

Can Agents Reconstruct Microservice Architecture?

Venla Liljas 

University of Southern Denmark, Vejle, Denmark

Matteo Esposito 

University of Oulu, Finland

Valentina Lenarduzzi 

University of Southern Denmark, Vejle, Denmark

Davide Taibi 

University of Southern Denmark, Vejle, Denmark

Abstract

Microservice architectures are increasingly becoming the standard design for cloud-native applications, yet their complex dependency structures make architectural understanding a challenge. Our research explores how Large Language Models (LLMs) can be leveraged to automatically reconstruct microservice architectures directly from source code, a task that requires both high-level architectural reasoning and fine-grained code understanding. We introduce a Minimum Viable Agent (MVA) as a single-agent that combines tool-assisted repository exploration with role-guided prompting to infer the services, inter-service connections, and exposed endpoints in Java microservice systems. We evaluate two state-of-the-art LLMs, GPT-4o and Llama4-16:17B, on 17 open-source microservice applications with 20 independent runs per system. GPT-4o achieves a mean F1 score of 0.604, outperforming Llama4 (F1 = 0.254). Both models show higher precision than recall, suggesting a tendency to under-represent architectural relationships rather than hallucinate them. Results also reveal substantial variation across applications and architectural element types. Analysis by architectural element indicates that service-to-service connections are the most challenging to recover for both models, while large performance differences emerge from endpoint reconstruction. Overall, our findings suggest that LLMs show promise for automated architecture recovery, but current capabilities remain insufficient for fully reliable reverse engineering. We discuss implications for integrating LLMs into architecture analysis workflows and outline directions for future research.

2012 ACM Subject Classification Software and its engineering → Software system structures; Software and its engineering → Software architectures; Computing methodologies → Natural language processing

Keywords and phrases Large Language Models, Microservice Architecture, Software Architecture Reconstruction, Agentic AI

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.62

Category Emerging Results, Vision & Reflection Track Paper

1 Introduction

Architectural recovery enables software engineers to understand, maintain, and evolve complex software systems. Documentation often becomes outdated, forcing developers to reconstruct system structure directly from implementation artifacts. This challenge is particularly pronounced in distributed systems, where the absence of an up-to-date architectural view contributes to architectural drift and erosion, increasing maintenance effort and reducing reliability [5, 16]. Microservice architectures amplify these difficulties: functionalities are distributed across independently deployable services that communicate through network-based interfaces [5, 7], with service boundaries, interactions, and endpoints scattered across



© Venla Liljas, Matteo Esposito, Valentina Lenarduzzi, and Davide Taibi;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;

Article No. 62; pp. 62:1–62:13



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

source code, configuration files, and infrastructure artifacts. Reconstructing a complete architectural view therefore requires synthesizing information from multiple heterogeneous sources, making manual reconstruction labour-intensive at scale [15].

Static analysis tools have been developed to automate this task through dependency analysis, call-graph extraction, and configuration parsing. While effective in certain contexts, these approaches remain limited for microservices, where service interactions are often defined through runtime configuration, service discovery or asynchronous messaging rather than explicit source-code dependencies. As a result, existing tools often produce incomplete or fragmented architectural views [3, 23]. Large Language Models (LLMs) have recently achieved promising results across a range of software engineering tasks, including code generation, defect localization, and automated issue resolution [1, 11, 28]. Their competence in reasoning over natural language and source code suggests potential for higher-level tasks requiring synthesis of diverse artifacts.

Agentic LLM systems capable of planning actions, interfacing with external tools, and revising outputs based on intermediate results [12] provide a practical mechanism for applying LLMs to such tasks, enabling iterative analysis that more closely resembles how human engineers investigate unfamiliar systems. Despite growing interest in applying LLMs to software architecture recovery, the application of LLM-based agents remains largely underexplored. Existing studies primarily combine LLMs with static analysis [17, 10], leaving the capabilities of standalone LLM agents largely unexamined. We base our single-agent design on the Minimum Viable Agent (MVA) principle, which has emerged from industry practice as a design philosophy for building the smallest agent capable of completing a target task^{1,2}. Accordingly, our agent consists of a single LLM reasoning engine and a minimal tool set for repository navigation, without external memory, a planner, or sub-agent coordination. We evaluate the design on the microSecEnD dataset [21] of open-source Java microservice applications. We examine the reconstruction accuracy of two state-of-the-art LLMs across multiple architectural element types. This provides initial evidence of the feasibility and limitations of LLMs as an MVA-based LLM agent for automated microservice architecture recovery.

To summarize, we contribute (i) an MVA design for architecture reconstruction based solely on iterative repository navigation and tool use, (ii) an empirical evaluation of 17 open-source Java microservice systems comparing a closed-weight model (GPT-4o) and an open-weight model (Llama 4), and (iii) a fine-grained analysis of reconstruction accuracy across architectural element types (components, connections, and endpoints), enabling comparison between LLM-only reconstruction and existing static-analysis approaches on the same benchmark.

2 Related Works

Fokaefs et al. [6] examine LLM-based architecture generation from natural language requirements, reporting that OpenAI and Gemini models reach benchmark accuracy levels while Llama performs the least successfully. Combining two models with split responsibilities for requirement interpretation and refinement improves accuracy, and adding context and reasoning further benefits complex systems. Their study, however, operates on documentation

¹ <https://www.agenticfoundry.ai/post/minimum-viable-autonomy-a-framework-for-building-agents-that-do-their-jobs>

² <https://simple.ai/p/an-agents-an-agent-no-matter-how-small>

and textual requirements without access to source code addressing architecture generation from requirements, whereas we investigate architecture recovery directly from implementation artifacts. Pan et al. [17] introduce ArchAgent, a framework that augments LLM reasoning with static analysis and adaptive code segmentation to derive multi-view architectural representations from large-scale legacy systems. In contrast, our approach relies solely on LLM-driven tool use via an MVA-style agent, without static analysis support. Hatahet et al. [10] propose a hybrid approach to software architecture description generation that combines reverse engineering with LLM analysis, recovering both static and behavioural views by passing source code directly to GPT-4o with few-shot, role-based prompting. A key limitation they identify is the model context window: when source code exceeds the input limit, recovery degrades, constraining applicability to larger systems. They identify agentic prompting as a promising future direction, a gap our study addresses by employing an MVA-style agent loop that navigates repositories incrementally through tool use, avoiding the need to pass the full codebase at once.

Siala & Lano [22] propose LLM4Models, applying LLMs to reverse engineering by abstracting Object Constraint Language (OCL) specifications from Java and Python source code. Using a fine-tuned Mistral-7B-v0.1, they report improved accuracy, consistency, and F1 over the base model, with Mistral outperforming Llama 2 13B on 20 evaluation examples. Their focus is on OCL extraction from individual programs, whereas our study targets distributed architectural elements spanning multiple services, connections, and endpoints across full repositories. In our study, the choice of GPT-4o reflects its established strong performance on instruction-following and structured output tasks [6], while Llama4-16×17B represents a capable open-source alternative deployable without API dependency, providing a practically relevant option for environments with strict data privacy policies or cost constraints.

Overall, existing studies focus on architecture generation from requirements rather than recovery from implementation artifacts [6], LLM-assisted reverse engineering of code or software models [22], and hybrid approaches that combine LLMs with static analysis [17, 10] for architecture recovery. Our work complements these directions by isolating the capabilities of a lightweight MVA-style LLM agent that relies solely on iterative tool use to recover microservice architectures from source code repositories. This enables us to assess the feasibility and limitations of LLM-only architecture recovery without static analysis as a confounding factor.

3 Empirical study design

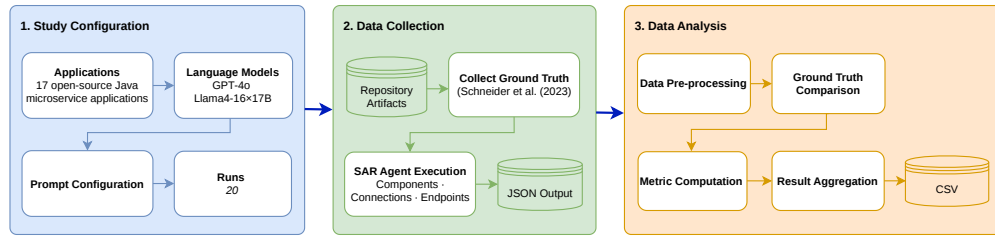
This section presents the empirical study design, including the study goal and research questions, the study context, the data collection methodology, and the data analysis approach [26]. An overview of the study design is described in Figure 1.

3.1 Goal and Research Questions

Our goal for this work is to analyze the use of LLMs in a single-agent workflow for reconstructing a microservice architecture, including the components, connections and endpoints with respect to their agreement and accuracy and limitations in the context of open-source Java based microservice architectures.

To achieve our goal, we formulate the following research questions (RQs):

RQ₁ How accurate LLMs are in reconstructing a microservice architecture?



■ **Figure 1** Study Design Overview.

Our research question aims to evaluate the ability of simple LLM agents to accurately and reliably reconstruct microservice software architectures. Accuracy metrics provide quantitative evidence of how well these models can identify true components, connections and endpoints within the architectures while minimizing falsely identified entities. The results from RQ₁ will provide insights into how LLMs, considering their probabilistic and semantic nature, can compare with or complement current predominantly deterministic methods consisting of static analysis tools while demonstrating the capabilities between the two LLMs for excelling in this task. We therefore formulate the following hypotheses:

- H_{01} : The choice of LLM **does not affect** the LLMs accuracy of reconstructing a microservice architecture.
- H_{11} : The choice of LLM significantly **affects** the LLMs accuracy for reconstructing a microservice architecture.

RQ₂ Does the architectural element type of the microservice architecture affect the LLMs accuracy for reconstructing a microservice architecture?

The complexity of large software systems may introduce cognitive biases in LLMs, potentially weakening their ability to perform precise reasoning when analyzing microservice architectures. This challenge also affects context handling, as models may introduce non-existent components or interactions when attempting to fill gaps in incomplete or ambiguous information. The risk of false inferences becomes particularly relevant given the different architectural elements in microservice architectures. Identifying high-level components, inter-service connections, and concrete API endpoints requires progressively finer-grained reasoning and deeper contextual integration. As the level of detail increases, the task may become more cognitively demanding for LLMs, which may influence reconstruction accuracy. To examine whether this granularity affects the performance of the LLMs, we formulate the following hypotheses:

- H_{02} : The architectural element type **does not affect** the LLMs accuracy of reconstructing a microservice architecture.
- H_{12} : The architectural element type significantly **affects** the LLMs accuracy for reconstructing a microservice architecture.

3.2 Study Context

Schneider et al. [21] introduced the microSecEnD dataset, which provides a collection of open-source microservice-systems represented as dataflow diagrams. The dataset consists of open-source Java microservice applications implemented using the Spring framework

and hosted on GitHub. For each application, the available artifacts include the source code, configuration files, and a manually validated reference architecture describing the services, their inter-service connections, and exposed endpoints. The selected systems vary in scale, ranging from 604 to 14,527 lines of code and comprising between 2 and 14 services. Architectural complexity, quantified by the total number of nodes and edges in the corresponding data flow diagrams (DFDs), spans from 29 to 246 elements across the dataset. Schneider et al. [20] used microSecEnD to benchmark state-of-the-art static analysis tools for microservice architecture recovery. Their evaluation reported an overall F1-score of 0.86 for the strongest individual tool, which increased to 0.91 when multiple tools were combined. We adopt microSecEnD as the primary dataset to investigate whether the same reconstruction task can be performed using an LLM-based agent.

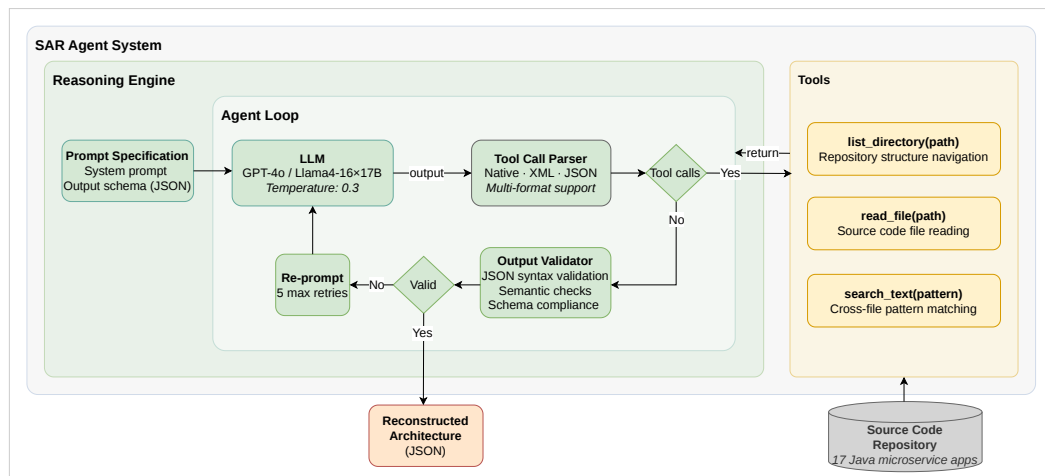
3.3 Agent Architecture

The architecture consists of a lightweight MVA agent workflow that enables the LLM to process the entire application directory and produce a single, consolidated output in a standardized JSON format. The agent system programmatically inspects the source code and associated artifacts of each microservice application. Its behavior is organized around a role-based prompt provided with a static analysis context.

- **Source Code Access:** The agent is provided with the full path to the application repository with the ability to reference all files, source code, configuration and when constructing the prompt ensuring that the LLM has complete architectural context.
- **Tools:** The LLM can request the use of three tools: one to list subdirectories and files in each repository, one to read the complete content of any file, and one to search for text patterns across all files in the repository.
- **Prompt Construction:** Based on the given prompting strategy, the LLM receives a common set of instructions that require it to report strictly from file evidence, avoid assumptions or prior knowledge, and comply with specific formatting rules when presenting its findings. We experimented with different prompt approaches and chose the best performing, single role-based prompting approach for subsequent analysis, as follows:
 - **Role-based:** The agent is assigned an expert role in static code analysis. The prompt depends on the domain-specific heuristics associated with that role, without using explicit task decomposition or step-by-step reasoning. To promote reliable results, the prompt instructs the LLM to report only those findings that can be directly supported by the source files, without requiring the supporting evidence to be included in the output.
- **Output:** The agent retrieves the LLM's response, parses the JSON, and stores both the architecture and the reasoning for subsequent evaluation. It repeats this workflow for each application and prompt strategy, supporting comparative analysis of reasoning quality and extraction accuracy. Invalid outputs trigger an iterative re-prompting loop.

3.4 Data Collection

To assess how well large language models can reconstruct microservice architectures, we experimented with various prompt engineering techniques across multiple LLM providers to assess their effectiveness in restoring the microservice architecture from 17 open-source Java repositories. We obtained all necessary code artifacts and ground-truth architecture specification for each application, enumerating all validated microservice components, their



■ **Figure 2** A minimum viable SAR agent. LLMs iteratively navigate the repository using the provided tools until a valid, structured architecture reconstruction is produced.

connections, and endpoints. We then executed our agent on each application, with a role-based prompt and two LLMs. GPT-4o was accessed via the OpenAI API alias, which resolved to snapshot gpt-4o-2024-08-06 during our runs (2026-03-14). Llama4-16×17B was served via OpenAI-compatible inference endpoints on Danish HPC infrastructure. Both models were queried with temperature = 0.3 and other decoding parameters at provider defaults. The agent analyzed the application directory and generated structured outputs that described every inferred component, connection, and endpoint.

3.5 Data Analysis

For each of the 17 open source microservice applications, we performed an automated comparison against the groundtruth architecture. We executed 25 runs per model to provide redundancy against transient API and network errors. We retained 20 runs per model for the analysis, yielding 340 observations per model (17 applications × 20 runs). Within the 20 runs, application-level observations that fail to produce a parseable extraction due to missing output, malformed JSON format or schema errors, are retained in the analysis and scored as zero rather than excluded. The analysis included the following steps:

- **Data pre-processing:** We parsed and normalized the data by converting both the LLM-generated outputs and the ground-truth specifications into a canonical representation, harmonizing naming conventions to enable a fair comparison.
- **Metric Computation:** For each application, we evaluated accuracy by computing standard information retrieval metrics consisting of precision, recall, and F1-score separately for components, connections, and endpoints.
- **Result Aggregation:** We consolidated the evaluation metrics across all applications, runs and LLM providers, and exported the aggregated results to a CSV file for subsequent analysis and visualization.
- **Error Analysis:** We investigated instances with low or zero metric values to uncover recurring failure patterns, investigating such outputs for inconsistent naming patterns or incomplete extractions.

LLM Accuracy for Microservice Reconstruction (RQ₁). For each subject system, LLM, architectural element type, and run, we compute precision, recall, and F1-score against the ground truth. We then compute the macro-average across the three element types for each run, yielding a system-averaged score per run that represents overall reconstruction accuracy. We report descriptive statistics (mean, median, standard deviation) on the system-averaged accuracy metrics per run. Following standard empirical software engineering practice [26], we test data normality using the Anderson–Darling test [2, 24]. If normality is not rejected, we apply a two-sided paired t-test with Cohen’s d_z as effect size; otherwise, we apply a two-sided Wilcoxon signed-rank test [25] with the rank-biserial correlation $r = (W^+ - W^-)/(W^+ + W^-)$ as effect size [13]. We conduct inferential tests on per-application mean scores aggregated across repeated executions, treating the application as the experimental unit. We set significance level $\alpha = 0.05$ and apply the procedure separately for each metric ($n = 17$ paired observations).

The Impact of Architectural Element Type on Accuracy (RQ₂). For each subject system, we compute precision, recall, and F1-score for three element categories (components, connections, endpoints), separately for each LLM and metric. Following the same procedure as RQ₁ [26], we assess normality using the Anderson–Darling test [2, 24]. If normality is not rejected, we apply repeated-measures ANOVA [9] with partial η^2 as omnibus effect size; otherwise, the Friedman test [8] with Kendall’s W . Following a significant omnibus test, for normally distributed data, we conduct post-hoc pairwise comparisons using parametric paired t-tests with Cohen’s d_z or the non-parametric Wilcoxon signed-rank tests with the rank-biserial correlation $r = (W^+ - W^-)/(W^+ + W^-)$ if normality is rejected [13]. To control the family-wise error rate across nine post-hoc comparisons (three element type pairs $\times$ three metrics), we apply a Bonferroni correction at $\alpha = 0.05$, yielding an adjusted threshold of $\alpha = 0.0056$. A pairwise difference is considered practically meaningful when significant after correction and reaching at least a medium effect size ($d_z \geq 0.5$ or $r \geq 0.3$) [4, 14]. Because the goal is comparative evaluation, we do not define an absolute threshold for acceptable reconstruction accuracy.

4 Results

This section presents the accuracy of GPT-4o and Llama4 in reconstructing microservice architectures from source code, and examines whether architectural element type influences the accuracy. Results are based on 17 subject systems executed across 20 repeated runs per model, reporting precision, recall, and F1-score against ground-truth annotations.

LLM Accuracy for Reconstructing a Microservice architecture (RQ₁). The Anderson–Darling test rejected normality for both LLMs across all metrics (Precision: $A^2 = 21.91$ for GPT-4o, 13.09 for Llama4; Recall: $A^2 = 3.19$ and 6.28; F1: $A^2 = 5.74$ and 4.60; all $p < 0.0001$), motivating the use of the non-parametric Wilcoxon signed-rank test. Across the experiments, GPT-4o exhibits substantially higher scores than Llama4 across all accuracy metrics. Notably, GPT-4o achieves high precision, indicating that the majority of extracted elements are correct, while its mean F1 score is more than twice that of Llama4, with substantial gaps in both precision and recall. The Wilcoxon signed-rank test confirms a statistically significant difference in favour of GPT-4o across all three metrics, indicating that GPT-4o outperformed Llama4 on every paired observation. Due to GPT-4o outperforming Llama4-16x17B on every one of the 17 applications for Precision, Recall and F1, the three

Wilcoxon signed-rank tests yield identical statistics ($S = -76.5$, $r = 1.000$, $p < 0.001$), interpreted as a single sign-test-like confirmation of complete dominance rather than as three independent statistical confirmations. Full descriptive statistics and Wilcoxon signed-rank test results are provided in Table 1.

■ **Table 1** Descriptive statistics on system-averaged scores ($n = 340$ per model) and Wilcoxon signed-rank test results on per-application aggregated means ($n = 17$ paired observations, two-sided, Bonferroni $\alpha = 0.017$).

Metric	GPT-4o			Llama4			Wilcoxon		
	Mean	Median	Std	Mean	Median	Std	S	p	r
Precision	0.846	0.915	0.181	0.452	0.583	0.292	-76.5	<0.001*	1.000
Recall	0.514	0.533	0.168	0.200	0.176	0.161	-76.5	<0.001*	1.000
F1	0.604	0.655	0.171	0.254	0.261	0.189	-76.5	<0.001*	1.000

* Significant after Bonferroni correction ($\alpha = 0.017$).

Architectural Element Type in Reconstructing a Microservice architecture (RQ₂). As the Anderson–Darling test rejected normality of the raw accuracy scores in every architectural element type–metric–model combination (A^2 ranging from 2.26 to 80.95, all $p < 0.0001$), we chose the non-parametric Friedman test for the statistical analysis. Described in Table 2, the Friedman test indicates a significant effect of architectural element type on reconstruction accuracy for both models. The effect is large for GPT-4o for F1 and Recall, and moderate for Llama4. Precision shows no significant architectural element type effect for GPT-4o.

■ **Table 2** Friedman test results for Precision, Recall, and F1 scores across architectural element types.

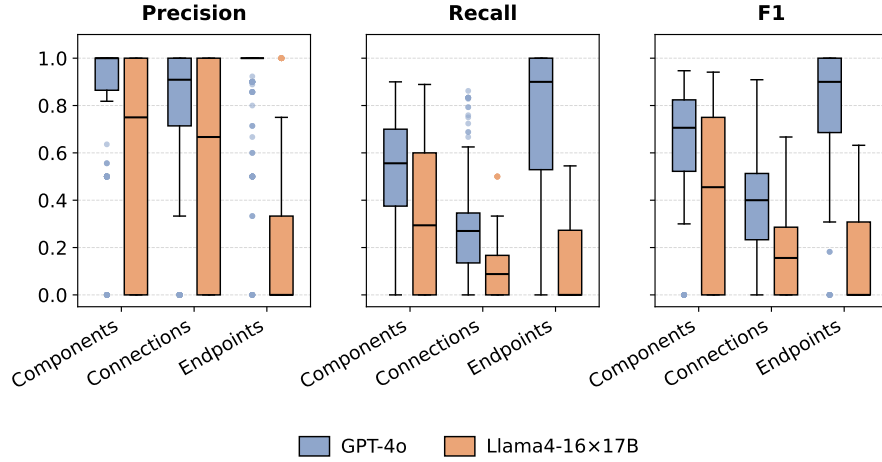
Model	Metric	χ^2	p-value	Kendall’s W
GPT-4o	Precision	2.362	0.307	0.069
	Recall	22.776	<0.001	0.670
	F1	22.776	<0.001	0.670
Llama4	Precision	9.556	0.008	0.281
	Recall	15.394	<0.001	0.453
	F1	15.636	<0.001	0.460

Post-hoc Wilcoxon signed-rank tests with Bonferroni correction reveal distinct patterns between models, highlighted in Table 3. For GPT-4o, connections perform significantly worse than both components and endpoints for F1 and Recall. No significant difference is observed between components and endpoints. For Llama4, components significantly outperform both connections and endpoints for F1 and Recall while connections and endpoints do not differ significantly. For Precision, none of the pairwise comparisons reach significance after correction. These results lead to rejection of H_{02} : architectural element type significantly affects reconstruction accuracy for both models, though the nature and magnitude of the effect differ between GPT-4o and Llama4. Figure 3 reveals additional distributional patterns. GPT-4o’s interquartile range for endpoint precision indicates high precision in the majority of runs rather than as isolated outliers. Llama4’s F1 distribution is bimodal, the model either performs moderately or fails completely, and for endpoints, more than half of all runs yield zero across all three metrics. Across both models, connections exhibit a precision-recall asymmetry, with high precision coexisting alongside consistently low recall.

■ **Table 3** Post-hoc Wilcoxon signed-rank results per architectural element pair ($n = 17$, Bonferroni $\alpha = 0.0056$).

Model	Metric		Component vs. Connection	Component vs. Endpoint	Connection vs. Endpoint
GPT-4o	F1	S	0	42	1
		p-value	<0.001*	0.109	<0.001*
		r	1.000	0.451	0.987
	Recall	S	0	32	3
		p-value	<0.001*	0.035	<0.001*
		r	1.000	0.582	0.961
Llama4	F1	S	0	13	63
		p-value	<0.001*	0.001*	0.548
		r	1.000	0.830	0.176
	Precision	S	29	19	24
		p-value	0.433	0.006	0.013
		r	0.256	0.752	0.686
	Recall	S	0	14	72
		p-value	<0.001*	0.002*	0.854
		r	1.000	0.817	0.059

* Significant after Bonferroni correction ($\alpha = 0.0056$).



■ **Figure 3** Distribution of Accuracy Metric by Architectural Element Type.

5 Discussions

Our research indicates that reconstruction accuracy is significantly affected by both the choice of LLM and the architectural element type being recovered. The two factors shape the outcome differently as the LLM choice determines the overall achievable level of accuracy while the element type shapes the pattern of strengths and weaknesses within both models. **The performance gap between GPT-4o and Llama4 is consistently large across all metrics and element types, indicating that model capability is a major factor of reconstruction quality.**

Beyond the difference in accuracy, the two models fail in qualitatively different ways. GPT-4o is consistently conservative by identifying fewer architectural element types than the ground truth but the elements it identifies are frequently correct. Llama4, by contrast, exhibits compound failure modes including inconsistent naming and malformed outputs. This distinction has relevance for downstream use. GPT-4o’s incompleteness can be addressed by augmenting the extracted architecture, but Llama4’s structural unreliability undermines the use of its outputs. Reconstruction accuracy also varies systematically across architectural element types. Endpoints, typically defined through REST annotations, are recovered most reliably, followed by components identifiable from class structure and naming conventions. Connections are the most difficult element type for both models, likely because they rely on indirect implementation details such as messaging middleware, gateway routing, or runtime configuration. However, GPT-4o and Llama4 differ in which element types they recover most reliably, suggesting that model capability also influences which architectural information can be extracted. Together, these findings indicate that **the limits of LLM-based SAR depend on both model capability and the observability of architectural information in static source code.**

Besides raw capability, **tool use behaviour can influence the observed differences.** Architectural reconstruction is fundamentally an exploratory task requiring iterative repository navigation [18, 19]. Llama4’s tendency to underuse available tools, requiring explicit re-prompting to elicit exploration [27], may explain its weak recall. Our observations suggest that equipping LLMs with reliable agentic exploration strategies can be a critical aspect of the task. Across both models and all element types, precision exceeds recall, often substantially, indicating that **the principal limitation lies in completeness rather than correctness.** This positions LLMs as suitable for complementing architecture recovery but insufficient as a standalone source of truth in our study configuration.

6 Threats to Validity

This section discusses the threats to validity explaining the different adopted tactics [26].

Construct Validity. We acknowledge that the reliance on LLMs for information extraction assessment introduces a potential construct validity concern. We operationalise reconstruction quality using Precision, Recall and F1 on components, connections and endpoints computed against a benchmark dataset of open-source Spring-based microservice systems with manually validated ground-truth annotations. This captures element-level extraction correctness but does not measure semantic aspects such as protocol correctness or the downstream usefulness of the reconstructed model. The ground-truth annotations themselves may be incomplete or inconsistent for individual applications.

Internal Validity. Each evaluated project is represented by a single canonical architectural reference. While this ensures analytical consistency, it may exclude alternative decompositions that would also be considered architecturally sound. Additionally, models may show preferences for specific tools, introducing evaluation bias.

External Validity. The study scope is limited to open-source microservice codebases written in Java and retrieved from GitHub. Although the selected projects span a range of sizes, application domains, and implementation styles, the findings may not transfer to proprietary systems, codebases developed in other programming languages, highly specialised domain contexts, or architectural patterns that fall outside those represented in our sample.

Conclusion Validity. Several factors may influence the reliability of the statistical conclusions. LLM outputs are inherently stochastic, which can lead to variation across repeated runs. To mitigate this, we used a low temperature setting (0.3), and performed multiple runs per application; however, minor variability may still remain. Tool execution may introduce additional noise when outputs fail to conform to the expected format. Finally, the reproducibility of the study may be affected by the evolution of external LLM APIs and open-source model versions over time.

7 Conclusions

We present an empirical evaluation of a simple MVA-based agent for reconstructing microservice architectures from source code, comparing the accuracy of a commercial and an open-source language model and the effect of architectural element type on the accuracy. GPT-4o substantially outperformed Llama4 across all accuracy metrics, with statistically significant differences for precision, recall, and F1. GPT-4o demonstrated high precision while recall remained moderate for both models, indicating that LLM-based reconstruction produces reliable but incomplete extractions. Architectural element type also had a statistically significant effect on reconstruction accuracy for both models. GPT-4o achieved its strongest performance on endpoints and moderate degradation on components and connections, whereas Llama4 achieved acceptable performance for components and degraded substantially on connections and endpoints. LLM-based reconstruction appears reliable for explicit, local architectural elements such as services and annotated endpoints, but less reliable for global structural relationships.

When compared with the strongest static-analysis approaches evaluated on the same microSecEnD benchmark [20], GPT-4o achieved a lower overall F1-score (0.604 versus 0.86 for the best individual tool and 0.91 for the combined approach). These results indicate that, although an LLM-only MVA can recover substantial architectural information, it does not yet match the completeness and overall accuracy of state-of-the-art static-analysis techniques. Specifically, the single-agent setup consistently under-performs on implicit inter-service dependencies and favors models with strong instruction-following capability. Future work should investigate agentic workflows with explicit task decomposition and self-refinement particularly to address the trade-off between completeness and reliability of architectural extraction.

Declaration on the use of generative AI

The authors used ChatGPT for suggestions on improving textual clarity. All research design, data analysis, interpretations, and manuscripts were created by the authors themselves.

Data Availability Statement

We provide all raw data, scripts, model prompts, and analysis configurations in a FAIR³ replication package on Zenodo⁴.

³ <https://www.go-fair.org/fair-principles/>

⁴ <https://doi.org/10.5281/zenodo.20445221>

References

- 1 Vivek Acharya. Generative ai and the transformation of software development practices, 2025. doi:10.48550/arXiv.2510.10819.
- 2 Theodore W. Anderson and Donald A. Darling. Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes. *The Annals of Mathematical Statistics*, 23(2):193–212, 1952. URL: <http://www.jstor.org/stable/2236446>.
- 3 Alexander Bakhtin, Xiaozhou Li, Jacopo Soldani, Antonio Brogi, Tomas Cerny, and Davide Taibi. Tools reconstructing microservice architecture: A systematic mapping study. In *Software Architecture – ECSA 2023 Tracks, Workshops, and Doctoral Symposium*, volume 14590 of *Lecture Notes in Computer Science*, pages 3–18, Cham, 2023. Springer. doi:10.1007/978-3-031-66326-0_1.
- 4 Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 2nd edition, 1988.
- 5 Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-67425-4_12.
- 6 Marios Fokaefs, Hashim Khan, and Bornha Ahmadzadeh. Can AI Build Systems? an Exploratory Study on Generating Software Architecture With LLMS . In *2025 IEEE International Conference on Collaborative Advances in Software and COmputiNg (CASCON)*, pages 1–10, Los Alamitos, CA, USA, November 2025. IEEE Computer Society. doi:10.1109/CASCON66301.2025.00069.
- 7 Martin Fowler and James Lewis. Microservices: a definition of this new architectural term, 2014. URL: <https://martinfowler.com/articles/microservices.html>.
- 8 Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937. doi:10.1080/01621459.1937.10503522.
- 9 Ellen R. Girden. *ANOVA: Repeated Measures*, volume 84 of *Quantitative Applications in the Social Sciences*. SAGE Publications, 1992.
- 10 Ahmad Hatahet, Christoph Knieke, and Andreas Rausch. Generating software architecture description from source code using reverse engineering and large language model. In *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 566–575. IEEE, October 2025. doi:10.1109/mode1s-c68889.2025.00080.
- 11 Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*, 33(8), December 2024. doi:10.1145/3695988.
- 12 Pengcheng Jiang, Jiacheng Lin, Zhiyi Shi, et al. Adaptation of agentic ai: A survey of post-training, memory, and skills, 2026. arXiv:2512.16301.
- 13 Dave Kerby. The simple difference formula: An approach to teaching nonparametric correlation. *Comprehensive Psychology*, 3, January 2014. doi:10.2466/11.IT.3.1.
- 14 Barbara Kitchenham, Tore Dybå, and M. Jorgensen. Evidence-based software engineering. In *Evidence-Based Software Engineering and Systematic Reviews*, pages 273–281, June 2004. doi:10.1109/ICSE.2004.1317449.
- 15 Genc Mazlami, Jürgen Cito, and Philipp Leitner. Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531, 2017. doi:10.1109/ICWS.2017.61.
- 16 Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2nd edition, 2021.

- 17 Rusheng Pan, Bingcheng Mao, Tianyi Ma, and Zhenhua Ling. Archagent: Scalable legacy software architecture recovery with llms. In *ICASSP 2026 - 2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 19357–19361, 2026. doi:10.1109/ICASSP55912.2026.11462552.
- 18 Yujia Qin, Shihao Liang, Yining Ye, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. doi:10.48550/arXiv.2307.16789.
- 19 Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*. Curran Associates Inc., 2023.
- 20 Simon Schneider, Alexander Bakhtin, Xiaozhou Li, Jacopo Soldani, Antonio Brogi, Tomas Cerny, Riccardo Scandariato, and Davide Taibi. Comparison of static analysis architecture recovery tools for microservice applications. *Empirical Softw. Engg.*, 30(5), June 2025. doi:10.1007/s10664-025-10686-2.
- 21 Simon Schneider, Tufan Özen, Michael Chen, and Riccardo Scandariato. microsecend: A dataset of security-enriched dataflow diagrams for microservice applications. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 125–129, 2023. doi:10.1109/MSR59073.2023.00030.
- 22 Hanan A. Siala and Kevin Lano. Towards using llms in the reverse engineering of software systems to object constraint language. In *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1–6, 2025. doi:10.1109/SANER64311.2025.00096.
- 23 Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215–232, 2018. doi:10.1016/j.jss.2018.09.082.
- 24 M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974.
- 25 Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.
- 26 Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering, Second Edition*. Springer, 2024. doi:10.1007/978-3-662-69306-3.
- 27 Bingxi Zhao, Lin Geng Foo, Ping Hu, Christian Theobalt, Hossein Rahmani, and Jun Liu. Llm-based agentic reasoning frameworks: A survey from methods to scenarios, 2025. doi:10.48550/arXiv.2508.17692.
- 28 Zibin Zheng, Kaiwen Ning, Qingyuan Zhong, Jiachi Chen, Wenqing Chen, Lianghong Guo, Weicheng Wang, and Yanlin Wang. Towards an understanding of large language models in software engineering tasks. *Empirical Softw. Engg.*, 30(2), December 2024. doi:10.1007/s10664-024-10602-0.