POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Integrating Large Language Models in Microservice Architecture Decomposition

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Raffaele Russo

Advisor: Prof. Giovanni Ennio Quattrocchi

Academic year: 2025-2026

1. Introduction

The migration from monolithic systems to microservices [1] architectures represents a major architectural transformation in modern software engineering. While microservices offer advantages [2] in terms of scalability, modularity, and independent deployment, the decomposition of an existing monolith into coherent and loosely coupled services remains a complex engineering problem [3, 4]. Identifying appropriate service boundaries, ensuring data ownership, and balancing cohesion and communication cost require systematic decision-making support. Among the proposed solutions, optimization-based approaches model architectural decomposition as a formal decision problem. In particular, the Cromlech framework [5] formulates monolith decomposition as a Mixed-Integer Linear Programming problem, balancing organizational cohesion and operational runtime costs through a mathematically defined objective function. Cromlech requires as input a structured specification describing system entities, operations, data access patterns, invocation frequencies, and architectural constraints. However, the practical adoption of such optimization-based frameworks is hindered by a fundamental limitation. The required input specification must be manually derived from informal requirement artifacts, such as user stories written in natural language [6]. This translation process is cognitively demanding, time-consuming, and prone to inconsistencies. The difficulty of bridging informal requirements and formal architectural models constitutes a major obstacle, referred to in this work as the Input Gap. Recent advances in Large Language Models have shown strong capabilities in semantic interpretation and structured output generation [7]. These models are capable of transforming unstructured natural language descriptions into structured representations that comply with predefined schemas. This suggests a potential integration between generative language models and formal optimization frameworks. This thesis proposes a structured pipeline that integrates Large Language Models with the Cromlech framework in order to automate the generation and refinement of architectural specifications. The approach extracts entities, operations, data access patterns, and transactional properties directly from user stories, producing machine-readable specifications suitable for optimization. Furthermore, it introduces model-inferred co-location constraints, referred to as Forced Operations, whose impact on the optimization outcomes is systemati-

cally evaluated. The objective of this research is twofold: first, to assess the feasibility and reliability of using Large Language Models to bridge the Input Gap by leveraging recent advances in LLM-based Requirements Engineering [8] in optimization-driven architectural decomposition; second, to analyze how model-inferred architectural constraints influence the quality and characteristics of the resulting microservice partitioning.

2. Problem Overview

Optimization-based microservice decomposition requires a formal and structured system model, including entities, operations, data access patterns, invocation frequencies, and architectural constraints. Frameworks such as Cromlech assume the availability of such information, yet in early development stages requirements are typically expressed as informal artifacts (e.g., user stories), creating a misalignment between requirement abstraction and the level of formalization required by optimization tools.

Manually translating user stories into formal specifications is cognitively demanding and prone to ambiguity, incompleteness, and terminological inconsistency. Moreover, the process is highly dependent on the designer's expertise, reducing reproducibility and turning input modeling into a critical bottleneck.

Since optimization quality directly depends on the correctness and completeness of the input model, inaccuracies or omissions may lead to suboptimal or architecturally inconsistent decompositions. The challenge therefore lies not only in solving the optimization problem, but in constructing a semantically sound and architecturally meaningful formal representation.

Additionally, certain architectural constraints — such as required co-location of operations due to transactional or consistency requirements — do not naturally emerge from cohesion and communication cost criteria alone. Failure to capture these relationships restricts the solution space and may yield decompositions misaligned with domain needs.

This work addresses the need for a systematic methodology capable of transforming informal requirements into a complete and evaluable formal specification, reducing manual effort while preserving the rigor required for automated architectural synthesis.

3. Proposed Methodology

This thesis proposes a structured methodology that integrates Large Language Models (LLMs) into an optimization-based workflow for microservice architectural design. The objective is to automatically transform natural language requirements into a formal specification suitable for mathematical decomposition.

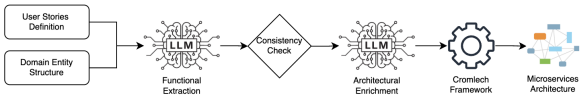

Figure 1: Proposed solution workflow overview.

As illustrated in Figure 1, the pipeline is organized into four sequential phases and explicitly separates requirement interpretation from architectural synthesis.

In the first phase, an LLM analyzes user stories together with a predefined domain entity model to extract elementary application operations, associated data access patterns (distinguishing reads and writes), and relevant dynamic properties such as invocation frequency and transactional requirements. The output is a formally structured `YAML` specification aligned with the schema required by the optimization framework.

A validation stage ensures syntactic correctness, alignment with the domain schema, and traceability between user stories and generated operations, preventing semantic inconsistencies and hallucinations.

Subsequently, an architectural enrichment step infers co-location constraints between operations (*Forced Operations*) when justified by transactional atomicity, performance considerations, or domain consistency. This enrichment follows a controlled update paradigm and does not alter the functional structure, thus preserving the separation between functional modeling and architectural reasoning.

Finally, the enriched specification is processed by the Cromlech framework, which formulates the decomposition problem as a Mixed Integer Linear Programming (MILP) model. The optimization maximizes intra-service cohesion and minimizes inter-service communication costs,

while enforcing inferred co-location constraints as hard requirements.
Overall, the methodology combines the interpretative capabilities of LLMs in formalizing requirements with the mathematical rigor of constrained optimization, enabling an automated, controllable, and formally grounded microservice decomposition process.

4. Experimental Methodology

The effectiveness of the proposed methodology is evaluated through a structured and replicable experimental framework designed to isolate and quantify the contribution of each pipeline component. As illustrated in Figure 2, the experimental workflow mirrors the logical structure of the proposed pipeline and is organized into three incremental phases, each targeting a distinct level of analysis: functional accuracy, architectural reasoning, and optimization impact.

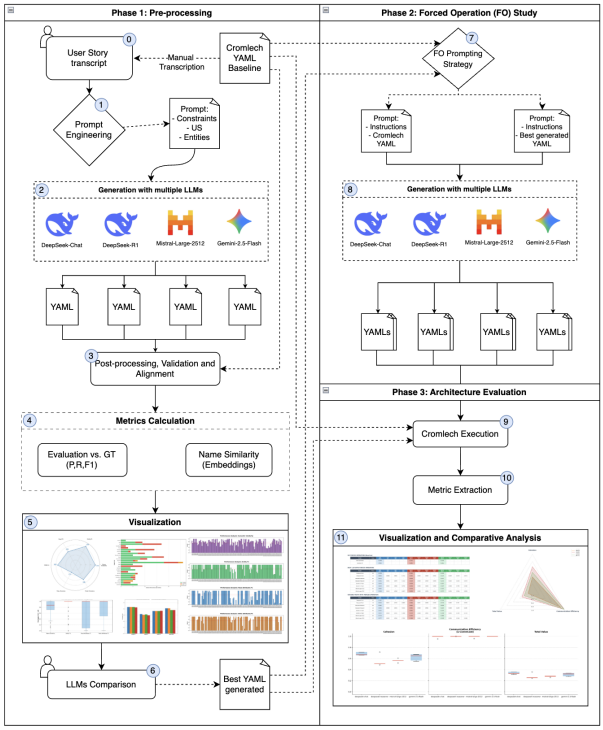

Figure 2: Methodological workflow: from User Stories acquisition to `YAML` specification synthesis.

4.1. Phase 1: Functional Extraction Evaluation

The first phase evaluates the capability of Large Language Models to reconstruct a formal functional specification from natural language requirements. A Ground Truth (GT) `YAML` specification was defined and used to derive a corresponding User Stories dataset, ensuring the availability of a precise reference for quantitative comparison.
For identical inputs (User Stories and domain schema), multiple LLMs were queried using a constrained prompt to guarantee structural compliance with the optimization framework. Generations were performed in deterministic mode ($T = 0$) to ensure reproducibility and isolate model capability as the only source of variation.
The generated specifications were validated and compared against the GT using structural metrics (Precision, Recall, F1-score), categorical accuracy for discrete parameters, and semantic similarity measures for operation naming. The best-performing configuration was selected as reference for the subsequent phases.

4.2. Phase 2: Forced Operations Analysis

The second phase analyzes the inference of architectural co-location constraints (Forced Operations). Starting from both the Cromlech baseline specification and the best-generated configuration from Phase 1, models were asked to enrich the specifications with colocation constraints.
Experiments were conducted in deterministic ($T = 0$) and stochastic ($T = 1$) modes, with multiple independent runs to assess variability and stability. The analysis considers both the presence of inferred constraints and their recurrence across executions, providing a measure of robustness in architectural reasoning.

4.3. Phase 3: Architectural Evaluation via Cromlech

The third phase evaluates the architectural impact of the inferred constraints through the optimization phase. All baseline and enriched specifications are provided as input to the Cromlech framework, which formulates the decomposition problem as a Mixed Integer Linear Programming (MILP) model.
The solver generates microservice partitions by optimizing internal cohesion and inter-service communication cost, with balancing parameter $\alpha = 0.5$. For each configuration, quantitative

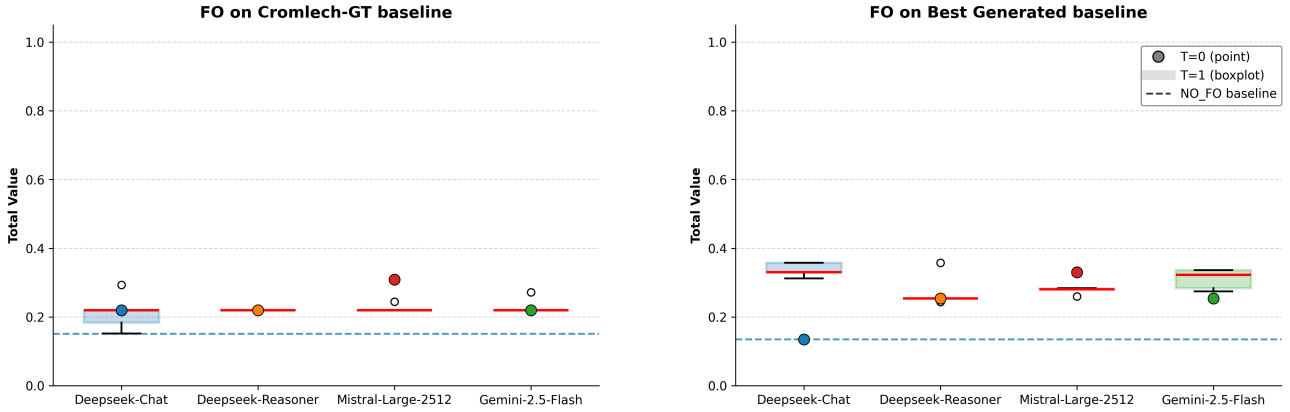

Figure 3: (Trainticket) Impact of LLM-inferred Forced Operations (FO) on Cromlech Total Value. Left: FO applied to the Cromlech-GT baseline; right: FO applied to the Best Generated baseline.

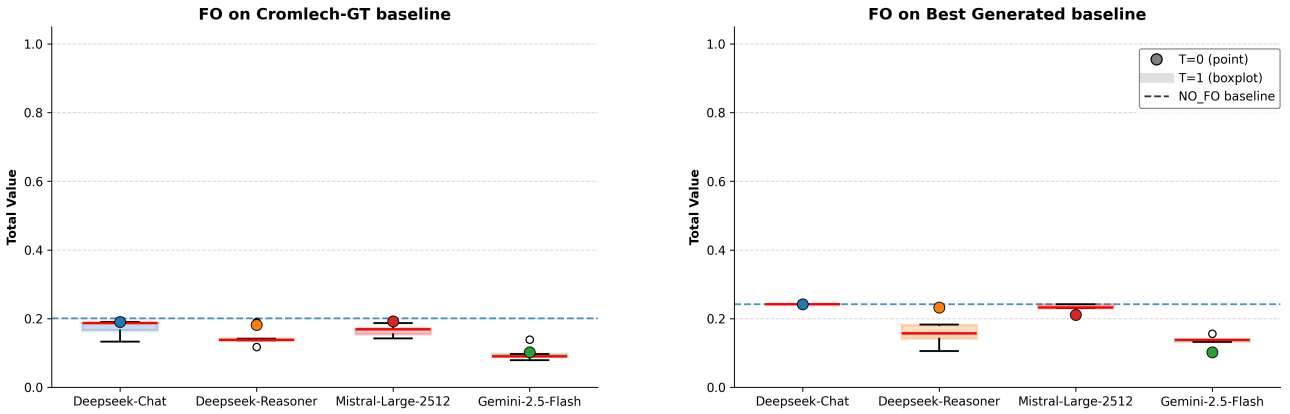

Figure 4: (Tutored) Impact of LLM-inferred Forced Operations (FO) on Cromlech Total Value. Left: FO applied to the Cromlech-GT baseline; right: FO applied to the Best Generated baseline.

metrics (cohesion, communication cost, and total objective value) were collected. In stochastic settings, the distribution of results across multiple runs was analyzed to evaluate architectural stability.

This multilevel evaluation framework enables a controlled and quantitative assessment of how language model decisions influence the quality, robustness, and overall value of the final microservice decomposition.

5. Results

The proposed workflow was evaluated on two case studies with different levels of complexity: *Trainticket*, a benchmark microservices system simulating a railway reservation platform, and *Tutored*, an EdTech platform supporting students and graduates. The evaluation focused on three main aspects: (i) the quality of formal specifications extracted from textual requirements by Large Language Models (LLMs), (ii) the impact of LLM-generated architectural constraints (Forced Operations) on microservice decomposition quality, and (iii) the stability of the overall process under deterministic ($T = 0$) and stochastic ($T = 1$) configurations.

Overall, LLMs were able to transform informal user stories into structured architectural specifications of sufficient quality to enable optimization-based synthesis. Across both case studies, entity identification and semantic consistency of generated operations were generally satisfactory, while modeling fine-grained attribute-level access patterns—especially read accesses—remained the most challenging aspect, particularly in the Tutored domain where requirements are more implicit.

Figures 3 and 4 provide a quantitative summary of the architectural impact of FO through Cromlech *Total Value*. For each model, the *NO_FO* baseline is shown as a dashed horizontal line, while the FO-enabled configurations are

reported both in deterministic mode ($T = 0$, point estimate) and in stochastic mode ($T = 1$, boxplot over multiple runs). Higher Total Value indicates a better trade-off between intra-service cohesion and inter-service communication cost under the same optimization setting.

In **Trainticket** (Figure 3), FO yields a clear positive effect when applied to the Cromlech-GT baseline (left), consistently increasing Total Value across all evaluated models. When FO is applied to the Best Generated baseline (right), gains become model-dependent: some models still improve Total Value substantially, while others show limited or no improvement in deterministic decoding, suggesting that constraint usefulness depends on the quality of the inferred co-location relationships and on the starting specification. Under $T = 1$, variability differs across models, highlighting distinct robustness profiles in the constraint inference step.

Conversely, in **Tutored** (Figure 4), FO does not produce systematic improvements. On the Cromlech-GT baseline (left), FO frequently reduces Total Value, indicating that inferred constraints may over-restrict the solution space or encourage excessive consolidation in a domain with more implicit requirements. On the Best Generated baseline (right), which already achieves strong performance without FO, the introduction of constraints is mostly neutral for some models or detrimental for others, and can also increase dispersion under stochastic decoding.

Taken together, these results show that integrating LLM-based requirement extraction with optimization-based architectural synthesis is feasible and impactful, while the benefits of FO are context-sensitive: they are consistently positive in Trainticket, whereas in Tutored they may introduce unfavorable trade-offs depending on the inferred constraints and on the baseline specification quality.

6. Conclusions & Future Work

This thesis addresses the challenge of translating informal textual requirements into the formal specifications required by optimization-based microservice decomposition frameworks, commonly referred to as the Input Gap. To tackle this problem, a structured methodology was proposed that integrates Large Language Models (LLMs) into a controlled architectural design pipeline. The approach enables the automatic extraction of structured specifications from user stories, which are then processed by a formal optimization engine.

The experimental results demonstrate that LLMs can generate syntactically valid and semantically coherent specifications suitable for formal optimization. In particular, the introduction of LLM-inferred co-location constraints significantly influences the resulting architectures, improving structural cohesion and domain alignment in several cases while introducing predictable trade-offs due to a reduced solution space. The analysis also highlights the importance of model selection and decoding configuration for ensuring stability and reproducibility. These findings confirm that LLMs can effectively support early-stage architectural modeling when embedded within a verifiable, constraint-driven framework.

Although the evaluation was conducted on a limited set of case studies and relies solely on user stories, the results indicate promising directions for future research, including multi-agent LLM architectures, grammar-constrained decoding, and integration with static code analysis and industrial-scale datasets. Overall, this work shows that the controlled combination of LLMs and formal optimization techniques provides a technically grounded and scalable approach to automated microservice architecture design.

References

[1] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2021.

[2] 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*, 2018.

[3] Lukas Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Sorg. Service cutter: A systematic approach to service decomposition. In *European Conference on Service-Oriented and Cloud Computing*, pages 185–200. Springer, 2016.

[4] Shriti Kalia et al. Mono2micro: A practical and effective tool for decomposing monolithic java applications to microservices. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1214–1224, 2021.

[5] Giovanni Quattrocchi, Davide Cocco, Simone Staffa, Alessandro Margara, and Gianpaolo Cugola. Cromlech: Semi-automated monolith decomposition into microservices. *IEEE Transactions on Services Computing*, 2024.

[6] Tor Thorsrud Sporsem, Torgeir Dingsøyr, and Klaas-Jan Stol. User stories as boundary objects in agile requirements engineering: A theoretical literature review. *Journal of Systems and Software*, 2025. URL `https://doi.org/10.1016/j.jss.2025.112693`.

[7] 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, 2024. URL `https://arxiv.org/abs/2308.10620`.

[8] Mohammad Amin Zadenoori, Jacek Dąbrowski, Waad Alhoshan, Liping Zhao, and Alessio Ferrari. Large language models (llms) for requirements engineering (re): A systematic literature review, 2025. URL `https://arxiv.org/abs/2509.11446`.