf. Figure 4b). Benchmarking was conducted in 43 % of studies, often involving comparisons with traditional or state-of-the-art approaches. Case studies were used in 29 % of studies, offering qualitative insights into real-world applications. Other evaluation methods each only appeared once, including data science-based validation, interviews, and controlled experiments.

Looking into RQ4.2, the evaluation of the LLM-generated outputs employed both traditional performance metrics and text-generation metrics. Traditional

performance metrics (e.g., precision, recall, F_1-score) were frequently applied to measure the correctness of LLM-generated outputs. Text generation metrics, which are used to assess the quality of generated content, include BLEU (Bilingual Evaluation Understudy) and BERTScore. BLEU was adopted by three studies (21 %; i.e., [3, 4, 37]). BERTScore, which evaluates semantic similarity using contextual embeddings, appeared in one study (i.e., [4]).

A key question in assessing the effectiveness of LLMs for software architecture is whether they outperform existing approaches (RQ4.3). Among the fourteen studies analyzed, nine included a comparison to other approaches. Five studies did not compare their methods to a baseline, limiting their ability to demonstrate relative effectiveness. In cases where a comparison was conducted, LLM-based solutions consistently outperformed the baseline in six studies. Two studies showed mixed results: Mahadi *et al.* [24] demonstrated better results within-dataset, but worse across, and Keim *et al.* [15] performed better than the baselines according to the F_1-score, but showed lower precision in some cases. Another study [16] was not able to outperform the baseline. However, these results suggest a generally positive impact of LLMs on software architecture tasks. While these results highlight the potential of LLMs, the lack of baseline comparisons in one-third of the studies indicates a need for more rigorous benchmarking to establish their practical advantages.

RQ4.4 tackles reproducibility as it is a crucial aspect of scientific research, enabling independent verification of results. Nearly all studies provided some form of supplementary material, such as datasets, source code, or implementation details. However, two works proposing reference architectures did not include additional materials, possibly due to the conceptual nature of their contributions. This suggests a strong commitment to reproducibility within the field, though improvements in providing accessible and well-documented supplementary materials could further enhance transparency.

3.5 RQ5: Future Work

Regarding RQ5, we considered the future work mentioned and related to LLMs. In total, five papers (36 %) do not report on future work [14, 15, 17, 27, 37], while nine papers (64 %) give a short outlook [3, 4, 5, 9, 10, 13, 16, 24, 28] (cf. Figure 5).

In these nine papers, future work aims to expand the papers' results in three different directions. First, four studies want to use different LLMs for testing [28], with integrated reasoning [10], with code support [16], or with multimodal capabilities [5]. Second, seven studies want to improve the LLMs' results, in general, [28] or with specific approaches. This includes a preprocessing or refinement of the input [5, 10], adding more context to the input [4, 10], applying different techniques (e.g., RAG) to the LLM [3, 4, 5, 9], or fine-tuning the LLM [4, 24]. Third, in one study, the authors plan to test LLMs for software architecture tasks continuously [13].

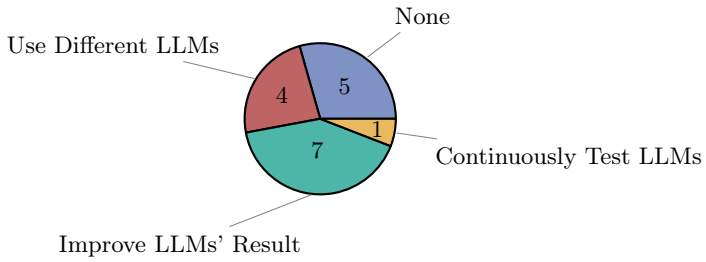


Fig. 5: Number of occurrences for different categories of future work (n=17).

4 Threats to Validity

In the following section, we discuss threats to validity [33]. One threat involves not finding all relevant articles due to our search strategy and query employed. We mitigated this threat by evaluating different queries and the relevance of papers found with them beforehand. We also checked if the results included relevant papers we knew of as a gold standard [6]. Another threat is the misclassification of articles. We need to extract information from the articles to answer our research questions, making it necessary to understand them correctly. All authors have expertise in the research field of our review, and papers were assigned based on their knowledge of the respective areas. Moreover, we discussed any issues that arose among the complete team of authors, ensuring consistent and accurate classification.

5 Discussion

In the following, we discuss our findings from Section 3 and identify future research directions.

Software Architecture Tasks. Our study shows the diverse applications of LLMs in software architecture, with tasks falling into four main categories (cf. Section 3.1): reference architectures, classification & detection, extraction & generation, and assistants. Examining the 14 articles on LLMs for software architecture, we found that most of them propose automated approaches that use LLMs end-to-end, suggesting that LLMs are capable of addressing complete architectural tasks.

Besides the 14 articles covering the applications of LLMs in software architecture, we also found four articles concerning the application of software architecture to LLMs. They propose reference architectures for incorporating LLMs into different domains, such as self-adaptive systems, chatbot frameworks, and autonomous agents. By structuring interactions between LLMs and external systems, software architecture enables more robust and adaptable applications, stressing how software architecture research can not only use LLMs but also benefit them.

Surprisingly, we found only one work generating source code for architectural components using LLMs [3]. Also, there is only one paper regarding cloud-native computing and architecture [3], indicating a potential avenue for further research in this regard. We found no articles regarding evaluating quality aspects of software architecture, such as evolvability, and architecture conformance checking. Both could be addressed in future research, e.g., by building on works identifying architectural patterns [9] and design rationales [37] from code.

Usage of LLMs. Most approaches (73 %) rely on decoder-only models (Section 3.2), particularly GPT-based variants, reflecting their dominance in recent research. This trend of using mostly decoder-only, GPT-based LLMs can also be observed in the broader software engineering context [11, 31]. However, there is also no consensus for a specific variant [11]. Fine-tuning was common for encoder models, whereas decoder models were primarily used via prompting (Section 3.3), with zero-shot prompting being the most frequent strategy (70 %). This also aligns with findings in the software testing context[31], where zero-shot prompting is also the most used strategy, followed by few-shot prompting. In the broader software engineering context, Hou *et al.* [11] found that few-shot prompting was the most commonly employed strategy, followed by zero-shot prompting. All surveys, including ours, show that advanced prompting techniques, like Chain-of-Thought and Retrieval-Augmented Generation, are only rarely used. Exploring whether these techniques can enhance approaches used for software architecture tasks is a question for future research.

Evaluation of Approaches. Evaluation methods were mainly technical experiments and benchmarking, with F_1-score being the most commonly used metric (Section 3.4). While most studies showed LLMs outperforming baselines, around one-third lacked comparative evaluation to a baseline. This indicates a need for more rigorous validation to demonstrate the added benefits of utilizing LLMs. However, nearly all studies provide supplementary material, enabling further insight into the approaches and results.

Future Work. Future research directions mentioned by the authors of the studies include testing different LLMs, refining input strategies, and integrating advanced techniques such as retrieval-augmented generation (RAG) and fine-tuning. These findings suggest that while LLMs offer significant potential for software architecture tasks and outperform baselines, it is a multi-dimensional problem to apply them in a way that ensures the best results.

The Future of LLMs in Software Architecture. Our findings indicate that the current body of published research on this topic is relatively limited. This is consistent with the review by Fan *et al.* [8] that characterized LLM-based design as an open research direction. Yet, there seems to be emerging research, as shown by the number of workshop publications and at this year's ICSA. One of the reasons for the comparatively low number of papers in software architecture as opposed to other software engineering disciplines could be that the capabilities of

LLMs were not sufficient to perform software architecture tasks until then: The three studies from before 2024 that utilize encoder-only models were not able to demonstrate consistent improvements of their approaches over the baselines [15, 16, 24]. This also illustrates the need for the continuous evaluation of both LLMs and proposed approaches for software architecture tasks: Given the fast-paced development of LLM technology, future research should consider strategies for ongoing assessment and adaptation of models in software architecture contexts.

6 Conclusion

In this paper, we present a systematic literature review on the usage of LLMs in software architecture and vice versa. Analyzing 18 relevant articles, we find that LLMs are mainly applied to tasks within Classification & Detection, Extraction & Generation, Assistants, and Reference Architecture, also showing how software architecture can be applied to LLMs. Most studies used decoder-only models, specifically GPT-based variants. While the number of articles covering software architecture and LLMs is increasing, the current body of research on this topic is relatively limited, and some areas, such as generating source code from architectural design, cloud-native computing and architecture, and checking conformance, are not well covered.

Most studies evaluate LLM-based approaches through technical experiments and benchmarking, often using performance metrics like the F_1-score. However, about one-third of the studies do not compare their results to a baseline, showing that stronger comparative evaluation is needed. Additionally, most studies use basic prompting techniques, while more advanced methods, such as Chain-of-Thought prompting and Retrieval-Augmented Generation, are rarely explored.

Our results suggest that future research should focus on improving LLM performance by refining input strategies, using more advanced prompting techniques, and regularly testing model capabilities. As LLMs continue to improve, their role in software architecture will likely grow, making continuous evaluation important to ensure their reliability and usefulness. As LLMs continue to evolve and more research emerges, this review should be repeated soon to keep up with new developments and trends.

7 Data Availability

We provide the complete data of this survey as supplementary material [29], i.e., the considered venues, the complete list of found articles, the filtered articles, and the classification of the examined articles.

Acknowledgments

This work was funded by Core Informatics at KIT (KiKIT) of the Helmholtz Assoc. (HGF), by KASTEL Security Research Labs, and the German Research Foundation (DFG) - SFB 1608 - 501798263.

References

1. Adnan, B. *et al.*: Leveraging LLMs for Dynamic IoT Systems Generation through Mixed-Initiative Interaction, (2025). arXiv: 2502.00689 [cs.SE]. https://arxiv.org/abs/2502.00689.
2. Ali, S., Arcaini, P., Arrieta, A.: Foundation Models for the Digital Twins Creation of Cyber-Physical Systems. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Application Areas, pp. 9–26. Springer Nature Switzerland, Cham (2025)
3. Arun, S., Tedla, M., Vaidhyanathan, K.: LLMs for Generation of Architectural Components: An Exploratory Empirical Study in the Serverless World. In: 22nd IEEE International Conference on Software Architecture (ICSA 2025). Institute of Electrical and Electronics Engineers (IEEE) (2025). https://arxiv.org/abs/2502.02539
4. Dhar, R., Vaidhyanathan, K., Varma, V.: Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical Study. In: 2024 IEEE 21st International Conference on Software Architecture (ICSA), pp. 79–89 (2024). https://doi.org/10.1109/ICSA59870.2024.00016
5. Díaz-Pace, J.A., Tommasel, A., Capilla, R.: Helping Novice Architects to Make Quality Design Decisions Using an LLM-Based Assistant. In: Software Architecture: 18th European Conference, ECSA 2024, Luxembourg City, Luxembourg, September 3–6, 2024, Proceedings, pp. 324–332. Springer-Verlag, Luxembourg City, Luxembourg (2024). https://doi.org/10.1007/978-3-031-70797-1_21
6. Dieste, O., Grimán, A., Juristo, N.: Developing search strategies for detecting relevant experiments. Empirical Software Engineering **14**, 513–539 (2009)
7. Donakanti, R. *et al.*: Reimagining Self-Adaptation in the Age of Large Language Models. In: 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pp. 171–174 (2024). https://doi.org/10.1109/ICSA-C63560.2024.00036
8. Fan, A. *et al.*: Large Language Models for Software Engineering: Survey and Open Problems. In: 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE), pp. 31–53 (2023). https://doi.org/10.1109/ICSE-FoSE59343.2023.00008
9. Fang, H. *et al.*: A Holistic Approach to Design Understanding Through Concept Explanation. IEEE Transactions on Software Engineering **51**(2), 449–465 (2025). https://doi.org/10.1109/TSE.2024.3522973
10. Fuchß, D. *et al.*: Enabling Architecture Traceability by LLM-based Architecture Component Name Extraction. In: 22nd IEEE International Conference on Software Architecture (ICSA 2025) (2025)
11. Hou, X. *et al.*: Large Language Models for Software Engineering: A Systematic Literature Review. **33**(8) (2024). https://doi.org/10.1145/3695988
12. Howard, J., Ruder, S.: Universal Language Model Fine-tuning for Text Classification. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 328–339. Association for Computational Linguistics, Melbourne, Australia (2018). https://doi.org/10.18653/v1/P18-1031. (Visited on 02/23/2022)
13. Jahić, J., Sami, A.: State of Practice: LLMs in Software Engineering and Software Architecture. In: 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pp. 311–318 (2024). https://doi.org/10.1109/ICSA-C63560.2024.00059

14. Johansson, N., Caporuscio, M., Olsson, T.: Mapping Source Code to Software Architecture by Leveraging Large Language Models. In: Software Architecture. ECSA 2024 Tracks and Workshops: Luxembourg City, Luxembourg, September 3–6, 2024, Proceedings, pp. 133–149. Springer-Verlag, Luxembourg City, Luxembourg (2024). https://doi.org/10.1007/978-3-031-71246-3_13
15. Keim, J. *et al.*: A Taxonomy for Design Decisions in Software Architecture Documentation. In: Batista, T. (ed.) Software Architecture. ECSA 2022 Tracks and Workshops, pp. 439–454. Springer International Publishing, Cham (2023)
16. Keim, J. *et al.*: Does BERT Understand Code? – An Exploratory Study on the Detection of Architectural Tactics in Code. In: Jansen, A. (ed.) Software Architecture, pp. 220–228. Springer International Publishing, Cham (2020)
17. Keim, J. *et al.*: Recovering Trace Links Between Software Documentation And Code. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. ICSE '24. Association for Computing Machinery, Lisbon, Portugal (2024). https://doi.org/10.1145/3597503.3639130
18. Kitchenham, B., Madeyski, L., Budgen, D.: SEGRESS: Software Engineering Guidelines for REporting Secondary Studies. IEEE Transactions on Software Engineering **49**(3), 1273–1298 (2023). https://doi.org/10.1109/TSE.2022.3174092
19. Kitchenham, B.A., Charters., S.: Guidelines for performing systematic literature reviews in software engineering. Tech. rep., Technical report, ver. 2.3 ebse technical report. ebse (2007). https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf
20. Konersmann, M. *et al.*: Evaluation Methods and Replicability of Software Architecture Research Objects. In: 2022 IEEE 19th International Conference on Software Architecture (ICSA), pp. 157–168 (2022). https://doi.org/10.1109/ICSA53651.2022.00023
21. Le, V.-H., Zhang, H.: Log Parsing: How Far Can ChatGPT Go? In: 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1699–1704 (2023). https://doi.org/10.1109/ASE56229.2023.00206
22. Lu, Q. *et al.*: Towards Responsible Generative AI: A Reference Architecture for Designing Foundation Model Based Agents. In: 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pp. 119–126 (2024). https://doi.org/10.1109/ICSA-C63560.2024.00028
23. Macías, A. *et al.*: Architecting Digital Twins Using a Domain-Driven Design-Based Approach*. In: 2023 IEEE 20th International Conference on Software Architecture (ICSA), pp. 153–163 (2023). https://doi.org/10.1109/ICSA56044.2023.00022
24. Mahadi, A., Ernst, N.A., Tongay, K.: Conclusion stability for natural language based mining of design discussions. Empirical Softw. Engg. **27**(1) (2022). https://doi.org/10.1007/s10664-021-10009-1
25. Quevedo, E. *et al.*: Evaluating ChatGPT's Proficiency in Understanding and Answering Microservice Architecture Queries Using Source Code Insights. SN Computer Science **5**(4) (2024). https://doi.org/10.1007/s42979-024-02664-0
26. Shamsujjoha, M. *et al.*: Swiss Cheese Model for AI Safety: A Taxonomy and Reference Architecture for Multi-Layered Guardrails of Foundation Model Based Agents. In: 22nd IEEE International Conference on Software Architecture (ICSA 2025). Institute of Electrical and Electronics Engineers (IEEE) (2025). https://arxiv.org/abs/2408.02205
27. Soliman, M.: Exploring Architectural Design Decisions in Mailing Lists and Their Traceability to Issue Trackers. In: Galster, M. (ed.) Software Architecture, pp. 307–323. Springer Nature Switzerland, Cham (2024)

28. Soliman, M., Keim, J.: Do Large Language Models Contain Software Architectural Knowledge? An Exploratory Case Study with GPT. In: 22nd IEEE International Conference on Software Architecture (ICSA 2025). Institute of Electrical and Electronics Engineers (IEEE) (2025)
29. Supplementary material to the SLR, https://doi.org/10.5281/zenodo.15475475.
30. Vaswani, A. *et al.*: Attention Is All You Need. In: Advances in Neural Information Processing Systems. Curran Associates, Inc. (2017). (Visited on 02/25/2022)
31. Wang, J. *et al.*: Software Testing With Large Language Models: Survey, Landscape, and Vision. IEEE Trans. Softw. Eng. **50**(4), 911–936 (2024). https://doi.org/10.1109/TSE.2024.3368208
32. Weber, I. *et al.*: FhGenie: A Custom, Confidentiality-Preserving Chat AI for Corporate and Scientific Use. In: 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pp. 26–31 (2024). https://doi.org/10.1109/ICSA-C63560.2024.00011
33. Wohlin, C. *et al.*: Experimentation in Software Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
34. Xiao, Y., Le, V.-H., Zhang, H.: Demonstration-Free: Towards More Practical Log Parsing with Large Language Models. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. ASE '24, pp. 153–165. Association for Computing Machinery, Sacramento, CA, USA (2024). https://doi.org/10.1145/3691620.3694994
35. Zan, D. *et al.*: Large Language Models Meet NL2Code: A Survey, (2023). arXiv: 2212.09420 [cs.SE]. https://arxiv.org/abs/2212.09420.
36. Zhang, N. *et al.*: Large Language Models for Explainable Decisions in Dynamic Digital Twins, (2024). arXiv: 2405.14411 [cs.AI]. https://arxiv.org/abs/2405.14411.
37. Zhao, J. *et al.*: DRMiner: Extracting Latent Design Rationale from Jira Issue Logs. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. ASE '24, pp. 468–480. Association for Computing Machinery, Sacramento, CA, USA (2024). https://doi.org/10.1145/3691620.3695019