ArchiGenMS: A Neuro-Symbolic Evolutionary Framework for Greenfield Microservice Design

Daniel Narváez[1,3], **Nicolás Battaglia**[1], **Alejandro Fernández**[2], **Gustavo Rossi**[1,2]

[1]CAETI – Universidad Abierta Interamericana (UAI) – Facultad de Tecnología Informática
Av. Montes de Oca 745 – Ciudad de Buenos Aires – Argentina

[2]LIFIA – Universidad Nacional de La Plata (UNLP) – Facultad de Informática
La Plata (1900) – Argentina

[3]Keiser University Latin American Campus – Department of Software Engineering
San Marcos, Department of Carazo, Nicaragua

josedaniel.narvaezf@alumnos.uai.edu.ar, nicolas.battaglia@uai.edu.ar

{alejandro.fernandez,gustavo}@lifia.info.unlp.edu.ar

Abstract. *Designing microservices from greenfield requirements remains challenging because early textual artifacts are ambiguous and LLM-based generation may produce structurally inconsistent decompositions. This paper presents* ***ArchiGenMS****, a neuro-symbolic framework that combines LLM-guided semantic exploration with a computable* ***Lean 4*** *validation kernel inside an elitist evolutionary loop. The framework operates over an early architectural abstraction of services, operations, parameters, and inter-service calls. Across 22 datasets from the Dalpiaz corpus, the percentage of Lean-valid candidates increases from 74.5% to 99.5% in five generations, while the average cohesion proxy improves from 0.29 to 0.23. These results position ArchiGenMS as a reproducible and formally constrained approach to AI-assisted microservice discovery.*

Keywords: *Microservices, Generative AI, Formal Verification, Lean 4, Evolutionary Algorithms, Neuro-symbolic AI.*

Resumo. *Projetar microsserviços a partir de requisitos greenfield continua sendo desafiador porque artefatos textuais iniciais são ambíguos e a geração com LLMs pode produzir decomposições estruturalmente inconsistentes. Este artigo apresenta o* ***ArchiGenMS****, um framework neuro-simbólico que combina exploração semântica guiada por LLMs com um núcleo computável de validação em* ***Lean 4*** *dentro de um laço evolucionário elitista. O framework opera sobre uma abstração arquitetural inicial de serviços, operações, parâmetros e chamadas entre serviços. Em 22 conjuntos de dados do corpus de Dalpiaz, a porcentagem de candidatos válidos em Lean cresce de 74,5% para 99,5% em cinco gerações, enquanto a proxy média de coesão melhora de 0,29 para 0,23. Esses resultados posicionam o ArchiGenMS como uma abordagem reproduzível e formalmente restrita para descoberta de microsserviços assistida por IA.*

Palavras-chave: *Microsserviços, IA Generativa, Verificação Formal, Lean 4, Algoritmos Evolutivos, IA Neuro-simbólica.*

1. Introduction

Microservices remain a dominant style for scalable cloud-native systems, yet their greenfield design is still difficult because architects must transform incomplete textual requirements into service boundaries that are cohesive, loosely coupled, and maintainable [Newman 2021, Dragoni et al. 2017, Taibi et al. 2017, Ünlü et al. 2024]. This challenge becomes sharper when the available artifacts are lightweight user stories, whose ambiguity directly affects decomposition quality [Lucassen et al. 2016, Al-Debagy and Martinek 2020]. Existing automated approaches partially alleviate this burden, but purely semantic decompositions may still produce structurally inconsistent outputs and transfer technical debt to later stages [Vera-Rivera et al. 2023, Bajaj et al. 2022, Esposito et al. 2025, Neri et al. 2020].

Large Language Models (LLMs) have opened new possibilities for automated software engineering and architectural ideation [Belzner et al. 2023, Díaz-Pace et al. 2024, Dhar et al. 2024]. However, as our prior secondary studies report, generative outputs may look plausible while still violating basic structural assumptions, a risk we previously framed as architectural hallucination [Narváez et al. 2025b, Narváez et al. 2024, Adams et al. 2023, Chen et al. 2021]. This motivates a constrained-search perspective in which LLMs explore candidate decompositions while a second component filters or penalizes infeasible structures [Harman et al. 2012, Mkaouer et al. 2015].

In that context, this paper presents **ArchiGenMS**, a neuro-symbolic framework that combines LLM-guided variation with a **Lean 4** executable validation kernel [Moura and Ullrich 2021]. Rather than generating a complete deployable architecture, the framework optimizes an *early structural abstraction* consisting of services, operations, parameters, and directed calls. Within that abstraction, Lean 4 enforces lightweight invariants, while architectural proxies rank valid candidates. The contribution is therefore a formally constrained search process for microservice discovery, not a proof of full architectural soundness. The empirical study over 22 datasets shows consistent internal improvement in validity and cohesion, and the entire pipeline is available as a reproducible package [ArchiGenMS Project 2025].

2. Related Work

Research on greenfield microservice discovery has evolved along three convergent lines. A first line focuses on requirement-driven processes. In this context, the *Behavior-Driven Microservice Architecture* (BDMA) frames decomposition as an iterative transition from behavioral requirements to candidate service boundaries, which is especially suitable for agile greenfield settings [Battaglia et al. 2025]. A second line uses semantic similarity, domain analysis, and metric-based decomposition. Domain-driven design and metric-based evaluations remain central references for service boundary decisions [Evans 2004, Taibi and Systä 2019]. Approaches such as SEMGROMI show that NLP can cluster requirements effectively, but semantic grouping alone does not ensure structural consistency or maintainability [Vera-Rivera et al. 2023, Al-Debagy and Martinek 2020, Neri et al. 2020]. Moreover, embedding-based analyses may suffer from anisotropy, which affects discrimination among functionally distinct requirements [Pérez et al. 2025].

A third line explores generative AI and formal methods. Recent work shows that

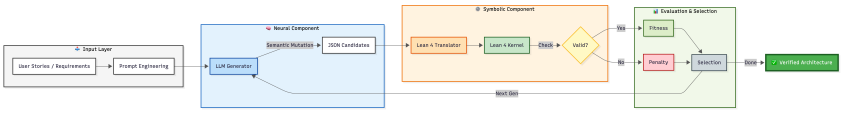


Figure 1. ArchiGenMS as a neuro-symbolic search pipeline.

LLMs can support architecture design and microservice discovery from requirements, but also highlights the stochastic instability of unconstrained generation [Belzner et al. 2023, Dhar et al. 2024, Pereira et al. 2025, Adams et al. 2023]. Formal methods offer a natural complement because they turn structural assumptions into executable checks [Clarke and Wing 1996, Moura and Ullrich 2021]. Our earlier doctoral work and preliminary study already suggested this direction [Narváez 2025, Narváez et al. 2025a]. The present paper advances that line with a reproducible neuro-symbolic loop that integrates LLM-based exploration, Lean 4 validation, and fitness-guided selection.

3. The ArchiGenMS Framework

Figure 1 summarizes ArchiGenMS. Starting from user stories, the framework generates candidate decompositions with an LLM, validates them in Lean 4, and ranks them with structural proxies. The search is semantic because candidates are inferred from requirements, and constrained because malformed structures are rejected or heavily penalized.

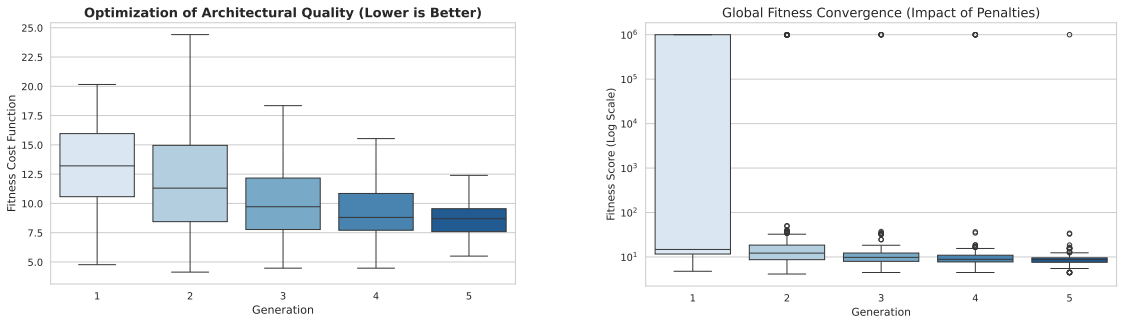
Neural layer. The neural layer uses the LLM as a semantic variation operator [Novikov et al. 2025, Harman and Yao 2010]. In generation 0, the prompt contains only user stories and requests a JSON object with two fields: `microservices` and `calls`. Each service includes a name and a list of operations; each operation contains its name and parameter list. In later generations, the prompt is enriched with one valid parent and up to three additional valid inspirations sampled from the search history. The prompt also states lightweight design heuristics such as single responsibility, low coupling, and bounded-context clarity.

Symbolic layer. ArchiGenMS operates on an early architectural abstraction composed of services, operations, parameters, and directed inter-service calls. It does not model gateways, discovery, tracing, messaging styles, data ownership, or transactional boundaries. Within this abstraction, Lean 4 acts as a computable validation kernel that checks three invariants: absence of self-calls, existence of every caller service, and existence of every callee service. Therefore, the symbolic layer enforces structural consistency under the adopted model, not full architectural correctness.

Fitness function. The search uses four computable proxies. For each service s, cohesion is estimated with an *LCOM-inspired* proxy based on parameter sharing among operations. Let p be the number of unordered operation pairs in s that share no parameter, and q the number of pairs that share at least one parameter. Then

$$LCOM(s) = \begin{cases} 0, & |Ops(s)| < 2 \text{ or } p = 0 \text{ or } q = 0 \\ \dfrac{p}{p+q}, & \text{otherwise} \end{cases} \tag{1}$$

Granularity is measured as $SGM(s) = |Ops(s)|$, parameter dispersion as the standard deviation of parameter counts per operation, denoted $sgmSd(s)$, and coupling as fan-out. At genotype level, the framework reports $lcom\_avg$, sgm_max, sgm_sd_sum, and



(a) Average fitness across generations.

(b) Log-scale convergence under formal penalties.

Figure 2. Search dynamics under the ArchiGenMS fitness function.

$coupling\_max$, and defines for each valid candidate A:

$$\mathcal{F}(A) = lcom\_avg(A) + sgm\_max(A) + sgm\_sd\_sum(A) + coupling\_max(A) \quad (2)$$

Invalid candidates receive a penalty of $10^6 + |m|$, where m is the validation message returned by Lean 4. Since these components are not normalized to a common unit, $\mathcal{F}$ is used as a ranking heuristic rather than as an absolute architectural quality measure.

4. Experimental Setup

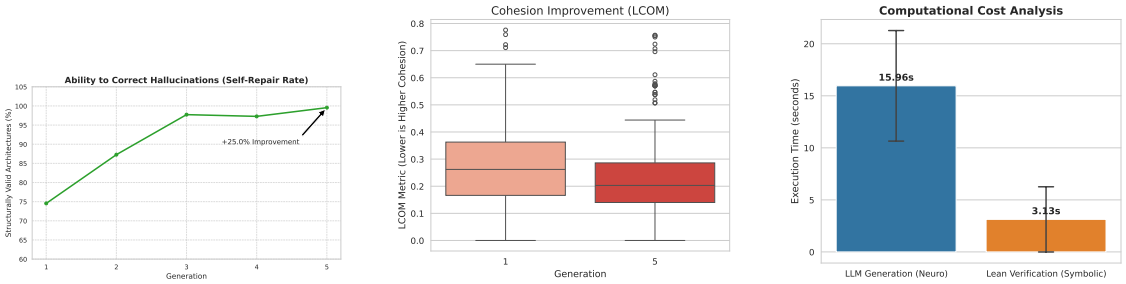
The evaluation used the 22 publicly available requirement datasets of the Dalpiaz corpus [Dalpiaz 2018]. Each dataset consists of story cards that serve as the sole textual input to the framework. This choice matches early greenfield scenarios, but it also constrains the granularity of the resulting decomposition because acceptance criteria and richer domain artifacts are not available.

The implementation uses *GPT-4o-mini* as the neural generator and a Lean 4 executable as the validation kernel. The search runs for five generations with a population size of 10 and an elite of 5 candidates per generation. The initial generation is built directly from user stories; subsequent generations are generated from the stories plus one selected parent and up to three valid inspirations. The temperature used in the reported configuration is 0.5. For every candidate, Lean 4 returns either an error message or a metrics report containing $lcom\_avg$, sgm_max, sgm_sd_sum, and $coupling\_max$. The framework code, datasets, and generated artifacts are available in the reproducible package [ArchiGenMS Project 2025].

5. Results and Limitations

Figure 2 summarizes the optimization behavior. The average aggregate fitness decreases from 16.3 in the first generation to values below 9.1 by generation 5, while the log-scale view shows the early dominance of penalized invalid candidates and the later concentration of the population inside the feasible region. Since invalid candidates receive a large formal penalty, the first stages are largely devoted to feasibility, whereas later stages emphasize structural improvement among already valid candidates.

Figure 3 shows the clearest quantitative trend: the percentage of Lean-valid candidates rises from 74.5% to 99.5%, and the average cohesion proxy improves from 0.29 to



(a) Lean-valid candidates.

(b) Average cohesion proxy.

(c) LLM vs. Lean time.

Figure 3. Validity, quality proxy, and computational overhead across generations.

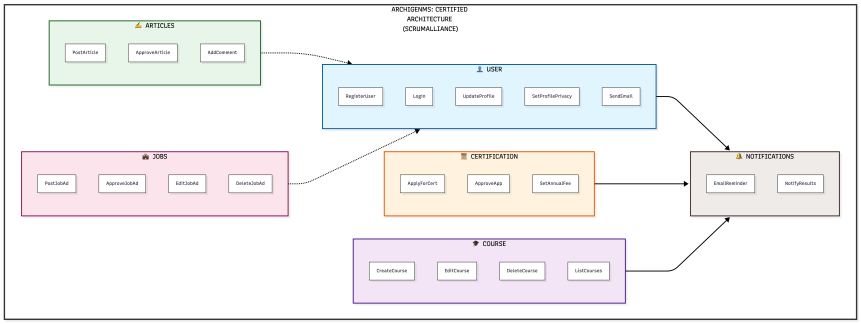


Figure 4. Selected final candidate for the ScrumAlliance dataset. The graph visualizes the service decomposition evolved by the framework under the adopted abstraction.

0.23. We interpret these results carefully. The increase in validity means that more candidates satisfy the modeled invariants of the symbolic layer; it does *not* mean that the final architectures are fully validated in the broader software-architecture sense. Likewise, the cohesion improvement should be read as improvement in the chosen service-level proxy, not as definitive proof of maintainability. Figure 3(c) also shows that the symbolic stage remains computationally lightweight relative to LLM generation, supporting the feasibility of keeping formal validation in the loop [Woods et al. 2021].

The *g10-scrumalliance* dataset illustrates the type of decomposition produced by the framework. Figure 4 shows the selected final genotype, where identity-related operations, certification management, course-related behavior, and notification concerns are separated into distinct services. The figure is useful because it makes the search outcome visually inspectable: the final candidate is not merely a metric vector, but an explicit service graph with internal operations. At the same time, it should be interpreted as an *early design artifact*; it does not yet encode messaging semantics, gateway patterns, tracing, service discovery, or data ownership.

These results should be read together with the study's limitations. First, the symbolic kernel is intentionally narrow: it checks referential integrity and absence of self-calls, but not stronger invariants such as cycle freedom, bounded-context integrity, communication mode, interface isolation, or transactional consistency. Second, the datasets contain only story cards; therefore, the framework infers operations and parameters from incomplete early requirements. Third, the empirical study evaluates internal evolutionary improvement only. It does not include expert judgment, LLM-as-a-judge assessment, statistical significance tests, or direct baselines against LLM-only, clustering-based, or

SBSE-only alternatives. Consequently, the current paper should be interpreted as an exploratory yet reproducible step toward formally constrained AI-assisted microservice discovery.

6. Conclusions and Future Work

ArchiGenMS combines LLM-based semantic exploration with a Lean 4 validation kernel to support microservice discovery from greenfield requirements. The empirical study over 22 datasets shows that, within the adopted structural abstraction, the framework substantially increases the proportion of valid candidates and improves a cohesion-oriented service proxy while keeping formal validation computationally inexpensive.

Methodologically, the work shows that neuro-symbolic search is a viable way to constrain stochastic architecture generation. Empirically, it provides reproducible evidence that lightweight formal checks can remain inside the loop without dominating runtime. At the same time, the study does not claim full architectural certification: the current model captures only an early structural view of services and calls, and the reported quality improvements are tied to explicit proxies rather than to comprehensive architecture evaluation.

Future work will extend the formal kernel with richer invariants, incorporate architectural elements such as communication style and ownership boundaries, compare against stronger baselines, add statistical analysis and expert-based evaluation, and advance toward an interactive human-in-the-loop workbench for verified architectural exploration.

References

Adams, L., Boyle, F., Boyle, P., Amoroso d'Aragona, D., Cerny, T., and Taibi, D. (2023). Chatgpt for microservice development: How far can we go? In *International Conference on Microservices*.

Al-Debagy, O. and Martinek, P. (2020). A metrics framework for evaluating microservices architecture designs. *Journal of Web Engineering*, 19(3–4):341–370.

ArchiGenMS Project (2025). ArchiGenMS: Reproducible package. `https://github.com/jdanieln/archi-ai`. GitHub Repository.

Bajaj, D., Goel, A., and Gupta, S. C. (2022). Greenmicro: identifying microservices from use cases in greenfield development. *IEEE Access*, 10:67008–67018.

Battaglia, N., Rossi, G., Fernández, A., and Narváez, D. (2025). Behavior-Driven Microservice Architecture: un marco metodológico para la identificación iterativa de microservicios en proyectos ágiles greenfield. *Revista Abierta de Informática Aplicada*, 9(1):102–123.

Belzner, L., Gabor, T., and Wirsing, M. (2023). Large language model assisted software engineering: prospects, challenges, and a case study. In *International conference on bridging the gap between AI and reality*, pages 355–374. Springer.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A.,

Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. (2021). Evaluating large language models trained on code.

Clarke, E. M. and Wing, J. M. (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643.

Dalpiaz, F. (2018). Requirements data sets (user stories). Mendeley Data. Dataset.

Dhar, R., Vaidhyanathan, K., and Varma, V. (2024). Can llms generate architectural design decisions?-an exploratory empirical study. In *2024 IEEE 21st International Conference on Software Architecture (ICSA)*, pages 79–89. IEEE.

Díaz-Pace, J. A., Tommasel, A., and Capilla, R. (2024). Helping novice architects to make quality design decisions using an llm-based assistant. In *European Conference on Software Architecture*, pages 324–332. Springer.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pages 195–216.

Esposito, M., Li, X., Moreschini, S., Ahmad, N., Cerny, T., Vaidhyanathan, K., Lenarduzzi, V., and Taibi, D. (2025). Generative ai for software architecture. applications, trends, challenges, and future directions. *arXiv preprint arXiv:2503.13310*.

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Harman, M., Mansouri, S. A., and Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, 45(1):1–61.

Harman, M. and Yao, X. (2010). Software module clustering as a multi-objective search problem. *IEEE Transactions on Software Engineering*, 37(2):264–282.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., and Brinkkemper, S. (2016). Improving agile requirements: the quality user story framework and tool. *Requirements engineering*, 21(3):383–403.

Mkaouer, W., Kessentini, M., Shaout, A., Koligheu, P., Bechikh, S., Deb, K., and Ouni, A. (2015). Many-objective software remodularization using nsga-iii. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(3):1–45.

Moura, L. d. and Ullrich, S. (2021). The lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, pages 625–635. Springer.

Narváez, D. (2025). Explorando el uso de inteligencia artificial en el descubrimiento y diseño de microservicios. In *Congresso Ibero-Americano em Engenharia de Software (CIbSE) - Doctoral Symposium*, pages 224–231, Ciudad Real, Spain. SBC.

Narváez, D., Battaglia, N., Fernández, A., and Rossi, G. (2025a). Descubrimiento automático de microservicios mediante modelos generativos y verificación formal. In

XXXI Congreso Argentino de Ciencias de la Computación (CACIC) (Viedma, 6 al 10 de octubre de 2025), Viedma, Argentina. Universidad Nacional de Río Negro (UNRN).

Narváez, D., Battaglia, N., Fernández, A., and Rossi, G. (2025b). Designing microservices using AI: A systematic literature review. *Software*, 4(1):6.

Narváez, D., Rossi, G., and Battaglia, N. (2024). Aplicación de inteligencia artificial en el diseño de microservicios. In *XXX Congreso Argentino de Ciencias de la Computación (CACIC)(La Plata, 7 al 11 de octubre de 2024)*.

Neri, D., Soldani, J., Zimmermann, O., and Brogi, A. (2020). Design principles, architectural smells and refactorings for microservices: a multivocal review. *SICS Software-Intensive Cyber-Physical Systems*, 35(1):3–15.

Newman, S. (2021). *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.".

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J., Mehrabian, A., et al. (2025). Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*.

Pereira, J. R. A., Albuquerque, D., Perkusich, M., Rodríguez, G., Díaz-Pace, J. A., Gorgônio, K., and Perkusich, A. (2025). Toward generating microservice architectures from textual requirements with large language models. In *Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS)*, pages 79–89. SBC.

Pérez, G., Mostaccio, C., and Antonelli, L. (2025). Análisis comparativo de arquitecturas de nlp para detectar similitudes entre escenarios en español. In *Workshop on Requirements Engineering (WER)*.

Taibi, D., Lenarduzzi, V., Pahl, C., and Janes, A. (2017). Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 Scientific Workshops*, pages 1–5.

Taibi, D. and Systä, K. (2019). A decomposition and metric-based evaluation framework for microservices.

Ünlü, H., Kennouche, D. E., Soylu, G. K., and Demirörs, O. (2024). Microservice-based projects in agile world: A structured interview. *Information and Software Technology*, 165:107334.

Vera-Rivera, F. H., Cuadros, E. G. P., Perez, B., Astudillo, H., and Gaona, C. (2023). Semgromi—a semantic grouping algorithm to identifying microservices using semantic similarity of user stories. *PeerJ Computer Science*, 9:e1380.

Woods, E., Erder, M., and Pureur, P. (2021). *Continuous architecture in practice: Software architecture in the age of agility and DevOps*. Addison-Wesley Professional.