http://www.diva-portal.org

This is the published version of a paper published in .

Citation for the original published paper (version of record):

Abbas, N., Shahzadi, N. (2026)
Large Language Models for Software Architecture Design Support in Self-Adaptive Systems: Early Insights from an Exploratory Systematic Review
WiPiEC Journal - Works in Progress in Embedded Computing Journal, 12(2): 1-8
https://doi.org/10.64552/wipiec.v12i2.149

Access to the published version may require subscription.

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-149341

Large Language Models for Software Architecture Design Support in Self-Adaptive Systems: Early Insights from an Exploratory Systematic Review

Nadeem Abbas
Linnaeus University, Växjö, Sweden
nadeem.abbas@lnu.se

Nazia Shahzadi
Linnaeus University, Växjö, Sweden
nazia.shahzadi@lnu.se

Abstract—**Modern computing systems exhibit increasing heterogeneity and often require runtime self-management and adaptation to cope with their structural and operational complexity, as well as changes in their environment and requirements. *Self-Adaptive Software Systems (SASS)* represent a class of context-aware and autonomous systems designed to manage such complexity. However, designing such systems remains challenging due to their complexity, runtime variability, and the continuous need to ensure functional and quality requirements. *Large Language Models (LLMs)* and *Generative AI (Gen AI)* offer promising capabilities, yet their use in the architectural design of SASS remains poorly understood. To that end, this study reports a *work in progress* systematic review. The review findings reveal that the use of LLMs and other Gen AI approaches for the architectural design of SASS remains nascent, with only *four* relevant studies identified. Across these studies, LLMs act as augmentative reasoning components, concentrated in the monitoring, analysis, planning, and knowledge phases of the MAPE-K loop and are only partially present in execution. Characteristics such as hybrid architectures, multi-agent reasoning, and retrieval-augmented grounding recur across the reviewed studies; however, given the small and heterogeneous evidence base, these are best viewed as preliminary observations rather than established trends, and trustworthiness and runtime assurance remain underexplored. As a work in progress, this paper contributes an initial characterization of LLM-supported design in self-adaptive systems, outlines research directions, and aims to stimulate discussion within the community on advancing LLM-supported architectural design for self-adaptive and autonomous software systems.**

Index Terms*—*Large Language Models, Self-Adaptive Software Systems, Software Architecture, Architectural Design, Architectural Reasoning, MAPE-K, Autonomous Computing Systems, Systematic Literature Review

I. INTRODUCTION

Modern computing systems increasingly operate at scale across the cloud–edge continuum, within rapidly evolving ecosystems with heterogeneous components, technologies, environments, and requirements. Their structural and operational complexity often exceeds what manual configuration or static control logic can manage, particularly when timely decisions must be made under strict quality-of-service constraints. These conditions place growing cognitive demands on software architects, who must balance quality attributes, evaluate alternatives, and reason over large design spaces under uncertainty [1], [2]. Architectural design[1] is inherently complex: architects must navigate combinatorial design spaces, evaluate multiple design options, and manage multi-objective trade-offs under given constraints. The architectural design becomes even more complicated in self-adaptive software systems and their product lines [3], [4].

Self-Adaptive Software Systems (SASS) are software systems capable of autonomously adapting their behavior and configuration at runtime in response to changes in requirements, operating environments, and system conditions [5]. The rapid emergence of agentic AI and increasing connectivity of Cyber-Physical Systems (CPS) introduces highly dynamic operating environments in which system goals, requirements, and execution contexts may evolve unpredictably during runtime. CPS tightly couple software with continuously changing physical environments through sensing, actuation, human interaction, and communication with other systems, resulting in persistent uncertainty and environmental variability. All such characteristics of modern software systems require self-adaptation as a fundamental architectural capability rather than an optional feature [6].

Consequently, software systems must be capable of continuously adapting both their functional behavior and quality attributes to maintain correctness, reliability, and performance under changing conditions. These evolving demands significantly increase the complexity of architectural design. Designing SASS[2] and their product lines requires architects to reason about multiple quality attributes and adaptation objectives while managing several dimensions of variability, including domain, runtime, and cross-domain variability [3], [4]. Collectively, these challenges substantially expand the architectural design space, making the architectural design of SASS an inherently complex task that demands sophisticated

[1]From this point onward, we use the term architectural design as an umbrella concept encompassing architectural analysis, reasoning, decision-making, and related activities performed during software architecture design.

[2] From this point onward, we use SASS to denote both individual self-adaptive systems and their product lines unless stated otherwise.

Manuscript received July 14, 2026; revised July 18, 2026; accepted July 17, 2026. Published August 25, 2026.
Issue category: Special Issue on DSD/SEAA 2026 on Works in Progress (WiP) Session, Krakow, Poland, Sept. 2026.
Paper category: Full
DOI: doi.org/10.64552/wipiec.v12i2.149

architectural reasoning and decision-making. Large Language Models (LLMs)[3] [7], as a form of generative and increasingly agentic AI, have emerged as a promising technology for augmenting architectural reasoning and decision-making, with growing evidence of their value across software engineering (SE) tasks [8], [9]. Their ability to interpret heterogeneous context, synthesize design alternatives, and explain rationale aligns closely with the reasoning-intensive activities that SASS demand [10], [11]. Yet how LLMs can systematically support architectural design, analysis, and reasoning specifically for SASS, as opposed to general software systems, remains poorly understood, motivating the review reported in this paper.

Gap: Despite growing use of LLMs and Gen AI for SE, little is known about how these can be used and effectively integrated into the architectural design of SASS, particularly for trade-off analysis and reasoning support required by architects of complex, large scale, and highly heterogeneous SASS.

Goal and Research Questions (RQs): This study aims to address the above gap by reporting an ongoing Systematic Literature Review (SLR) and early insights gained from the review. The review is guided by the following RQs:

- **RQ1:** *What has been reported to date for the use of LLMs and Gen AI for architectural design support in SASS?*

  Rationale: This question maps the state of the art by identifying existing studies on LLM and Gen AI-supported architectural design in SASS.

- **RQ2:** *What techniques are reported to optimize and evaluate LLMs and Gen AI for architectural design support in SASS?*

  Rationale: The support offered by LLMs and Gen AI is valuable only if results are accurate, grounded, and trustworthy; analyzing optimization and evaluation techniques reveals whether the reported LLM-based architectural design support methods meet these needs.

- **RQ3:** *What are the emerging trends in the use of LLMs and Gen AI for architectural design in the development of SASS?*

  Rationale: Understanding emerging trends helps anticipate innovations and guide SASS design tools and frameworks.

Guided by these RQs, the study yields the following contributions: **(1)** characterization of the role of LLMs in architecture design in early stage, and to our knowledge, the first SLR at the intersection of LLMs, architectural design, and SASS; **(2)** a consolidated view of the role of LLMs in the software architecture design process along the MAPE-K loop, spanning design-time and runtime activities; **(3)** identification and analysis of techniques reported to optimize and evaluate LLMs for architectural design support in the design of SASS; **(4)** a preliminary research agenda on how to use LLMs for architectural design support in SASS; and **(5)** the identification of 58 studies that target use of LLM for software architecture design for future analysis of their integration for architectural design support for SASS.

The early results show that the use of LLMs for architecture design support in SASS is very limited and largely underexplored. Recurring characteristics such as Retrieval-Augmented Generation (RAG), multi-agent orchestration, and feedback-driven refinement are observable across the selected studies, although the small and heterogeneous sample does not yet permit conclusions about broader trends in the field. Key challenges remain on the research agenda, including standardized evaluation, assurance, variability support, and architecture-centric integration across the MAPE-K loop.

The rest of the paper is organized as follows: Section II describes the research method; Section III reports results; Section IV synthesizes answers to the RQs and discusses implications; Section V examines threats to validity; Section VI describes what we plan for the future work; and Section VII concludes the study.

II. METHOD

The study goal and RQs, specified in Section I, require a review of published literature. We therefore used the SLR method and followed the guidelines of Kitchenham et al. [12], to systematically identify and analyze relevant studies. The review process is detailed in the following subsections.

A. Review Planning and Execution

To ensure methodological rigor and consistency, and reduce researcher bias, we developed a review protocol with well-defined RQs, search strategy, inclusion and exclusion criteria, quality assessment, and data extraction and analysis methods. We applied the protocol and used automatic search, manual screening, and snowballing [13] to conduct the review. We built a structured search string (Listing 1) using keywords derived from the RQs. We validated the search string using the Quasi–Gold Standard (QGS) [14] method to ensure effective retrieval and coverage of relevant studies. The validation using the QGS was done by constructing a set of representative studies and evaluating whether the search string successfully retrieved them. Details about the validation using QGS and the other review activities are documented in the protocol available in the replication package[4]. The studies used for QGS validation are listed in the file *MasterFile_LLMs_SA_SASS_Analysis.xlsx* in the worksheet *Quasi_Gold_Standard_Articles.*

The validated search string was then used for automatic search across three major digital libraries, namely ACM Digital Library, IEEE Xplore, and Scopus. The search targeted title and abstract fields and looked for studies published from January 1, 2018, to June 25, 2026[5].

[3] From this point onward, we use the term LLM to refer to both Large Language Models and Gen AI, unless specified otherwise.

[4] Replication Package: https://tinyurl.com/6jr3cd2x

[5] It is the time when Gen AI and modern LLMs started emerging.

LISTING 1: SEARCH STRING

("large language model" OR "large language models" OR LLM OR LLMs OR "generative AI" OR "generative Artificial Intelligence" OR "Gen AI" OR "Gen Artificial Intelligence")
AND
("software architecture" OR "architecture design" OR "architectural design" OR "architecture modeling" OR "architectural modeling" OR "architecture evaluation" OR "architecture analysis" OR "architectural reasoning" OR "architecture decision-making" OR "design rationale" OR "decision support" OR "architecture framework" OR "architecture pattern" OR "architecture tactic" OR "trade-off analysis" OR "quality attribute analysis" OR "design decision")
AND
(adapt* OR self* OR autonom*)

After the automatic search, the retrieved studies were screened manually with the inclusion and exclusion criteria (Table I) based on the title and abstract; if necessary, the full text was then screened. Then backward and forward snowballing was used to find other relevant studies in the screening. This process led to a final set of primary studies (Table IV) that was then systematically extracted and analysed as described below.

TABLE I: INCLUSION AND EXCLUSION CRITERIA (IC/EC)

Criteria	Description
IC1	The paper targets the use of LLMs for architectural design support for SASS.
Rationale:	This is the primary objective of the study as specified in Section I.
EC1	Non-peer-reviewed studies.
Rationale:	Non-peer-reviewed studies were excluded to ensure scientific rigor and reliability.
EC2	Non-English publications.
Rationale:	English is the primary language used by the authors.
EC3	Papers shorter than 5 pages.
Rationale:	Papers shorter than five pages were excluded because they often lack the architectural, methodological, and evaluation details required for reliable extraction of evidence addressing the review of questions.

B. Data Extraction and Analysis Method

After identifying the primary studies, both authors independently reviewed the full texts of all the included primary studies. To avoid inconsistencies in the data extraction process, a data extraction form comprising 18 data items was developed and used. Table II provides an overview of these data items, their purposes, and their mappings to the RQs. D1–D4 extract bibliographic information. D5 captures the quality score used to assess the reporting quality and rigor of the primary studies and to inform the interpretation and synthesis of the findings. It is measured using six quality items (Q1–Q6), listed in Table III.

For each primary study, each quality item was rated on a three-point scale {0, 1, 2}; 0 if it is a complete miss, 1 if it is partially satisfied, and 2 if it is completely satisfied. For instance, for the quality item Q1, a study receives a score of 0 if there is no problem definition, 1 if the problem definition is present but poorly specified, and 2 if there is an explicit problem definition. With a max score of 2 for each of the 6 quality items, the total quality score for each study thus ranges from 0 to 12.

TABLE II: DATA EXTRACTION ITEMS

ID	Item	Purpose (RQ)
D1–D4	Authors, year, title, venue	Documentation
D5	Quality score (Q1–Q6)	Quality
D6	LLM model used	RQ1
D7	Architectural activity supported	RQ1
D8	SASS lifecycle phase supported	RQ1
D9	Application domain	RQ1
D10	Role of LLM in architecture	RQ1
D11	Optimization technique	RQ2
D12	Evaluation method	RQ2
D13	Dataset / case study	RQ2
D14	Performance metrics	RQ2
D15	Reported limitations	RQ2
D16	Emerging trends	RQ3
D17	Research gaps	RQ3
D18	Future work	RQ3

The extracted data were then organized and analyzed using a combination of descriptive and qualitative content analysis techniques. Descriptive analysis was used to summarize study characteristics and identify patterns, while thematic analysis was employed to identify, compare, and group recurring concepts, themes, and findings relevant to the RQs. The data were analyzed qualitatively deductively with a coding frame that was a priori determined based on the RQ categories and the items to be extracted from the data (Table II): RQ1 was coded for D6-D10, RQ2 was coded for D11-D15, and RQ3 was coded for D16-D18. Rigor and individual coder bias were reduced by both authors independently coding all four major studies in this shared frame, then comparing assignment item by item and reaching consensus on the assignments.

TABLE III: QUALITY ITEMS USED TO ASSESS QUALITY OF THE REVIEWED STUDIES

Q1: Problem Definition	
No problem definition	0
General problem definition	1
Explicit problem definition	2
Q2: Problem Context	
No context description	0
General context description	1

Explicit context description	2
Q3: Research Design	
No description of how research was done	0
General description of how research was done	1
Explicit description of how research was done	2
Q4: Contributions	
No contributions description	0
General contributions description	1
Explicit contributions description	2
Q5: Insights	
No insights description	0
General insights description	1
Explicit insights description	2
Q6: Limitations	
No limitations description	0
General limitations description	1
Explicit limitations description	2

The coded data and supporting excerpts are available in the replication package, in the *Data_Extraction_Items* worksheet in the MasterFile_LLMs_SA_SASS_Analysis.xlsx file included in the replication package. After independently coding and analyzing the extracted data, both authors jointly reviewed their evidence, consolidated similar findings, and developed higher-level categories and themes. These themes were subsequently mapped to the RQs to provide a coherent and evidence-based synthesis of the literature. Throughout the process, an audit trail of extraction decisions, coding activities, and synthesis outcomes was maintained (see the replication package) to ensure transparency, consistency, and reproducibility of the review results.

III. Results

This section presents the outcomes of the search, study selection, and quality assessment, providing an overview of the available evidence on LLM-supported architectural design in SASS. The interpretation and analysis of these results are presented in Section IV.

A. Search and Selection Results

The automatic search yielded 1,183 studies (Scopus: 928, ACM DL: 47, and IEEE Xplore: 208). Applying filters to exclude non-English publications and items not published as peer-reviewed journal or conference articles reduced the set to 463 studies (Scopus: 226, ACM DL: 45, and IEEE Xplore: 192). After removal of duplicates, the dataset was further reduced to 363 studies. Subsequent screening using the inclusion and exclusion criteria (Table I) yielded only four primary studies (Table IV) that met all criteria. Neither backward nor forward snowballing identified any additional studies that met the inclusion criteria. The results of the search and selection process demonstrate that, despite the rapidly growing body of research on LLMs, only a very limited number of studies explicitly address the use of LLMs for architectural design in self-adaptive software systems.

The inclusion of only four primary studies constitutes a **major finding** of this review and reveals a substantial gap in the literature. The identified studies are solving the problem indirectly and are from different domains. In particular, only a handful of studies have explored the use of LLMs for supporting architectural design in SASS, indicating that the field is still at an early stage of development and remains largely underexplored.

TABLE IV: Selected primary studies (S1–S4) with bibliographic details.

ID	Bibliographic Details
S1 [15]	Yuchen Xia, Nasser Jazdi, and Michael Weyrich. An architecture for integrating large language models with digital twins and automation systems. In 2025 IEEE 30th International Conference ETFA, pp. 1–8, IEEE, 2025.
S2 [16]	Gianpaolo Ghiani, Emanuele Manni, and Sandro Zacchino. Improving adaptability in optimization-based decision support systems through large language models. IEEE Access, 2025.
S3 [17]	Çağatay Umut Öğdü, Kübra Arslanoğlu, and Mehmet Karaköse. An adaptive multi-agent LLM-based clinical decision support system integrating biomedical RAG and web intelligence. IEEE Access, 2025.
S4 [18]	Yangyang Zhuang, Wenjia Jiang, Jia-Yu Zhang, Ze Yang, Joey Tianyi Zhou, and Chi Zhang. Learning to be a doctor: Searching for effective medical agent architectures. Proc. 33rd ACM International Conference on Multimedia, pp. 6996–7005, 2025.

B. Quality Assessment Results

Before proceeding with the data extraction, analysis, and synthesis of the final four primary studies (Table IV), each study was subjected to a quality assessment to evaluate its methodological rigor and reporting quality. The assessment was conducted systematically by both authors, who independently evaluated each study against the predefined quality criteria and resolved any disagreements through discussion and consensus. The results of the quality assessment of each primary study are shown in Table V.

TABLE V: Quality Assessment Scores for Primary Studies.

Study	Q1	Q2	Q3	Q4	Q5	Q6	Total
S1	2	2	1	2	2	1	10/12
S2	2	2	2	2	2	2	12/12
S3	2	2	2	2	2	2	12/12
S4	2	2	2	2	2	0	10/12

Two studies (S2 and S3) reported at a high level (12) and the other two studies scored 10, suggesting good reporting quality. This implies that, although there are only a small number of studies of primary sources, the studies included are generally well reported. The scores should be interpreted as a sufficiently transparent basis for data extraction and analysis from the quality items, not as an assurance of methodological soundness since the quality items are more focused on reporting completeness. Each of the primary studies is assessed in detail, and justification of quality assessment scores is reported in *Reasons_for_Assigned_Quality_Scores.pdf*, which is part of the replication package, for transparency and reproducibility.

IV. Analysis And Discussion

This section analyzes and synthesizes the findings from the four primary studies (S1–S4). Given the limited number of studies, the focus is on identifying emerging patterns, architectural implications, and research directions rather than generalizable claims.

Inclusion Rationale: Since none of the primary studies use the term SASS explicitly, we document why each satisfies IC1. (S1) targets industrial automation systems that must adapt task execution at runtime: the LLM–digital-twin architecture supports replanning when exception events or execution failures are detected, making self-adaptation an explicit architectural capability. Its architectural design activity is twofold: the study contributes a three-layer reference architecture and three information-modeling design paradigms (design-time architectural design), while at runtime the LLM performs context interpretation and plan revision within that architecture, acting as a reasoning component inside a deliberately designed adaptive architecture rather than as a generic application service.

(S2) addresses an optimization-based decision support system whose hard-coded algorithms cannot follow evolving requirements; an LLM regenerate and verifies the code of decision-metric components and re-ranks candidate plans under new natural-language trade-off policies. This constitutes requirement-driven dynamic reconfiguration of system components, a canonical self-adaptation mechanism, in which the LLM directly performs the component redesign activity rather than application-level reasoning.

(S3) is an adaptive multi-agent clinical decision support system that monitors the confidence of its own outputs and, below a threshold, reconfigures its retrieval and query parameters through a bounded feedback loop, i.e., a MAPE-like control loop over its own reasoning pipeline (self-optimization). We note transparently that the LLM reasoning itself is largely application-level (diagnosis); the study is included because its contribution is the adaptive multi-agent architecture and the architecture-level design decisions it reports (agent decomposition, per-agent model selection, and loop and threshold design), making it a borderline but defensible case under IC1.

(S4) is the clearest instance of LLM-supported architectural design: the LLM analyzes diagnostic failures and applies node-, structural-, and framework-level modifications to the agent's workflow, performing automated architecture search and evolution. The managed element is the agent architecture itself, which adapts autonomously architectural self-adaptation in its strongest form, albeit directed at the agent system rather than an external operational process.

A. RQ1: Use of LLMs for Architectural Design Support in SASS

The four reviewed studies do not use LLMs as first-class architectural elements, but rather as augmentative reasoning components that operate alongside existing system components to assist in interpreting context, generating and refining design alternatives, supporting trade-off analysis, and providing explanations for architectural decisions. There is an obvious separation between design-time and run-time use. (S2) combines design-time reconfiguration with runtime decision support: the LLM generates and self-verifies decision-metric code when company policies change (design time) and selects among candidate plans according to natural-language trade-off preferences each time a new request arrives (runtime), thereby adapting an optimization-based decision support system without redesigning it. (S1) and (S3), in turn, operate primarily at runtime: in (S1), LLM agents interpret context from digital twins and produce control plans for industrial automation, while in (S3), a multi-agent clinical decision support system interprets patient context, retrieves biomedical evidence, and makes diagnostic decisions through a confidence-driven feedback loop.

(S4) is distinctive in that the LLM operates on the agent architecture itself: framed as an automated, AutoML-inspired search over multi-agent diagnostic workflows, it iteratively adds, removes, and modifies workflow nodes and control structures in response to diagnostic feedback. From an architectural perspective for SASS, the reviewed studies show that LLMs primarily support the monitoring, analysis, planning, and knowledge phases of the MAPE-K loop [5], contributing to context understanding, reasoning, and decision support, with execution-phase involvement appearing only partially, in (S4), where the LLM enacts changes to the agent architecture rather than to a managed operational system. All four studies rely on an explicit or implicit knowledge component, whether a digital twin (S1), an optimization model (S2), a biomedical retrieval store (S3), or a structured search space over agent workflows (S4).

Together, these observations indicate that LLMs in the reviewed studies primarily support architectural reasoning rather than being incorporated as explicit components within the architectural design of SASS. The MAPE-K coverage across the four studies is summarized in Table VI, and the justification of inclusion is given in Table VII. The coding rules for a study to be mapped to a MAPE-K phase were a study explicitly described as the functionality of LLM that fulfilled the rules of the phase.

Monitor: The LLM (or part of its chain of execution) repeatedly receives information at runtime about the managed system or its environment; A single user prompt is not sufficient.

TABLE VI: MAPE–K Phase Coverage Across the Identified Primary Studies

Study	Monitor	Analyze	Plan	Execute	Knowledge
S1	✓	✓	✓	×	✓
S2	×	✓	×	×	✓
S3	✓	✓	×	×	✓
S4	✓	✓	✓	partial	✓

Analyze: monitor context for deviation, diagnose cause, evaluate alternatives based on stated preferences, etc. producing a recommendation is not planning.

Plan: creation of an actionable adaptation plan or change set for the managed system, for example, control plans or workflow changes but not refinement of own queries or analyses.

Execute: The adaptation is implemented without the intervention of a human being; if the adaptation is implemented with a human intervention, then the enactment is coded as no execution support.

Knowledge: explicit shared knowledge components (models, indexes, repositories, etc.) consumed by phases of a loop.

With these rules, the outputs of S1 pass through human validation, and the outputs of S3 are handled by the system's own adaptive optimizer, which modifies the system's own queries, but S4 is written as partial Execute because validated modifications are applied automatically, but do not affect an external managed system they affect the agent architecture that is being managed. Evidence excerpts supporting each cell of Table VI are provided in the replication package.

B. RQ2: Optimization and Evaluation Techniques

The reviewed studies employ a range of techniques to improve LLM performance and reliability, including structured prompting and prompt templating, retrieval-augmented grounding, multi-agent decomposition, comprehension verification with self-debugging (S2), confidence-driven feedback loops (S3), and automated search over agent architectures (S4). These aim to mitigate well-known LLM limitations such as hallucination and lack of grounding; (S1) additionally relies on human validation and test-driven development, while (S3) constrains hallucination through biomedical retrieval and cross-checking against web evidence. From an architectural perspective, however, these techniques remain largely external to the software architecture, not explicitly integrated into architectural models or frameworks. Evaluation approaches are heterogeneous and domain specific, focusing on task performance rather than architectural quality attributes such as robustness, scalability, or adaptability.

(S3) and (S4) report task accuracy on domain benchmarks (medical question answering and dermatological image diagnosis, respectively), (S2) evaluates correctness, robustness to ill-defined prompts, and interpretability on a vehicle-routing case, and (S1) provides qualitative proof-of-concept case studies rather than benchmark metrics. Notably, even (S4), whose explicit goal is architecture search, evaluates the task accuracy, output consistency, and resource cost of the resulting workflows, but no architectural qualities such as adaptability or maintainability. The absence of architecture-aware evaluation limits comparability across studies and indicates a need for benchmarks and frameworks tailored to LLM-supported architectural design for SASS.

C. RQ3: Recurring Characteristics and Research Gaps

Despite the limited number of studies, several commonalities can be observed. Given the small and heterogeneous sample, we frame these as recurring characteristics of the selected studies rather than broader trends in the field, as it is difficult to distinguish a meaningful trend from an incidental property of one or two studies. The observed characteristics include the adoption of hybrid architectures based on using the LLM in conjunction with other components, such as digital twins (S1) and optimization engines (S2); the increased use of multi-agent reasoning, which is being automated in (S4) by searching for and evolving agent architectures; and the adoption of grounding mechanisms, such as retrieval augmentation and confidence-based feedback, to ensure reliability (S3). Meanwhile, there are still many areas of research that need to be addressed.

The first is the absence of architecture focused integration frameworks that treat LLM as explicit architectural components in SASS instead of as add-ons. Second, trustworthiness, explainability and assurance issues are not adequately investigated, either in the case of (S1), where the work is based on human validation; or in the case of (S3), where the level of confidence is set in a specific way; in both, there are no guarantees in the architecture. Third, there is no support for variability management, which is at the core of the SASS and their product lines. Last, but not least, there is a lack of a standardized and architecture-aware evaluation approach that hinders comparison and generalization. The identified gaps suggest that the research into using LLMs for architectural design in SASS is still in its exploratory stages, and there is a significant amount of work still to be done in order to create robust and architecture-centric approaches.

TABLE VII: CROSS-STUDY SYNTHESIS OF ARCHITECTURAL ROLES AND ADAPTIVE CHARACTERISTICS.

Study	Architecture	LLM Role	Adaptation Mechanism
S1 [15]	Layered Digital-Twin (DT)	Coordinator	DT-mediated adaptation; feedback; runtime
S2 [16]	Hybrid	Decision reasoner	Dynamic reconfiguration; runtime
S3 [17]	Multi-agent	Orchestrator	Feedback-driven optimization
S4 [18]	Graph-based	Architecture designer	Workflow evolution; iterative feedback

D. Synthesis and Implications

In the reviewed studies, LLMs play a role of more than just an AI service; they are responsible for coordinating components, orchestrating reasoning processes, integrating heterogeneous knowledge sources, and supporting architectural evolution. Overall, the results suggest that LLM are currently not fully embedded in SASS frameworks with a focus on architecture but are employed to enhance the human and system reasoning processes by providing context interpretation, decision support and explanations. The almost complete lack of some execution support (only partially present in (S4), where the LLM modifies and deploys the agent architecture), as well as architecture-based integration and variability management, further highlights the limited execution support and the need to strengthen it for SASS.

The two studies closest to treating architecture as a first-class concern were (S1), with its layered LLM–digital twin automation design approach, and (S4), with its automated architecture search approach; both hint at a possible direction of travel: embedding the LLM not just as a thinking resource, but as a co-creative design agent. This indicates that there is a need for new architectural models that explicitly integrate LLMs in the MAPE-K loop and tackle important issues like trustworthiness, adaptability, and scalability. The results must be viewed as preliminary insights into possible future research avenues and should serve as a starting point for more systematic, architecture-informed use of LLMs in SASS, given the limited amount of evidence.

V. Threats And Validity

This section outlines the main threats to the validity of the study, the measures taken to mitigate them, and the residual threats that remain despite these measures.

- **Search validity** concerns whether or not relevant studies could be obtained. The RQs were used to develop the search terms, which were narrowed down through iterative search and verified using the QGS method, and backward and forward snowballing was adopted to enhance the search coverage. Even with these efforts, there is a possibility of the residual threat of terminology bias: no terminology is yet standardized, and other related studies, such as studies of self-adaptive systems that are referred to by different terms (e.g., autonomic computing, agentic AI, dynamic reconfiguration, or specific self-* properties), or studies that do not use self-adaptive in the title or abstract, might not have been retrieved. The wide-ranging wildcard terms (adapt*, self*, autonom*) in the search string diminish, but do not completely remove, this risk.
- **Selection validity** addresses bias in the selection of studies (inclusion/exclusion) that is addressed by having two-step screening with pre-established inclusion/exclusion criteria with disagreements resolved by consensus. There remains a subjective issue about the relevance of the studies for architectural design support for SASS (IC1): When evaluating the relevance of studies for architectural design support for SASS, decisions are needed about these studies and whether they explicitly use the SASS terminology or address the issue of adaptivity only indirectly. Although independent screening and consensus minimize individual interpretation bias, other researchers may have interpreted the same borderline studies differently. In particular, S3 was the most contested inclusion decision; we retained it because its contribution is the adaptive multi-agent architecture rather than the diagnostic task itself, but other reviewers could reasonably have excluded it.
- **Data extraction and quality assessment validity** concerns the accuracy and interpretation of data extracted and quality scores; addressed by use of pre-established data extraction items that were related to the RQs, each author extracting and assessing independently, and discrepancies resolved by discussion. Even so, qualitative coding and quality scoring remain interpretive activities, and residual subjectivity in mapping four heterogeneous studies to a common coding frame cannot be fully excluded.
- **External validity** is a question of generalizability. The results are considered preliminary indications only, because there are only four primary studies, which reflects the **newness of the area**. This threat cannot be mitigated at this stage; instead, we address it by framing the findings as recurring characteristics of the selected studies rather than generalizable trends.

Further details are provided in the protocol document in the replication package.

VI. Future Works

Given the limited number of primary studies, we plan to extend the scope of the literature review as future work. Specifically, we aim to review studies from our search results that investigate the use of LLMs for software architectural design in general, rather than being limited to SASS. Using this extended scope, we have already screened the search results and identified 58 additional studies. These studies will be examined in future work to derive insights relevant to SASS. A complete list of these studies is given as a worksheet *LLMs_and_SoftwareArchitecture* in the file *MasterFile_LLMs_SA_SASS_Analysis.xlsx* available in the replication package. As a next step, we plan to analyze these studies to determine which LLM-based architectural design techniques can be transferred to or adapted for SASS, and to identify SASS-specific concerns, such as runtime adaptation, variability management, and assurance requirements, that may not be adequately addressed by approaches for general software architecture design.

A. Practical Implications

The results indicate that software architects could leverage LLMs for decision-making tasks during architectural analysis, alternative evaluation, and architectural documentation of SASS. Current LLM-based systems seem to perform best when they are used alongside expert reasoning, such as engaging in a process of exploratory design space exploration. To guarantee trustworthy and safe architectural choices, it is crucial that organizations implement mechanisms for verification, traceability, and human oversight when incorporating LLM into their architectural workflows.

VII. Conclusion

This paper shares insights from a work-in-progress systematic review on the use of LLMs in support of architectural design in SASS. The findings show that this research area is still in its infancy: among the four primary studies identified, the central observation is that the topic is not yet widely researched. So far, LLMs are used as augmentative reasoning components that support the interpretation of context, decision-making, and architectural rationale across the monitoring, analysis, planning,

and knowledge phases of the MAPE-K loop, with execution-phase involvement appearing only partially in a single study (S4). Their use as explicit architectural elements remains limited. Although the evidence base is small, the study contributes an initial characterization of LLM-supported architectural design in SASS, a first look at the different roles of LLMs across the MAPE-K loop, and an identification of key research gaps, namely architecture centric integration, standardized evaluation, and support for variability and runtime assurance. As future work, we plan to analyze the additional 58 studies on software architecture design beyond SASS to learn from the LLM-based architectural design practices, techniques, and methods they report. The goal is to identify approaches that can be transferred to or adapted for SASS and to support architects in addressing the complexity of SASS architectural design.

REFERENCES

[1] I. Mistrík, R. M. Soley, N. Ali, J. Grundy, and B. Tekinerdogan, Software quality assurance: in large scale and complex software-intensive systems. Morgan Kaufmann, 2015.

[2] W. Hasselbring, Software Architecture: Past, Present, Future. Cham: Springer International Publishing, 2018, pp. 169–184. [Online]. Available: https://doi.org/10.1007/978-3-319-73897-0_10

[3] N. Abbas and J. Andersson, "Architectural reasoning for dynamic software product lines," in Proceedings of the 17th International Software Product Line Conference Co-located Workshops, 2013, pp. 117–124.

[4] N. Abbas and J. Andersson, "Architectural reasoning support for product lines of self-adaptive software systems-a case study," in European Conference on Software Architecture. Springer, 2015, pp. 20–36.

[5] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," ACM transactions on autonomous and adaptive systems (TAAS), vol. 4, no. 2, pp. 1–42, 2009.

[6] D. Weyns, I. Gerostathopoulos, N. Abbas, J. Andersson, S. Biffl, P. Brada,T. Bures, A. Di Salle, M. Galster, P. Lago et al., "Self-adaptation in industry: A survey," ACM Transactions on Autonomous and Adaptive Systems, vol. 18, no. 2, pp. 1–44, 2023.

[7] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," ACM Transactions on Intelligent Systems and Technology, vol. 16, no. 5, pp. 1–72, 2025.

[8] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," ACM Transactions on Software Engineering and Methodology, vol. 33, no. 8, pp. 1–79, 2024.

[9] A. S. Shethiya, "Llm-powered architectures: Designing the next generation of intelligent software systems," Academia Nexus Journal, vol. 2, no. 1, 2023.

[10] J. Li, M. Zhang, N. Li, D. Weyns, Z. Jin, and K. Tei, "Exploring the potential of large language models in self-adaptive systems," in Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2024, pp. 77–83.

[11] R. Donakanti, P. Jain, S. Kulkarni, and K. Vaidhyanathan, "Reimagining self-adaptation in the age of large language models," in 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C). IEEE, 2024, pp. 171–174.

[12] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering– a systematic literature review," Information and software technology, vol. 51, no. 1, pp. 7–15, 2009.

[13] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, ser. EASE. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2601248.2601268

[14] H. Zhang and M. Ali Babar, "On searching relevant studies in software engineering," 2010.

[15] Y. Xia, N. Jazdi, and M. Weyrich, "An architecture for integrating large language models with digital twins and automation systems," in 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, 2025, pp. 1–8.

[16] G. Ghiani, E. Manni, and S. Zacchino, "Improving adaptability in optimization-based decision support systems through large language models," IEEE Access, 2025.

[17] Ç. U. Ögdü, K. Arslano ˘ glu, and M. Karaköse, "An adaptive multi-agent llm-based clinical decision support system integrating biomedical rag and web intelligence," IEEE Access, 2025.

[18] Y. Zhuang, W. Jiang, J.-Y. Zhang, Z. Yang, J. T. Zhou, and C. Zhang, "Learning to be a doctor: Searching for effective medical agent architectures," in Proceedings of the 33rd ACM International Conference on Multimedia, 2025, pp. 6996–7005.