Structural Validation of LLM-Generated Microservice Decompositions Using Source-Code Dependencies

Daniel Anderson Silva
VIRTUS/UFCG
Federal University of Campina Grande (UFCG), Paraiba, Brazil
daniel.silva@virtus-cc.ufcg.edu.br

Renan Alves
Federal University of Campina Grande (UFCG)
Campina Grande, Paraíba, Brazil
jose.pereira@embedded.ufcg.edu.br

Emanuel Dantas
VIRTUS/UFCG
Federal University of Campina Grande (UFCG), Paraiba, Brazil
emanuel.filho@jaboatao.ifpe.edu.br

Ademar França de Sousa Neto
VIRTUS/UFCG
Federal University of Campina Grande (UFCG), Paraiba, Brazil
ademar.sousa@virtus.ufcg.edu.br

Mirko Perkusich
VIRTUS/UFCG
Federal University of Campina Grande (UFCG), Paraiba, Brazil
mirko@virtus.ufcg.edu.br

Danyllo Albuquerque
VIRTUS/UFCG
Federal University of Campina Grande (UFCG), Paraiba, Brazil
danyllo.albuquerque@virtus.ufcg.edu.br

Kyller Gorgônio
VIRTUS/UFCG
Federal University of Campina Grande (UFCG), Paraiba, Brazil
kyller@dee.ufcg.edu.br

Angelo Perkusich
VIRTUS/UFCG
Federal University of Campina Grande (UFCG), Paraiba, Brazil
perkusich@dee.ufcg.edu.br

Abstract

Decomposing monolithic systems into microservices is a key activity in software modernization. Although Large Language Models (LLMs) can generate semantically plausible decompositions from textual requirements, it remains unclear whether these proposals preserve the structural dependencies implemented in the source code. This paper evaluates the structural adherence of microservice decompositions generated by OpenAI o3 for the PetClinic and Bookstore systems. We propose an automated validation pipeline based on static dependency analysis and compare zero-shot and few-shot prompting using dependency preservation (TPD) and dependency violation (TVD) metrics. A robustness analysis was conducted to control for differences in class-to-service mapping coverage. After normalization, both prompting strategies produced equivalent structural adherence, achieving TPD values of 68.0% (PetClinic) and 83.3% (Bookstore). The findings demonstrate that structural evaluations of LLM-generated decompositions should explicitly control for mapping coverage, as apparent differences between prompting strategies may otherwise reflect methodological bias rather than genuine architectural quality.

Keywords

Large Language Models, Microservice Architecture, Monolith Decomposition, Static Analysis, Structural Validation, Reproducibility

1 Introduction

Modernizing legacy monolithic systems to microservice architectures depends on identifying boundaries that preserve high cohesion and low coupling—grouping related functionalities while avoiding the separation of components with strong structural dependencies. Without this care, migration can produce a distributed architecture more complex and fragile than the original monolith.

In recent years, Large Language Models (LLMs) have played an increasing role in Software Engineering activities, including code generation, documentation, refactoring, and architectural design support. Several studies have investigated using these models to generate microservice decompositions directly from textual requirements, obtaining promising results in service identification, with F_1-score values close to 0.97 [1, 2]. These results suggest that LLMs can understand domain semantics and propose conceptually coherent architectures, making them natural candidates to support software modernization.

However, when used solely with textual requirement descriptions, these approaches lack access to the structure actually implemented in the source code. As a consequence, a semantically plausible decomposition may separate components that exhibit strong structural coupling, creating inter-service dependencies that will require additional communication, refactoring, or integration mechanisms during migration. In other words, semantic quality does not necessarily imply structural feasibility. This distinction becomes critical when proposed architectures support real modernization decisions, where technical constraints and existing dependencies play a fundamental role.

Recent approaches incorporate structural information into the decomposition process [3, 4], but focus on generating decompositions from code. A complementary problem remains: how to structurally validate architectures already produced by LLMs from textual requirements.

From the perspective of Intelligent Software Engineering, this gap shifts the focus from merely generating architectures to validating architectural decisions produced by AI systems. Existing LLM-based approaches can suggest semantically plausible decompositions, but there is limited evidence on whether these decompositions respect the monolith's existing dependency structure. Therefore, instead of assuming that a conceptually coherent architecture is structurally feasible, it becomes necessary to confront AI-produced recommendations with objective evidence extracted from the source code. This validation step increases transparency,

helps architects understand the structural implications of generated decompositions, and contributes to making the use of LLMs in modernization tasks more auditable and reproducible.

In this context, this study proposes an automated and reproducible structural validation pipeline for LLM-generated microservice decompositions. The approach was applied to decompositions produced by OpenAI o3 for the PetClinic and Bookstore monolithic systems, using zero-shot and few-shot strategies. In addition to comparing strategies, a robustness analysis was conducted to control for differences in class-to-service mapping coverage, revealing that mapping coverage may bias structural comparisons between prompting strategies if left uncontrolled.

This study makes three contributions: (i) it identifies and mitigates a methodological bias caused by differences in class-to-service mapping coverage; (ii) it proposes an automated pipeline for structurally validating LLM-generated microservice decompositions using static analysis; and (iii) it introduces an experimental protocol based on TVD and TPD metrics to quantify structural adherence between generated decompositions and the original source code.

The remainder of this paper is organized as follows. Section 3 presents the study methodology, including the experimental design, the analyzed systems, the proposed pipeline, and the metrics used. Section 4 presents the experimental results and the robustness analysis. Section 5 discusses the main findings in light of the literature. Section 6 presents the implications for research and industry. Section 7 discusses threats to validity, and Section 8 concludes the work and presents directions for future research.

2 Background and Research Gap

Decomposing monoliths into microservices is a central challenge in the modernization of legacy systems. Classic works such as Newman [6] and Fowler [7] emphasize the importance of well-defined boundaries, low coupling, and high cohesion. However, identifying such boundaries requires understanding both domain responsibilities and the structural dependencies existing in the source code. Recent reviews, such as that of Abgaz et al. [10], show that the area still lacks public datasets, unified metrics, and comparable protocols to evaluate monolith decompositions.

LLMs can generate microservice architectures from textual requirements with high conceptual accuracy [1, 2, 5]. Recent work also characterizes structural properties of these architectures, noting that few-shot prompting tends to produce more fine-grained decompositions. However, these studies evaluate semantic coherence without systematically verifying whether the proposed boundaries preserve real source-code dependencies.

Another line of research incorporates structural information directly into the decomposition process [3, 4, 13, 18], reinforcing the importance of combining semantic and structural signals. However, these works focus on generating decompositions from code, whereas we investigate how to structurally validate decompositions already produced by LLMs from textual requirements.

The present study positions itself in this gap by proposing a structural validation pipeline attachable to any LLM-based architectural synthesis process. Instead of modifying the model or the prompt, the approach confronts the generated decompositions with a dependency graph extracted from the monolith, quantifying dependency violations and preservations through the TVD and TPD metrics.

3 Study Settings

Figure 1 summarizes the research methodology adopted in this study. The process is organized into four stages: (i) experimental design, (ii) definition of the experimental objects, (iii) execution of the structural validation pipeline, and (iv) evaluation of the generated decompositions through structural metrics and robustness analysis.

Stage 1: Experimental Design. We investigate whether prompting strategy influences structural adherence, using the same systems (PetClinic, Bookstore), LLM (OpenAI o3), and metrics (TVD, TPD) — differing only in prompting (zero-shot vs. few-shot).

The study is guided by the following research questions (RQs):

RQ1. What is the structural adherence of LLM-generated microservice decompositions with respect to the actual source-code dependencies?

RQ2. How do zero-shot and few-shot prompting affect the structural adherence of the generated decompositions?

RQ1 establishes a baseline by quantifying the structural adherence of LLM-generated decompositions with respect to the original source-code dependencies. This is important because semantically plausible decompositions may still conflict with the structural organization of the implemented system. *RQ2* investigates whether the adopted prompting strategy (zero-shot or few-shot) influences this structural adherence. Understanding this effect is important for determining whether improvements reported for prompting strategies also translate into structurally more consistent decompositions.

Stage 2: Experimental Objects. The evaluation considers two open-source Java monolithic systems selected because they differ in size and architectural complexity, allowing the proposed pipeline to be assessed on both a small benchmark and a larger application.

- *PetClinic*: 25 domain classes, widely used as a benchmark for monolith decomposition research.
- *Bookstore*: 93 domain classes, representing a larger e-commerce application with multiple business domains and authentication mechanisms.

The decompositions were generated by OpenAI o3 under zero-shot and few-shot strategies [1, 2, 5]. Dependency graphs were extracted with Tree-sitter and the Java grammar; the validation pipeline was implemented in Python (Tree-sitter, NetworkX, Pandas).

Stage 3: Structural Analysis. The proposed pipeline automatically validates LLM-generated microservice decompositions by comparing their service boundaries against structural dependencies extracted from the original monolith. The process comprises four sequential steps:

(1) *Static extraction.* The source code is parsed with Tree-sitter and the Java grammar to construct the dependency graph G_{real}, capturing inheritance, field and constructor injections, repository usage, and JPA annotations. The extraction is fully automated; a manual audit was performed on a small number of framework-mediated relationships to correct extraction errors.

(2) *Class-to-service mapping.* Each class is assigned to one of the proposed microservices using a heuristic based on package and class names, followed by manual refinement when no clear correspondence exists. The final mapping is stored as a

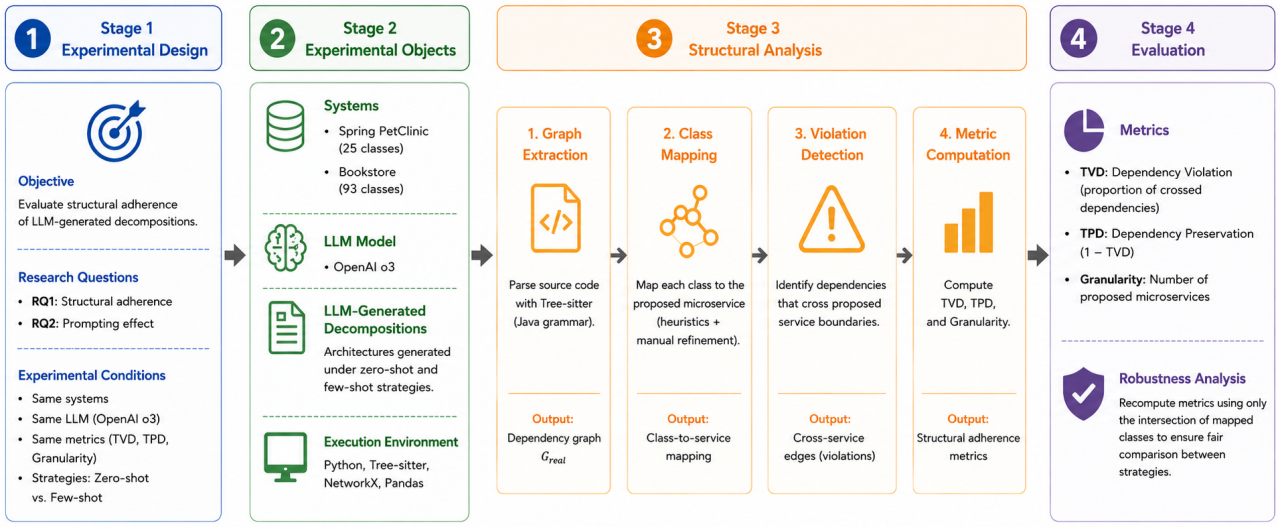

Figure 1: Overview of the proposed research methodology.

CSV file listing every domain class and its assigned service, making the process auditable and reproducible.

(3) *Violation detection.* The dependency graph is analyzed to identify edges crossing service boundaries, corresponding to structural violations introduced by the proposed decomposition.

(4) *Metric computation.* The detected violations are used to compute TVD, TPD, and decomposition granularity, providing a quantitative assessment of structural adherence.

The proposed pipeline produces a complete set of artifacts that make the structural validation process auditable and reproducible. These include dependency graphs (GraphML), class-to-service mappings (CSV), structured violation reports, and the computed TVD and TPD metrics. Together, these artifacts allow every reported result to be traced back to the corresponding source-code dependencies, facilitating independent verification and reproduction of the experiments.

The pipeline also highlighted two methodological aspects relevant to reproducibility. First, class-to-service mapping coverage directly influences the computed metrics and must therefore be controlled when comparing decompositions. Second, although dependency extraction is largely automated, manual auditing remained necessary for a small number of framework-mediated relationships (e.g., `Vet` → `Specialty`), reducing potential extraction errors.

Stage 4: Evaluation. The generated decompositions are evaluated using three complementary metrics that quantify their structural adherence to the original monolith.

- *TVD (Total Violated Dependencies):* proportion of extracted dependencies that cross proposed service boundaries. Higher values indicate greater structural divergence from the original monolith.
- *TPD (Total Preserved Dependencies):* complement of TVD ($TPD = 1 - TVD$). It represents the proportion of extracted dependencies preserved within the same proposed microservice.
- *Granularity:* total number of proposed microservices, reported as a descriptive characteristic of the generated decompositions rather than as a structural adherence metric.

Formally, let E denote the set of extracted dependencies and $s(c)$ the function that assigns class c to a microservice. A dependency $(c_i, c_j) \in E$ is considered preserved if $s(c_i) = s(c_j)$; otherwise, it is classified as a structural violation. Thus,

$$TVD = \frac{|\{(c_i, c_j) \in E \mid s(c_i) \neq s(c_j)\}|}{|E|} \qquad TPD = 1 - TVD$$

TVD and TPD quantify structural adherence rather than overall architectural quality. A dependency crossing a service boundary is not necessarily undesirable, as it may result from legitimate architectural decisions such as refactoring, API-based communication, or domain-driven redesign. Therefore, these metrics should be interpreted as indicators of the potential structural adaptation effort required by a proposed decomposition, complementing semantic evaluations of service boundaries.

Automation and manual effort. The dependency extraction (Step 1) and the metric computation (Step 4) are fully automated. The initial class-to-service mapping (Step 2) is performed automatically by a heuristic script, but manual verification and correction are required for a subset of classes whose assignment is ambiguous. Violation detection (Step 3) is automated once the mapping is finalized. No AI agents were employed during the structural validation itself; the LLM was used exclusively to generate the candidate decompositions in a prior stage, while the validation pipeline relies on deterministic static analysis and human-curated mappings.

4 Results

This section presents the results obtained from applying the pipeline to the PetClinic and Bookstore systems. Initially, the structural adherence of the model-generated decompositions is analyzed (RQ1). Next, the impact of the prompting strategy is investigated (RQ2), including a robustness analysis aimed at controlling coverage differences between the produced mappings.

4.1 Structural Adherence of Generated Decompositions (RQ1)

Table 1 presents the preliminary structural adherence results, computed before normalizing the class-to-service mapping coverage.

Table 1: Raw results – PetClinic and Bookstore with OpenAI o3.

Metric	PetClinic		Bookstore	
	Zero-shot	Few-shot	Zero-shot	Few-shot
Analyzed edges	40	40	203	230
Structural violations	9	8	34	52
TVD (Violation)	22.5%	20.0%	16.8%	22.6%
TPD (Preservation)	77.5%	80.0%	83.2%	77.4%
Granularity (# serv.)	8	8	6	7

The raw results show distinct behaviors between the two systems. In PetClinic, the difference between strategies was small: few-shot produced one fewer structural violation than zero-shot (TVD of 20.0% versus 22.5%). In Bookstore, however, zero-shot appeared to outperform few-shot, exhibiting 34 structural violations compared with 52. Because all experimental factors other than the prompting strategy were kept constant, these differences suggested the presence of an additional source of variation. This observation motivated a detailed inspection of the class-to-service mappings generated by each strategy.

Inspection of the mappings revealed that the observed differences did not stem from the structural organization of the decompositions, but from the class-to-service mapping coverage available for auditing. Because some monolith classes had not been assigned to a microservice in certain previously published decompositions, the metrics were initially computed over different sets of source-code dependencies. Under these conditions, the direct comparison between strategies introduces a methodological bias, as each is evaluated over a distinct universe of structural dependencies.

In Bookstore, zero-shot did not map the 13 classes responsible for JWT-based authentication, whereas few-shot grouped them into a dedicated authentication service. Since this subsystem concentrates several structural dependencies, excluding these classes removed 18 potential violations from the zero-shot evaluation, creating an apparent advantage that was unrelated to the prompting strategy itself. A similar phenomenon occurred in PetClinic, where the class `PetClinicApplication` was mapped only by zero-shot, introducing one exclusive structural violation.

To eliminate this bias, the metrics were recomputed considering only the intersection of the classes mapped by both strategies, ensuring that the comparison was performed over exactly the same set of source-code dependencies. The normalized results are presented in Table 2.

After normalization, the results became identical in both systems. In PetClinic, both strategies achieved a TVD of 32.0% and a TPD of 68.0%. In Bookstore, both obtained a TVD of 16.7% and a TPD of 83.3%. These findings indicate that the differences observed in the preliminary analysis were entirely explained by differences in mapping coverage. Once this source of bias was removed, zero-shot and few-shot exhibited equivalent structural adherence in both evaluated systems. More broadly, this result demonstrates that comparing structural metrics over different mapping coverages can lead to misleading conclusions regarding the effectiveness of prompting strategies.

Table 2: Results normalized by the intersection of class-to-service mappings.

Metric	PetClinic		Bookstore	
	Zero-shot	Few-shot	Zero-shot	Few-shot
Classes in intersection	15	15	68	68
Analyzed edges	25	25	203	203
Structural violations	**8**	**8**	**34**	**34**
TVD	**32.0%**	**32.0%**	**16.7%**	**16.7%**
TPD	**68.0%**	**68.0%**	**83.3%**	**83.3%**

Answer to RQ1 (Structural Adherence). After normalization, both strategies preserved a substantial but incomplete portion of the source code's dependencies. The preservation of 68.0% of dependencies in PetClinic and 83.3% in Bookstore indicates that the generated decompositions captured part of the systems' structural organization, while still leaving a non-negligible set of dependencies crossing the proposed service boundaries.

4.2 Impact of Prompting Strategy on Adherence (RQ2)

Although the normalized results indicate no quantitative differences between zero-shot and few-shot prompting, the violation reports generated by the pipeline provide qualitative insights into the types of structural inconsistencies introduced by both strategies. The analysis of structural violations identified by the pipeline revealed two recurring patterns. The first corresponds to the separation of entities related by JPA associations into distinct microservices. The second involves the displacement of controllers to services other than those containing the repositories used by those classes.

A representative example occurs in PetClinic, where `PetController` directly depends on `OwnerRepository`. In the analyzed decomposition, these classes were assigned to `Client Service` and `Pet Service`, respectively, causing this dependency to cross the boundary between microservices. This example illustrates a recurring structural violation pattern observed across the analyzed decompositions, rather than a limitation of a particular prompting strategy.

It is important to highlight that a structural violation does not, by itself, imply an incorrect architecture. It merely indicates that an existing dependency in the monolith would cross the proposed service boundary, requiring additional communication mechanisms, refactoring, or architectural reorganization during the migration process, which can increase coupling and consequently application latency. Thus, TVD should be interpreted as an indicator of potential structural adaptation effort, not as an absolute measure of architectural quality.

After controlling for the effect of mapping coverage, no differences were observed between the zero-shot and few-shot strategies in either of the analyzed systems. Moreover, both strategies exhibited the same categories of structural violations, suggesting that the

remaining inconsistencies are primarily associated with the limited structural information available from textual requirements rather than with the prompting strategy itself.

Answer to RQ2 (Impact of Prompting Strategy). After normalization by the intersection of mapped classes, zero-shot and few-shot produced structurally similar results in both analyzed systems. The differences observed in the raw metrics are fully explained by differences in class-to-service mapping coverage. Under the controlled conditions of this study, no evidence was found that one prompting strategy systematically outperforms the other in terms of structural adherence. However, this finding should be interpreted within the scope of the two systems and the single LLM evaluated.

5 Discussion

The results provide two main insights into the structural validation of LLM-generated microservice decompositions. First, the normalized TPD values (68.0% for PetClinic and 83.3% for Bookstore) indicate that LLMs preserve a substantial portion of the monolith's structural organization. However, the remaining dependency violations show that semantic plausibility alone does not guarantee structural adherence. These findings are consistent with previous studies demonstrating that LLMs can identify conceptually coherent services from textual requirements [1, 2, 5], while reinforcing evidence that structural information is essential for producing technically feasible decompositions [12].

From an architectural perspective, TVD should be interpreted as an estimate of the structural adaptation effort required during migration rather than as an indicator of poor architectural quality. Dependencies crossing service boundaries identify locations where architects may need to introduce APIs, refactor responsibilities, or redesign service boundaries. Consequently, the proposed metrics complement semantic evaluations by highlighting where migration effort is likely to concentrate.

Our findings also support hybrid approaches that combine semantic and structural information, such as MonoEmbed [3] and MicroDec [4]. The recurring violation patterns observed in this study, including the separation of strongly coupled entities and controllers from their associated repositories, suggest that these inconsistencies arise because textual requirements do not capture implementation dependencies. This observation reinforces the argument of Sellami [16] that combining multiple sources of architectural evidence is essential for structurally consistent decompositions.

A key methodological contribution is demonstrating that controlling for class-to-service mapping coverage eliminates spurious differences between prompting strategies. After normalization, zero-shot and few-shot yielded indistinguishable structural adherence, highlighting the risk of comparing raw metrics over different class subsets.

More broadly, the results suggest that the observed limitations are more closely related to the absence of structural information in the input than to the prompting strategy itself. Under the evaluated conditions, zero-shot and few-shot converged to equivalent structural behavior once mapping coverage was controlled. Therefore, future advances in LLM-assisted architectural synthesis are likely to depend less on prompt engineering and more on integrating semantic information from requirements with structural evidence extracted from source code.

6 Implications

The findings of this study extend beyond the evaluation of two prompting strategies. They provide methodological guidance for future empirical studies on LLM-assisted software architecture and practical insights for practitioners adopting LLMs in software modernization workflows. The implications span both research and industrial practice, highlighting how structural validation can complement semantic assessments and support more reliable architectural decision-making.

For research, structural adherence metrics such as TVD and TPD should complement semantic evaluation metrics in LLM-based architectural synthesis. Our robustness analysis demonstrates that comparisons between prompting strategies should report class-to-service mapping coverage and normalize evaluations whenever coverage differs. The need to manually correct framework-mediated dependencies (e.g., `Vet` → `Specialty`) indicates opportunities for hybrid extraction techniques.

For industrial practice, the proposed pipeline can serve as an automated architectural auditing mechanism. Practitioners can use the reported structural violations to identify coupling hotspots, estimate migration effort, and prioritize manual review. Structural validation can be incorporated into modernization workflows, providing objective evidence about the consequences of candidate decompositions.

7 Threats to Validity

The findings should be interpreted considering the limitations inherent to the adopted methodology and experimental setting.

External validity. The evaluation considered only two Java monolithic systems and a single LLM (OpenAI o3). Although the selected systems differ in size and architectural complexity, the findings may not generalize to other programming languages, application domains, industrial-scale systems, alternative architectural styles, or newer language models. Replications involving additional systems and LLMs are therefore needed to assess the broader applicability of the proposed pipeline and the observed methodological findings.

Internal validity. The structural metrics depend on correctly mapping source-code classes to the generated microservices. Although automated heuristics and manual inspection were employed, ambiguous mappings may affect the computed TVD and TPD values. Furthermore, the evaluated decompositions were obtained from previously published studies rather than generated within the present experimental protocol, which may introduce characteristics specific to those datasets. Differences in mapping coverage can also bias comparisons between prompting strategies; this threat was mitigated through the robustness analysis based on the intersection of mapped classes.

Construct validity. TVD and TPD quantify structural adherence rather than overall architectural quality. Dependencies crossing service boundaries may represent acceptable design decisions

after refactoring or API-based communication. Moreover, the proposed metrics do not capture other architectural quality attributes, such as scalability, maintainability, performance, fault isolation, or domain alignment. Finally, the static analysis does not identify runtime behaviors such as reflection, dynamic proxies, or framework-mediated interactions, although manual inspection was used to reduce extraction errors.

Conclusion validity. The evaluation relied on deterministic metrics and descriptive comparisons without statistical hypothesis testing. Consequently, the observed equivalence between zero-shot and few-shot prompting should be interpreted only for the analyzed systems and experimental conditions. Replications involving additional systems, prompting strategies, and LLMs are necessary to strengthen the generality of the conclusions.

8 Final Remarks

We investigated the structural adherence of LLM-generated microservice decompositions by proposing an automated validation pipeline based on static dependency analysis, class-to-service mapping, and TVD/TPD metrics. The methodology was applied to decompositions generated by OpenAI o3 for PetClinic and Bookstore under zero-shot and few-shot prompting.

The results showed that the generated decompositions preserved a substantial portion of the original dependency structure, achieving TPD values of 68.0% for PetClinic and 83.3% for Bookstore, while still introducing structural violations requiring architectural adaptation during migration. More importantly, the apparent superiority observed between prompting strategies disappeared after controlling for class-to-service mapping coverage. After normalization, zero-shot and few-shot exhibited comparable structural adherence within the scope of this study, demonstrating that differences initially attributed to prompting strategy were actually caused by methodological bias. Broader replications are needed to assess the generalizability of this finding.

These findings have implications for both research and practice. For research, they highlight the importance of complementing semantic evaluations with structural adherence metrics and explicitly controlling mapping coverage when comparing LLM-based architectural synthesis approaches. For practice, the proposed pipeline provides an auditable mechanism for validating AI-generated decompositions before migration decisions are made, helping architects identify structural inconsistencies and estimate adaptation effort based on objective evidence extracted from the source code.

Future work includes extending the evaluation to additional benchmark systems, such as MediaStore and TeaStore, and investigating other LLMs, prompting strategies, and decomposition approaches. Another promising direction is integrating the proposed validation pipeline into iterative architecture generation workflows, allowing structural feedback to guide successive generations toward more reliable and explainable AI-assisted software modernization.

Artifact Availability

A replication package is available at https://doi.org/10.5281/zenodo.20709439. It contains the complete analysis pipeline, input and intermediate datasets, generated results, and instructions required to reproduce the experiments.

Generative AI Use

Generative AI tools were used to support language editing and figure preparation. All scientific content was reviewed and approved by the authors, who take full responsibility for the manuscript.

Acknowledgments

This work has been partially funded by the project "ISOP Engineering: Métodos e Ferramentas de Engenharia Inteligente para Sensoriamento Inteligente" supported by CENTRO DE COMPETÊNCIA EMBRAPII VIRTUS EM HARDWARE INTELIGENTE PARA INDÚSTRIA - VIRTUS-CC, with financial resources from the PPI HardwareBR of the MCTI grant number 055/2023, signed with EMBRAPII

References

[1] Albuquerque, Danyllo *et al.* From Textual Requirements to Microservice Architectures: An Empirical Evaluation of LLM-Based Architectural Synthesis. arXiv preprint arXiv:2607.28307, 2026.

[2] Pereira, Jose Renan A. *et al.* Toward Generating Microservice Architectures from Textual Requirements with Large Language Models. In: *Anais do XIX Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS)*, p. 79–89, Porto Alegre, RS, 2025.

[3] Sellami, Khaled; Saied, Mohamed Aymen. MonoEmbed: Enhancing LLM representations for monolith to microservices decomposition through contrastive learning. *Empirical Software Engineering*, v. 31, n. 11, 2026.

[4] Alsayed, Ahmed Saeed; Dam, Hoa Khanh; Nguyen, Chau. MicroDec: Leveraging Large Language Models for Microservice Decomposition. *Journal of Systems and Software*, v. 203, 111622, 2024.

[5] Pereira, José Renan A. *et al.* Do LLMs Agree on Microservice Decompositions? A Multi–Model Study from Textual Requirements. In: *The 41st ACM/SIGAPP Symposium on Applied Computing (SAC '26)*, Thessaloniki, Grécia, 2026.

[6] Newman, Sam. *Building Microservices: Designing Fine-Grained Systems*. Sebastopol, CA: O'Reilly Media, 2015.

[7] Lewis, James; Fowler, Martin. Microservices: a definition of this new architectural term. ThoughtWorks, 2014. https://martinfowler.com/articles/microservices.html

[8] Krause, Alexander *et al.* Microservice Decomposition via Static and Dynamic Analysis of the Monolith. In: *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, p. 9–16, 2020.

[9] Matias, Tiago *et al.* Determining Microservice Boundaries: A Case Study Using Static and Dynamic Software Analysis. In: *European Conference on Software Architecture (ECSA)*, p. 315–332, 2020.

[10] Abgaz, Yalemisew *et al.* Decomposition of Monolith Applications into Microservices Architectures: A Systematic Review. *IEEE Transactions on Software Engineering*, v. 49, n. 8, p. 4213–4242, 2023.

[11] Bucaioni, Alessio *et al.* Artificial Intelligence for Software Architecture: Literature Review and the Road Ahead. In: *Proceedings of the ACM International Conference on the Foundations of Software Engineering (FSE '25)*, ACM, 2025.

[12] Ait Mansour, Nassima *et al.* Semantic Approaches to Microservice Identification: A Systematic Literature Review. *IEEE Access*, v. 13, 2025.

[13] Sooksatra, Korn *et al.* Using Static Analysis to Aid Monolith to Microservice System Transformation: Tuning Fuzzy c-Means in a VAE-Based GNN Approach. In: *39th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '24)*, p. 43–53, 2024.

[14] Saucedo, Ana Martínez *et al.* On the Variability of Microservice Decompositions: A Data-Driven Analysis. In: *2024 L Latin American Computer Conference (CLEI)*, p. 1–9, 2024.

[15] Saucedo, Ana Martínez *et al.* Exploring Alternative Microservice Decompositions using Data-driven Techniques and LLMs. *CLEI Electronic Journal*, v. 28, n. 3, paper 6, 2025.

[16] Sellami, Khaled. Automated Migration of Monolithic Systems into Microservices-based Architecture. 217 f. Tese (Doutorado em Informática) – Université Laval, Québec, Canadá, 2026.

[17] Weerasinghe, Mineth *et al.* From Monolith to Microservices: A Comparative Evaluation of Decomposition Frameworks. *arXiv preprint arXiv:2601.23141*, 2026.

[18] Alsayed, Ahmed Saeed; Dam, Hoa Khanh; Nguyen, Chau. MicroRec: Leveraging Large Language Models for Microservice Recommendation. In: *21st International Conference on Mining Software Repositories (MSR '24)*, ACM, 2024.