Designing and Evaluating an Agentic AI Pipeline for Microservice Architecture Generation from User Stories

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

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

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

Guillermo Rodríguez
ISISTAN/UNICEN
Tandil, Buenos Aires, Argentina
exarodriguez@gmail.com

Jorge Andrés Díaz-Pace
ISISTAN/UNICEN
Tandil, Buenos Aires, Argentina
adiazpace@gmail.com

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

Large Language Models (LLMs) have shown promising capabilities for generating software architectures directly from natural language requirements. However, most existing approaches rely on a single LLM, providing limited support for collaborative architectural reasoning and validation. This study investigates the feasibility of an Agentic AI workflow for generating microservice architectures from user stories. We propose a multi-agent pipeline comprising two software architect agents with complementary design perspectives and a validator agent responsible for consolidating candidate architectures through different collaboration strategies. The approach was evaluated on two benchmark systems (BookStore and PetClinic) by comparing the generated architectures against reference decompositions in terms of service identification and inter-service communication links. The proposed workflow achieved F1-scores of up to 1.00 for service identification and 0.88 for interaction recovery. Although the multi-agent workflow achieved performance comparable to a strong Few-Shot baseline, the results indicate that its main benefit lies in providing a structured and extensible architectural reasoning process through collaborative validation rather than consistently improving decomposition accuracy.

Keywords

Software Architecture, Agentic AI, Microservices, User Stories, Architectural Decomposition

1 Introduction

Software architecture design begins long before source code is produced [8]. During the early stages of software development, architects must translate high-level requirements—commonly expressed as user stories—into architectural decisions such as service boundaries, responsibilities, and communication patterns [18]. These decisions strongly influence software quality attributes, including maintainability, scalability, and evolvability, particularly in microservice-based systems [20].

Although user stories capture business goals and functional expectations, transforming them into coherent microservice architectures remains a knowledge-intensive activity [4]. Architects must identify domain concepts, group related functionalities, define service responsibilities, and establish communication relationships while balancing principles such as cohesion and loose coupling [17]. Consequently, architecture design still relies heavily on human expertise and iterative reasoning [3].

Existing automated approaches primarily rely on static and dynamic code analysis to recover architectural structures [1]. While effective for reverse engineering existing systems, these techniques are unsuitable during early design stages, where only textual artifacts (e.g., requirements, business rules, and user stories) are available [21]. Moreover, approaches based solely on structural information often fail to capture business intent and domain semantics [13].

Large Language Models (LLMs) have recently demonstrated remarkable capabilities for interpreting natural language and generating software artifacts [9, 24]. Previous studies have shown that LLMs can generate preliminary microservice architectures directly from textual requirements [5, 11, 15]. However, most existing approaches rely on a single LLM to perform the entire architectural reasoning process. This differs considerably from professional software architecture practice, where architectural decisions emerge through iterative analysis, exploration of alternatives, validation, and refinement.

Inspired by the collaborative nature of architectural design, Agentic AI enables multiple AI agents to cooperate while solving complex software engineering tasks [16]. Rather than delegating the entire reasoning process to a single model, specialized agents can contribute complementary architectural perspectives and collaboratively refine candidate solutions. Despite this potential, little empirical evidence exists regarding the use of collaborative AI workflows for transforming user stories into microservice architectures.

To address this gap, we propose a multi-agent pipeline for generating microservice architectures from user stories. The proposed workflow combines complementary architectural agents and a validation agent responsible for consolidating alternative architectural proposals through different collaboration strategies. By explicitly introducing collaboration and validation into the architectural reasoning process, we investigate whether these mechanisms can improve the quality of the generated architectures beyond what is achieved through a single LLM invocation. The approach is evaluated using

two benchmark systems with known reference decompositions [10], considering both service identification and inter-service communication recovery.

This paper offers three main contributions: (1) A multi-agent pipeline for generating microservice architectures from user stories through collaborative architectural reasoning; (2) An empirical evaluation of different agent collaboration strategies for consolidating architectural proposals; and (3) An experimental comparison between the proposed workflow and reference architectures using service identification and interaction recovery metrics.

The remainder of this paper is organized as follows: Section 2 presents foundational concepts and related work. Section 3 describes our experimental design and evaluation strategy. Section 4 reports results, whereas Section 5 discusses implications for research and practice. Section 6 outlines threats to validity. Finally, Section 7 concludes the paper and identifies research directions.

2 Fundamentals

This section provides the background for this study. First, we discuss the challenges of deriving microservice architectures from user stories. Next, we review recent research on LLM- and Agentic AI-based approaches for architecture generation.

From User Stories to Microservice Architectures. Deriving microservice architectures from user stories is challenging because architectural decisions are rarely explicit in textual requirements. Architects must identify domain concepts, define bounded contexts, assign responsibilities, and establish communication patterns while balancing cohesion, coupling, and business alignment [23].

Traditional approaches address this problem primarily through source-code analysis, using graph-based clustering [6], metrics-driven heuristics [14], or combined static and dynamic information [7]. While effective for reverse engineering, these approaches assume existing source code and overlook business knowledge contained in user stories and requirements [12, 19].

Related Work and Research Gap. The use of Large Language Models (LLMs) for software architecture design has attracted increasing attention in recent years. Hou et al. [9] identified architecture design as an emerging application area for LLMs within software engineering, while Dhar et al. [5] investigated their potential to support architectural decision-making by generating design alternatives from textual specifications. Pereira et al. [15] advanced this line of research by demonstrating that LLMs can generate complete microservice architectures directly from textual requirements. Their empirical evaluation showed that few-shot prompting improves the identification of services and inter-service interactions, providing evidence that LLMs can support early architectural design from natural language artifacts.

Building upon these advances, recent work has started to investigate Agentic AI as a means of structuring the architectural reasoning process. Calamo et al. [2] proposed ArchiLLM, a multi-agent framework that transforms user stories into microservice architectures and subsequently generates application code. Their workflow integrates specialized agents for microservice extraction, retrieval-augmented generation (RAG), architecture generation, and code generation, demonstrating the feasibility of end-to-end AI-assisted software development from user stories. This work represents an important step beyond single-LLM approaches by introducing collaboration among specialized agents during architecture generation.

Despite these advances, important research questions remain open. Existing studies investigate whether AI can generate architectures from textual artifacts or support end-to-end software generation. However, little empirical evidence exists regarding how different collaboration strategies among AI agents influence the architectural reasoning process itself. In particular, the effects of alternative mechanisms for validating and consolidating architectural proposals remain largely unexplored. This work addresses this gap by evaluating a collaborative multi-agent pipeline focused on microservice architecture generation from user stories, investigating how different consolidation strategies affect the identification of services and inter-service interactions. Unlike previous single-LLM approaches that delegate the entire reasoning process to one model, our workflow distributes architectural analysis across specialized agents and explicit validation steps. Compared with ArchiLLM, which targets full code generation, our pipeline focuses solely on the architectural decomposition and interaction mapping, enabling a more detailed evaluation of collaborative design strategies.

The present pipeline was developed independently of ArchiLLM [2]. While ArchiLLM targets full code generation, our study isolates architecture generation to examine collaboration and validation strategies; the two approaches are complementary.

3 Study Settings

This study investigates the feasibility of employing an Agentic AI pipeline to generate microservice architectures directly from textual requirement descriptions. The proposed workflow combines specialized software architect agents and a validator agent to collaboratively derive architectural proposals from user stories. The effectiveness of the generated architectures is evaluated through the following research questions (RQs):

- *RQ1*: Can an Agentic AI pipeline identify the individual microservice components from textual requirement descriptions?
- *RQ2*: How accurately can an Agentic AI pipeline recover the inter-service communication links present in the reference architecture?

RQ1 and RQ2 address two complementary aspects of microservice architecture generation. The first focuses on identifying the appropriate microservice components from textual requirements, whereas the second evaluates whether the proposed pipeline can correctly establish the communication links among these services. Since a microservice architecture is defined by both its decomposition and the interactions among its components, analyzing these two dimensions provides a more complete assessment of the generated architectures. To address the proposed RQs, we conducted the study following the four-step workflow illustrated in Figure 1. The workflow comprises four steps as described in what follows.

Step 1: Dataset Preparation. We employed two benchmark systems from the dataset curated by Imranur et al. [10]: *BookStore*, an e-commerce application, and *Spring PetClinic*, a veterinary management system. These systems were selected because they provide both textual requirements and reference microservice architectures, enabling a direct comparison between the generated and expected architectural decompositions.

The user stories were minimally edited to remove redundancies and standardize terminology without modifying their functional meaning. The reference architectures were extracted from the `.dot` files provided in the original replication package and used as the ground truth throughout the evaluation. Each system comprises

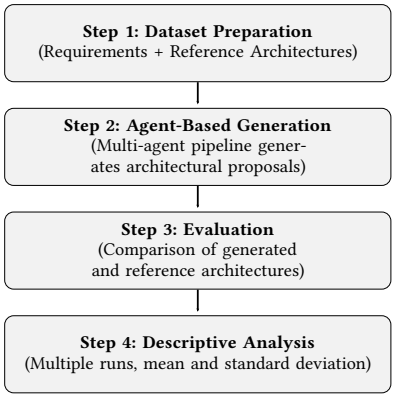

Figure 1: Methodological workflow of the experiment.

seven reference microservices, with 19 interaction links for PetClinic and 9 for BookStore.

Step 2: Agent-Based Architecture Generation. The proposed workflow comprises three specialized AI agents implemented using *CrewAI* and *LangChain*, with `gemini-pro-latest` (Google Gemini) as the underlying language model. Three agents were adopted as the minimal configuration capable of supporting proposal generation from complementary perspectives followed by explicit architectural validation. As illustrated in Figure 2, two software architect agents independently generate candidate microservice architectures from the input user stories, while a validator agent consolidates both proposals into a single architecture. The workflow was designed to emulate collaborative architectural reasoning by combining complementary design perspectives.

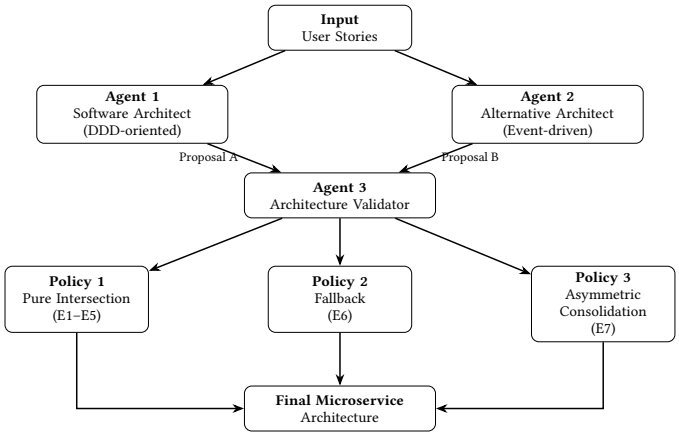

Figure 2: Overview of the proposed Agentic AI pipeline.

Agent 1 - Software Architect. Agent 1 follows a conservative decomposition strategy inspired by Domain-Driven Design (DDD). Its objective is to identify only the microservices explicitly supported by the user stories, emphasizing precise service boundaries while avoiding the introduction of infrastructure components that are not required by the requirements.

Agent 2 - Alternative Architect. Agent 2 adopts a complementary architectural perspective by exploring event-driven communication and commonly adopted microservice infrastructure elements, such as API gateways and asynchronous messaging. Rather than replacing the conservative decomposition, this agent aims to produce alternative architectural proposals that may complement those generated by Agent 1.

Agent 3 - Architecture Validator. The validator receives both candidate architectures and consolidates them into a single solution. Three consolidation policies were investigated, as detailed below.

The three policies represent progressively less restrictive approaches to consensus. *Pure Intersection* retains only elements agreed upon by both architect agents. *Fallback* preserves Agent 1's entire proposal if the intersection yields fewer than three services. *Asymmetric Consolidation* uses Agent 1 as the baseline and adds contributions from Agent 2 only when they are clearly grounded in the requirements and do not introduce unjustified complexity; it was adopted as the default policy.

All agents generate a structured CSV representation containing the identified microservices, their responsibilities, and the corresponding communication links. To reduce variability across executions, both architect agents receive the same few-shot architectural example as part of their prompts.

Step 3: Evaluation. The consolidated architectures generated by the proposed workflow were evaluated against the reference architectures using *Precision*, *Recall*, and *F1-score*. These metrics were computed independently for service identification (RQ1) and inter-service communication links (RQ2).

For *RQ1*, the name of each generated microservice was normalized using a predefined synonym dictionary (e.g., `EurekaServer` → `discovery-server`, `AuthenticationService` → `auth-service`) to account for semantically equivalent naming variations. After normalization, true positives, false positives, and false negatives were computed by comparing the generated services with the reference architecture. For *RQ2*, communication links were extracted as pairs of interacting microservices from the generated CSV files and compared with the reference architecture. An undirected comparison criterion was adopted, meaning that the order of the services within each pair was ignored to avoid penalizing equivalent architectural representations that differed only in interaction direction.

Step 4: Descriptive Analysis. The evaluation comprised five independent executions using the original consolidation policy (Pure Intersection), allowing the variability of the baseline workflow to be characterized through the mean and standard deviation of Precision, Recall, and F1-score. After analyzing the initial results, two additional experiments investigated alternative consolidation strategies. Experiment E6 evaluated the Fallback policy, whereas Experiment E7 evaluated Asymmetric Consolidation. Both were conceived as part of the iterative refinement of the pipeline: E6 was a transitional improvement over pure intersection, while E7 was subsequently adopted as the definitive policy and integrated into the final pipeline design. Given this exploratory and constructive role, E6 and E7 are reported as single-run analyses, consistent with their purpose of guiding architectural decisions in the pipeline rather than providing statistically replicated comparisons.

4 Results and Discussion

All experiments were conducted in July 2026 using the `gemini-pro-latest` model (Google Gemini) with temperature set to 0.3. The proposed workflow was evaluated on the PetClinic and BookStore systems to assess both service identification (RQ1) and inter-service communication links (RQ2). Throughout this section, *A1* denotes the architecture generated by Agent 1 (Software Architect), *A2* denotes the architecture generated by Agent 2 (Alternative Architect), and *Cons.* refers to the consolidated architecture

produced by Agent 3 (Architecture Validator). The complete architectural outputs generated in every experiment, including the CSV files used in the evaluation, are available in the replication package (see the *Artifact Availability* section).

Service Identification (RQ1). Table 1 summarizes the F1-score obtained for service identification across the seven experiments. Overall, Agent 1 (A1) achieved high performance in both systems, with mean F1-scores of 0.96 ($\sigma = 0.04$) for PetClinic and 0.94 ($\sigma = 0.08$) for BookStore across Experiments E1–E5. The main source of precision loss was the recurring generation of the `pets-service` microservice for PetClinic, which is absent from the reference architecture despite being architecturally plausible.

Table 1: F1-score for service identification.

Exp.	PetClinic			Bookstore		
	A1	A2	Cons.	A1	A2	Cons.
E1	0.93	0.93	0.93	0.86	0.11	0.86
E2	0.93	0.59	0.93	0.86	0.00	0.86
E3	1.00	0.47	1.00	1.00	0.56	1.00
E4	0.93	0.47	0.93	1.00	0.10	1.00
E5	1.00	0.38	1.00	1.00	0.00	1.00
E6	0.93	0.88	0.93	1.00	0.33	1.00
E7	0.93	0.88	0.93	1.00	0.63	1.00

Agent 2 (A2) exhibited considerably greater variability. While it occasionally achieved competitive results (e.g., F1 = 0.93 in PetClinic, E1), its exploratory architectural strategy frequently introduced non-canonical service names or additional infrastructure components, leading to substantial performance degradation in some runs, particularly for BookStore (F1 = 0.00 in E2 and E5).

Although Agent 2 did not outperform Agent 1 in terms of accuracy, its role in the pipeline should not be interpreted solely through quantitative metrics. Agent 2 was designed to explore architectural alternatives that a strictly conservative approach might overlook. In several experiments, it proposed services or communication links that, while not present in the reference architecture, were architecturally plausible. More importantly, when its proposals aligned with those of Agent 1, the consensus reinforced the confidence in those decisions; when they diverged, the asymmetric consolidation policy ensured that only well-justified additions were retained. Thus, Agent 2 contributes to the robustness and richness of the design space exploration, complementing the precision of Agent 1.

The consolidated architecture (Cons.) closely followed the performance of A1 throughout the study. Under the *Pure Intersection* policy (E1–E5), minor reductions were observed only for BookStore in two experiments due to the removal of services identified by a single architect agent. After adopting the *Asymmetric Consolidation* policy (E7), the consolidated architecture matched A1 in both systems, indicating that the validator no longer reduced the quality of the generated decomposition.

The highest service-identification performance was achieved in Experiment E3, where both A1 and Cons. reached an F1-score of 1.00 for PetClinic and BookStore. Across the repeated experiments (E1–E5), the proposed workflow achieved a mean combined F1-score of approximately 0.95 with low variability, indicating that the proposed pipeline consistently identified the expected microservice components.

Answer to RQ1

The results indicate that the proposed Agentic AI pipeline can accurately identify the expected microservice components from textual requirements. It achieved a maximum combined F1-score of 1.00 and a mean F1-score of approximately 0.95 across the repeated experiments. The asymmetric consolidation strategy preserved the high accuracy of Agent 1 while eliminating the small losses introduced by the Pure Intersection policy.

Inter-Service Interactions (RQ2). Table 2 summarizes the F1-scores obtained for recovering the communication links between microservices. Overall, Agent 1 (A1) outperformed the remaining configurations, achieving mean F1-scores of 0.84 ($\sigma = 0.10$) for PetClinic and 0.64 ($\sigma = 0.13$) for BookStore across Experiments E1–E5. Unlike service identification, recovering communication links proved to be a more challenging task due to the larger number of possible relationships and the need to correctly infer dependencies that are only implicitly described in the user stories.

Table 2: F1-score for inter-service communication links.

Exp.	PetClinic			Bookstore		
	A1	A2	Cons.	A1	A2	Cons.
E1	0.81	0.00	0.77	0.57	0.00	0.57
E2	0.81	0.23	0.77	0.50	0.00	0.50
E3	1.00	0.19	1.00	0.75	0.00	0.75
E4	0.84	0.17	0.84	0.57	0.00	0.57
E5	0.73	0.09	0.71	0.82	0.00	0.82
E6	0.84	0.72	0.82	0.61	0.00	0.61
E7	0.84	0.77	**0.84**	0.57	0.04	**0.57**

Agent 2 (A2) produced substantially lower F1-scores across all experiments. Although its exploratory architectural strategy occasionally identified valid interactions, it also introduced numerous unsupported communication links, resulting in consistently lower precision than A1.

The consolidated architecture (Cons.) closely followed the behavior of A1. Under the *Pure Intersection* policy (E1–E5), a small reduction in F1 was observed whenever valid interactions were identified by only one architect agent and therefore removed during consolidation. After adopting the *Asymmetric Consolidation* policy (E7), the consolidated architecture achieved the same F1-score as A1 in both systems, eliminating the degradation introduced by the previous policy.

The highest interaction-recovery performance was obtained in Experiment E3, where both A1 and Cons. reached an F1-score of 1.00 for PetClinic and 0.75 for BookStore. Across the repeated experiments (E1–E5), the proposed workflow achieved a mean combined F1-score of approximately 0.74, indicating stable performance despite the greater complexity of recovering inter-service communication links compared with service identification.

Answer to RQ2

The proposed Agentic AI pipeline accurately recovered inter-service communication links, achieving a maximum combined F1-score of 0.88 and a mean combined F1-score of approximately 0.74 across the repeated experiments. Although recovering interactions proved more challenging than identifying services, the Asymmetric Consolidation strategy preserved the performance of Agent 1 while avoiding the loss of valid interactions introduced by the Pure Intersection policy.

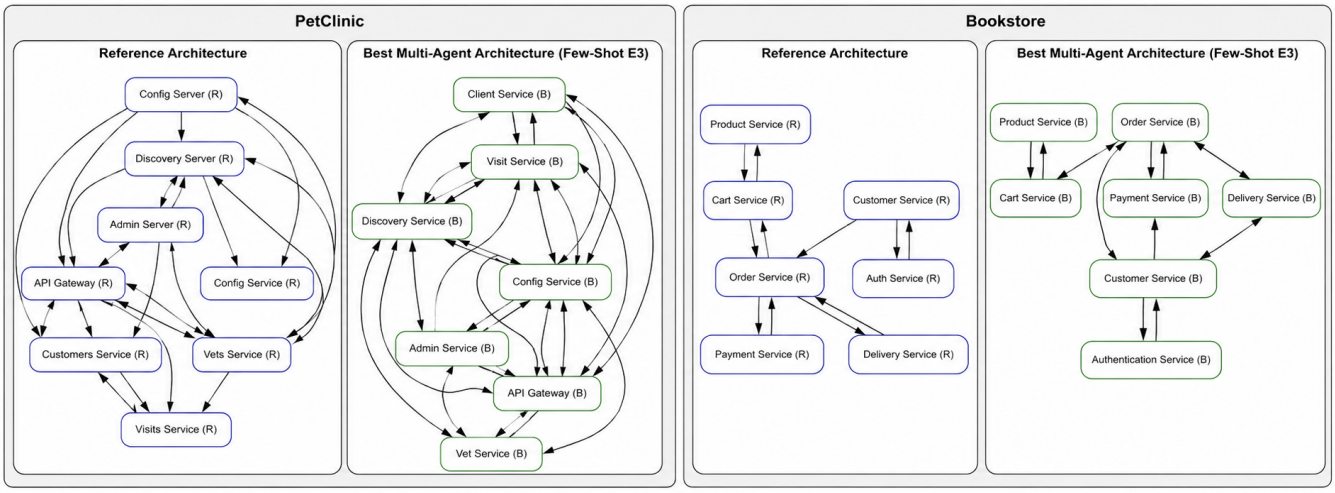

Figure 3: Comparison between the reference architectures and the proposed Agentic AI pipeline (Experiment E3).

Integrated Discussion. Figure 3 complements the quantitative evaluation by comparing the reference architectures with the best architecture generated by the proposed Agentic AI pipeline. In both PetClinic and BookStore, the generated architectures recovered nearly all expected microservices, showing that the proposed workflow effectively identifies the main business capabilities described in the user stories. However, service interactions proved considerably more difficult to infer than service boundaries. Although the generated architectures preserve the overall systems' organization, they introduce additional communication links—particularly in PetClinic—indicating a tendency to overestimate dependencies among semantically related services. This observation is consistent with the quantitative results, where service identification outperformed interaction recovery.

Compared with Pereira et al. [15], who reported F1-scores of 0.97 (services) and 0.86 (interactions) using a single GPT-4 model, our best execution (E3) reached 1.00 and 0.88, respectively, while the mean combined F1-scores were 0.95 and 0.74. The additional complexity of a multi-agent workflow did not consistently improve accuracy; its main advantage lies in a structured, extensible reasoning process with explicit validation. Our findings complement ArchiLLM [2] by isolating the architecture-generation stage and showing that collaboration strategy, rather than agent count, determines effectiveness. More broadly, these results suggest that the value of Agentic AI lies not in replacing the architect, but in making the design process more auditable, collaborative, and transparent — qualities that are particularly relevant in early-stage software design, where requirements are still evolving and architectural decisions must be regularly revisited.

5 Implications

The findings of this study have important implications for both research and industrial practice.

From a research perspective, our results indicate that adopting multiple agents does not inherently lead to superior architectural outcomes. The experiments suggest that collaboration strategy influences architectural quality more than the number of agents. Asymmetric consolidation preserved accuracy while avoiding strict consensus losses, motivating future research on richer collaboration mechanisms, including negotiation protocols, iterative refinement, and human-in-the-loop workflows.

From an industrial perspective, the proposed pipeline supports the transformation of user stories into microservice architectures through a structured and transparent reasoning process. The workflow provides intermediate proposals and explicit validation steps, making the design process more auditable during early development stages when decisions must be justified before implementation. Rather than replacing architects, the approach serves as a decision-support tool that preserves human oversight. Organizations should invest in well-designed collaboration and validation workflows rather than assuming that additional agents alone will improve quality.

6 Threats to Validity

To ensure a comprehensive evaluation, we discuss potential threats to validity following the four dimensions commonly adopted in empirical software engineering [22].

Construct Validity. The reference architectures, while widely adopted, are not the only valid solutions. We considered both service identification and interactions as complementary views. Generating architectures solely from user stories may omit implementation details; future work could incorporate source code analysis to complement the textual perspective.

Internal Validity. LLM outputs are non-deterministic. We repeated experiments (E1–E5) and evaluated multiple strategies under identical conditions to characterize variability. The chosen agent roles represent one possible organization; alternative configurations may produce different results. Therefore, the findings should be interpreted as evidence for the evaluated workflow rather than for Agentic AI in general.

External Validity. The evaluation used two benchmark systems with English user stories. Although these systems differ in size and complexity, results may not generalize to industrial systems, other domains, multilingual requirements, or different architectural styles. Replications with additional systems and LLMs are needed to assess broader applicability.

Reliability. All artifacts are publicly available to support replication. Exact outputs may vary over time due to LLM updates beyond the authors' control.

7 Final Remarks

This study investigated an Agentic AI pipeline for generating microservice architectures from user stories. The workflow combines specialized agents that produce and consolidate architectural proposals through different validation strategies. The approach was evaluated on two benchmark systems (BookStore and PetClinic) by comparing generated architectures against reference decompositions in terms of service identification and inter-service interactions.

Regarding *RQ1*, the pipeline accurately identified microservice boundaries, achieving perfect agreement with reference architectures in the best executions. *RQ2* showed competitive recovery of inter-service interactions. Although the pipeline did not consistently surpass single-LLM baselines, asymmetric consolidation preserved high-quality decisions more effectively than strict consensus.

Beyond the quantitative results, this study shows that the primary contribution of Agentic AI is not necessarily higher architectural accuracy, but a more structured, transparent, and extensible architectural reasoning process. Although simply increasing the number of agents did not consistently improve the generated architectures, the experiments demonstrate that collaboration mechanisms—particularly validation and consolidation strategies—play a central role in the overall workflow. For practitioners, the proposed approach offers an inspectable decision-support process that preserves the software architect as the final decision maker while facilitating collaborative architectural analysis.

Future work should investigate richer collaboration mechanisms (e.g., reviewer agents, iterative refinement, negotiation protocols), integrate human feedback through Human-in-the-Loop approaches, and evaluate the workflow on additional systems, LLMs, and architectural styles to establish best practices for Agentic AI in early-stage design.

Generative AI Use

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

Artifact Availability

All artifacts used in this study (e.g., user stories, generated architectures, and evaluation scripts) are publicly available to ensure transparency, facilitate replication, and support future research.[1]

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] Yalemisew Abgaz, Andrew McCarren, Peter Elger, David Solan, Neil Lapuz, Marin Bivol, Glenn Jackson, Murat Yilmaz, Jim Buckley, and Paul Clarke. 2023. Decomposition of monolith applications into microservices architectures: A systematic review. *IEEE Transactions on Software Engineering* 49, 8 (2023), 4213–4242.

[2] Marco Calamo, Flavia Monti, Fabio Spaziani, Francesco Leotta, and Massimo Mecella. 2026. From User Stories to Architectures: Using LLM-powered Agents to Design and Improve Microservice-based Software . *IEEE Software* 01 (Feb. 2026), 1–8. doi:10.1109/MS.2026.3668669

[3] Rafael Capilla, Anton Jansen, Antony Tang, Paris Avgeriou, and Muhammad Ali Babar. 2016. 10 years of software architecture knowledge management: Practice and future. *Journal of Systems and Software* 116 (2016), 191–205.

[4] Fabiano Dalpiaz and Sjaak Brinkkemper. 2021. Agile requirements engineering: from user stories to software architectures. In *2021 IEEE 29th international requirements engineering conference (RE)*. IEEE, 504–505.

[5] Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. 2024. Can llms generate architectural design decisions?-an exploratory empirical study. In *2024 IEEE 21st International Conference on Software Architecture (ICSA)*. IEEE, 79–89.

[6] Jonas Fritzsch, Justus Bogner, Alfred Zimmermann, and Stefan Wagner. 2018. From monolith to microservices: A classification of refactoring approaches. In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Springer, 128–141.

[7] Matthias Gysel, Lukas Kölbener, Peter Gantenbein, and Olaf Zimmermann. 2016. Service cutter: A systematic approach to service decomposition. In *European Conference on Service-Oriented and Cloud Computing*. Springer, 185–200.

[8] Christine Hofmeister, Robert Nord, and Dilip Soni. 2000. *Applied software architecture*. Addison-Wesley Professional.

[9] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79.

[10] Mohammad Imranur, Sebastiano Panichella, Davide Taibi, et al. 2019. A curated dataset of microservices-based systems. *arXiv preprint arXiv:1909.03249* (2019).

[11] Jasmin Jahić and Ashkan Sami. 2024. State of Practice: LLMs in Software Engineering and Software Architecture. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 311–318.

[12] Munezero Immaculee Joselyne, Gaurav Bajpai, and Frederic Nzanywayingoma. 2021. A systematic framework of application modernization to microservice based architecture. In *2021 International Conference on Engineering and Emerging Technologies (ICEET)*. IEEE, 1–6.

[13] Vinicius L Nogueira, Fernando S Felizardo, Aline MMM Amaral, Wesley KG Assunção, and Thelma E Colanzi. 2024. Insights on Microservice Architecture Through the Eyes of Industry Practitioners. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 765–777.

[14] Idris Oumoussa and Rajaa Saidi. 2024. Evolution of microservices identification in monolith decomposition: A systematic review. *IEEE Access* 12 (2024), 23389–23405.

[15] Jose Pereira, Danyllo Albuquerque, Mirko Perkusich, Guillermo Rodríguez, Jorge Díaz-Pace, Kyller Gorgônio, and Angelo Perkusich. 2025. Toward Generating Microservice Architectures from Textual Requirements with Large Language Models. In *Anais do XIX Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 79–89. doi:10.5753/sbcars.2025.14591

[16] Mirko Perkusich, Lenardo Chaves e Silva, Alexandre Costa, Felipe Ramos, Renata Saraiva, Arthur Freire, Ednaldo Dilorenzo, Emanuel Dantas, Danilo Santos, Kyller Gorgônio, et al. 2020. Intelligent software engineering in the context of agile software development: A systematic literature review. *Information and Software Technology* 119 (2020), 106241.

[17] Mark Richards and Neal Ford. 2020. *Fundamentals of software architecture: an engineering approach*. O'Reilly Media.

[18] Yamina Romani, Okba Tibermacine, and Chouki Tibermacine. 2022. Towards migrating legacy software systems to microservice-based architectures: a data-centric process for microservice identification. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 15–19.

[19] Ana Martínez Saucedo, J Andres Diaz-Pace, Hernán Astudillo, and Guillermo Rodríguez. 2024. On the Variability of Microservice Decompositions: A Data-Driven Analysis. In *2024 L Latin American Computer Conference (CLEI)*. IEEE, 1–9.

[20] Ana Martínez Saucedo, Guillermo Rodríguez, Fabio Gomes Rocha, and Rodrigo Pereira dos Santos. 2025. Migration of monolithic systems to microservices: A systematic mapping study. *Information and Software Technology* 177 (2025), 107590.

[21] Alberto Tagliaferro, Simone Corbo, and Bruno Guindani. 2025. Leveraging llms to automate software architecture design from informal specifications. In *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 291–299.

[22] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2012. *Experimentation in software engineering*. Vol. 236. Springer.

[23] Eberhard Wolff. 2016. *Microservices: flexible software architecture*. Addison-Wesley Professional.

[24] Zibin Zheng, Kaiwen Ning, Qingyuan Zhong, Jiachi Chen, Wenqing Chen, Lianghong Guo, Weicheng Wang, and Yanlin Wang. 2025. Towards an understanding of large language models in software engineering tasks. *Empirical Software Engineering* 30, 2 (2025), 50.

[1] Replication package available at: https://zenodo.org/records/21287467.