LAPLACE. An AI platform designed to work with multiple code repositories for software projects

Nicolás Crespo[1], Alejandra Lliteras [1,2], Patricia Bazán [3], Joaquín Bogado [1],

[1] UNLP Facultad de Informática, Centro LIFIA, Buenos Aires Argentina
[2]CICPBA, Buenos Aires Argentina
[3] UNLP Facultad de Informática, LINTI, Buenos Aires Argentina
ncrespo@lifia.info.unlp.edu.ar,
lliteras@lifia.info.unlp.edu.ar, pbaz@info.unlp.edu.ar,
jbogado@lifia.info.unlp.edu.ar

Abstract. The emergence of large language models (LLMs) can be applied to the optimization and improvement of source code within software project development environments, providing automatic feedback and contextual suggestions, and leveraging the project repository itself as a source of knowledge to support the programming process. LAPLACE proposes the design and implementation of a unified platform for integrating large language models focused on software development. The proposed solution avoids the fragmentation resulting from the use of multiple independent environments. A key feature of LAPLACE is its high versatility regarding the large language models involved: it allows interaction with both global APIs (OpenAI, Anthropic, Google) and the deployment and use of local models (such as Ollama) by default. This flexibility ensures proper management of sensitive information, reducing or eliminating the exposure of source code to third-party services. To optimize workflows, LAPLACE implements an architecture based on Retrieval-Augmented Generation (RAG) and AI Agents. This approach uses code repositories as a contextual knowledge base, enabling the model to provide assistance, analysis, and precise feedback tailored to the project's codebase. This promotes change traceability, data security, and efficient collaboration among development teams.

Keywords: Artificial Intelligent Agents, LLMs Orchestration, Coding Agents, RAG.

LAPLACE. Una plataforma de IA para usarse con múltiples repositorios de código de proyectos de software

Resumen. La aparición de grandes modelos de lenguaje (LLMs) se puede aplicar a la optimización y mejora de código fuente dentro de los entornos de desarrollo de proyectos de software, aportando retroalimentación automática y sugerencias

contextuales y pudiendo utilizar el propio repositorio del proyecto como una de las fuentes de conocimiento para apoyar el proceso de programación. LAPLACE propone el diseño e implementación de una plataforma unificada para la integración de grandes modelos de lenguaje orientados al desarrollo de software. La solución propuesta evita la fragmentación derivada del uso de múltiples entornos independientes. Una característica fundamental de LAPLACE es su alta versatilidad respecto a los grandes modelos de lenguaje involucrados: permite interactuar tanto con APIs globales (OpenAI, Anthropic, Google), como desplegar y utilizar modelos locales (como Ollama) por defecto. Esta flexibilidad garantiza una adecuada gestión de la información sensible, reduciendo o eliminando la exposición del código fuente a servicios de terceros. Para optimizar los flujos de trabajo, LAPLACE implementa una arquitectura basada en Generación Aumentada por Recuperación (RAG) y Agentes de IA. Este enfoque utiliza los repositorios de código como base de conocimiento contextual, dotando al modelo de la capacidad de brindar asistencia, análisis y retroalimentación precisa y adaptada a la base de código del proyecto. De este modo, se favorece la trazabilidad de los cambios, la seguridad de los datos y la colaboración eficiente entre equipos de desarrollo.

Palabras clave: Agentes de Inteligencia Artificial, Orquestación de LLMs, Agentes de Codificación, RAG.

1 Introduction

With the rise of language models, numerous tools have emerged that are designed to optimize and improve source code, such as GitHub Copilot[1], which works as a development assistant for software projects. This tool provides automatic feedback and contextual suggestions, using the project repository as a knowledge base to support the programming process.

On the other hand, there are general-purpose language models, including ChatGPT[2] (developed by OpenAI[3]), Gemini[4] (from Google[5]) and Claude[6] (from Anthropic[7]), which do not provide automatic feedback unless required. Comparing results across these models requires interacting with each tool separately, due to differences in their approaches, capabilities, and levels of specialization.

This situation increases the time consumption and resources and forces the maintenance of multiple disconnected workflows, which hinders collaboration, the traceability of changes, and, potentially, security management by exposing sensitive code across various platforms. Furthermore, the lack of a unified environment to work with limits the advanced customization of AI Agents. In light of this scenario, an opportunity was

[1] https://github.com/features/copilot
[2] https://chatgpt.com/
[3] https://openai.com
[4] https://gemini.google.com
[5] https://www.google.com/
[6] https://claude.ai
[7] https://www.anthropic.com/

identified to implement a solution that integrates these concepts into a single system, giving rise to the LAPLACE platform.

AI-powered tools for software development increase developer productivity by providing real-time code suggestions, debugging support, and intuitive natural-language-based coding. These tools speed up software production and reduce development time; however, they also present new challenges (Kanmani et al.,2026).

LAPLACE aims to address this issue by integrating the following concepts into a single platform: 1-Custom AI Agents: knowledge-based language models with the ability to learn and adapt, 2- Centralized Agent and knowledge management: a system with a single OAuth login for repositories such as GitLab and GitHub and 3- Contextual Intelligence (RAG): Implementation of Retrieval-Augmented Generation (RAG) using vector databases to provide agents with contextualized and specific information.

LAPLACE combines relational and vector databases to manage custom AI Agents, indexed knowledge, and chats. It incorporates RAG (Retrieval-Augmented Generation) technology to provide accurate responses tailored to the user's context.

Among LAPLACE's key features is the ability to chat with an AI Agent that uses the user's code as context. Users can associate knowledge (files/repositories) with their AI Agents. Semantic searches are also implemented in vector databases, and the persistence of critical data (users, Agents, chats) is ensured in a relational database.

This paper is organized as follows: Section 2 presents the conceptual framework; Section 3 discusses related work; Section 4 describes LAPLACE, including its requirements, functionality, and architecture; Section 5 demonstrates LAPLACE’s capabilities through the definition of test scenarios; and Section 6 addresses the conclusions and future work.

2 Conceptual Framework

As part of the LAPLACE platform, it is necessary to cover some general concepts related to vector databases, AI Agents, and the RAG technique, which form the foundation of it—that is, they define LAPLACE’s storage, functionality, and use.

Relational databases function properly when the information they store has a well-defined structure. Columns represent characteristics of the data stored in the tables, and indexes enable this information to be retrieved quickly. In contrast, non-relational databases—particularly NoSQL databases—allow data to be modeled and stored without a predefined structure. Both models support exact-match searches.

Meanwhile, vector databases enable semantic similarity searches. The fundamentals of these techniques are described in (Mikolov et al., 2013) and (Reimers, Gurevych, 2019). (Mikolov et al., 2013) describes a method to convert words into vectors that represent the meaning of the word, and (Reimers, Gurevych, 2019) describes a technique that converts sentences—that is, text fragments, phrases, or paragraphs extracted from a source document—into vectors using BERT-type language models (Bidirectional Encoder Representation from Transformers). Vector databases store these sentences as documents, and the vectors are used as indexes. When performing a query, the search question, phrase, or instruction entered into this type of database is

vectorized using the same model used to generate the indexes. The document is then retrieved using vector similarity metrics. In other words, the documents whose vectors are most similar to the query vector are retrieved. These vectors are known as embeddings (Malkov & Yashunin, 2018) and are internal representations of language models that condense the semantics of the document's sentences. Embeddings can be viewed as points in a multidimensional space where each component of the vector represents a coordinate in that dimension. Due to the way embedding-generating models are trained, points with similar semantic characteristics end up clustered in nearby regions of the multidimensional space. With the advent of multimodal models, it is now possible not only to create text embeddings but also to generate them from images, videos, and audio.

Vector databases optimize vector operations to enable efficient searches across tens or hundreds of millions of vectors, each with thousands of dimensions. Algorithms such as HNSW (Hierarchical Navigable Small World) (Malkov et al., 2018), Faiss (Facebook AI Similarity Search) (Douze et al., 2024), and Annoy (Approximate Nearest Neighbors Oh Yeah) (Li et al., 2019) enable the identification of similar vectors without the need to compare all indexes.

LAPLACE also uses LLM-based AI Agents. An LLM-based AI Agent is an artificial intelligence system based on large language models that is capable of using external tools to answer user queries. The most typical example occurs when the language model is asked, "What is today's date?" A standard language model will respond with a date similar to the cutoff date (the dates extracted from the most recent documents in the training dataset). However, an AI Agent will identify the task as one it cannot normally fulfill. At that point, the AI Agent autonomously decides to use an external tool and invokes the operating system's `date` command. It then uses the output of this command to construct a response for the user. The mechanisms required for a foundational language model—models that constitute large-scale AI models pre-trained on vast amounts of general data (Schneider et al., 2024)—to behave in this manner are described in (Luo et al., 2025). Most modern language models that support access to external tools, beyond static knowledge sources, are considered AI Agents, insofar as they must identify how and when to use the available tools to answer user queries. LAPLACE does not provide direct tool access or configuration but allows models that use external tools via API queries. Examples of these models include the gpt-4o[8] and above, Claude Sonnet[9] and Opus 4[10] and above, DeepSeek-r1[11] and above, etc.

A RAG (Retrieval-Augmented Generation) system (Lewis et al., 2020) is a language model that enriches the response it provides to the user by using information retrieved from a vector database via a semantic search. If the language model decides when to retrieve information from the database, then we refer to it as an AI-agent-based RAG system. The system uses the language model to generate a query. This query is vectorized using an embedding generator model, and this vector is used to perform a semantic

---

8 https://developers.openai.com/api/docs/models/gpt-4o

9 https://platform.claude.com/docs/es/about-claude/models/overview

10 https://www.anthropic.com/news/claude-opus-4-7

11 https://cdn.deepseek.com/policies/en-US/model-algorithm-disclosure.html

search in the database. The retrieved documents are incorporated into the context of the language model, and the final response to the user is generated based on these documents.

3 Related Works

In the first half of 2025, coding agents emerged as a category of development tools that were quickly integrated into professional practice. Unlike traditional code autocomplete systems, they allow for the generation of complete pull requests based on a task description provided by the developer. These agents tend to leave more explicit traces in software engineering artifacts, such as co-authorship of commits or pull requests (Robbes et al., 2026). In particular, this section describes GitHub Copilot, a coding assistant, to highlight some of its features and identify areas for improvement.

According to (Sobania et al., 2022), GitHub Copilot is an extension for the Visual Studio Code[12] development environment based on the large-scale language model Codex, which makes automatic code generation available to software developers. This model has been extensively studied in the field of deep learning. It offers a natural language interface that allows users to interact with code and files in a conversational manner.

GitHub Copilot offers three main features: converting comments into code, suggesting tests based on the code implementation, and autocompleting repetitive code (Copilot, 2021) (Nguyen et al., 2022). According to (Reddy, 2025), GitHub Copilot "is a powerful AI pair programmer".

Based on the features observed in GitHub Copilot, it is desirable for this project to propose a document coding and interpretation agent that can be used across multiple repositories, featuring multiple AI Agents with customized knowledge bases, including RAG and chat with specialized agents. The proposal includes unified authentication across multiple repositories, knowledge source management, and multiple indexing of code and documents. It also includes integration of different APIs (beyond Microsoft, needed for GitHub Copilot) and a modular architecture with scalable microservices.

4 LAPLACE. Requirements, functionality and architecture

LAPLACE's functional requirements can be summarized as follows: 1- Unified and centralized access, 2- Customized and specialized AI Agents, 3- Traceable interaction and workflow, and 4- Hybrid data persistence. Each of these is detailed below.

By *unifying and centralizing access*, LAPLACE addresses the issue of fragmented AI tools (Copilot, Gemini, ChatGPT) by providing a single, centralized access point with OAuth authentication, allowing users to view repositories without compromising the security of sensitive code. Additionally, through repository APIs, the code is used as an active knowledge base for interactions with AI Agents.

[12] https://code.visualstudio.com/

Customized and specialized AI Agents aim to enhance the functionality of generic code assistants by allowing users to create and configure them and by associating knowledge (files, documents, or indexed repositories) through RAG techniques and vector databases.

Interaction and traceable workflows are addressed through a conversational chat, contextual code lookup, and in-depth code analysis.

To address the *hybrid data persistent*, the LAPLACE platform implements a hybrid database model that combines a relational database for user records, metadata configuration, and chat logs, and a vector database to store documents indexed by its embeddings and performing large-scale semantic similarity searches.

LAPLACE is implemented using a decoupled layered architecture (or N-tier) (Tanenbaum et al.,2017), in which each layer fulfills a specific responsibility and communicates with adjacent layers through well-defined interfaces.

The main architectural components are: 1- *Presentation Layer (Next.js Frontend):* user interface for interacting with Agents and managing knowledge sources; 2- *Knowledge Logic, Agents, and External Integration Layer (FastAPI Backend):* the core of the application, responsible for orchestrating Agents, RAG logic, user management, and the module for interacting with external APIs (GitHub, GitLab, LLMs), 3- *Data Persistence Layer (Databases & Services):* hybrid storage that ensures the persistence of structured (relational) data and semantic (vector) indexing. Figure 1 details the LAPLACE architecture.

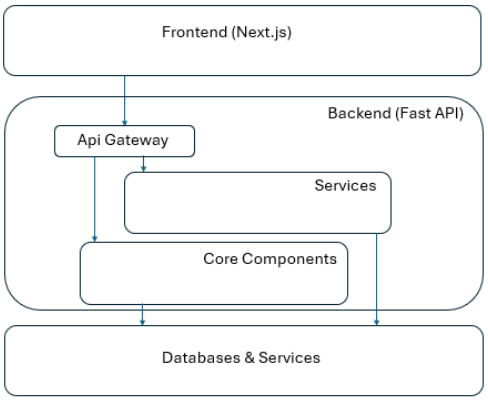

Fig. 1. LAPLACE architecture

The following sections describe each of the layers shown in Figure 1, detailing the components that make up each layer.

4.1 Presentation Layer (Next.js)

The application's front end, developed using Next.js[13], provides a series of user interfaces designed to facilitate various interactions and features, with the aim of optimizing the user experience. The components that make up the front end are: a- *Auth Interface:* login interface for authenticating users. It supports external authentication providers such as GitHub or GitLab, simplifying the login process and leveraging users' existing credentials, b- *Chat Interface:* allows users to interact directly with the platform via a dynamic chat, enabling them to choose from various pre-saved conversations and, if necessary, resume interactions where they left off. Additionally, users can select the specific chat agent they wish to interact with, c- *Knowledge Manager:* allows users to manage documents and information, including uploading new documents, editing existing content, and deleting outdated information, d- *Repo Indexer:* allows you to index code repositories. This feature is tightly integrated with the Knowledge Manager module, enabling the knowledge stored in code repositories to be integrated and managed within the platform. It facilitates the search, organization, and access to the information contained in the source code, e- *Agent Manager:* it allows users to have full control over the configuration of each entity, including creating new agents, editing their settings, assigning their respective API URLs for external integrations, or enabling the default local model. From here, you can associate knowledge with each agent, f- *Analyze Interface:* known internally as "Deep Research," provides advanced features for code and documentation analysis. It enables users to gain insights into their projects, helping them identify patterns, detect potential issues, and improve the overall quality of their code and documentation.

4.2 Backend Layer (FastAPI)

The backend is designed and built using the FastAPI framework[14]. Its architecture is based on a microservices approach that improves scalability and maintainability, allowing for the independent development and deployment of each service. Structurally, the backend consists of the API Gateway, a Services Component, and a Core Components component.

API Gateway, serves as the entry point for external requests coming from the frontend. It acts as a reverse proxy, routing each request to the appropriate backend service. Also, the API Gateway can handle cross-cutting concerns such as basic authentication, token expiration, and request transformation, providing a layer of security and centralized management. **Services** brings together the modules that handle business logic, file processing, and user management. The services it contains are: a- *Indexing Service:* handles the indexing of various sources, including code repositories and different types of documents (txt, doc, csv, pdf). It uses advanced techniques to extract metadata and relevant content from each source for efficient searches and information retrieval. b- *Auth Service*: manages user authentication and authorization of user

[13] https://nextjs.org/

[14] https://fastapi.tiangolo.com/

actions. It implements standard security protocols to verify user identities and control their access to the platform's various resources and features. c- *File Processor:* handles the initial processing of files uploaded by users. Once a file is received, its relevant content is extracted. These files are then persistently stored in a PostgreSQL[15], and the extracted content is indexed in Weaviate[16], enabling advanced contextual information retrieval. d- *RAG Engine (Retrieval-Augmented Generation Engine):* combines information retrieval and natural language generation. It uses the files and indexed content to search for relevant information and employs advanced language models to generate coherent and contextualized responses. e- *Clone/Process Service:* manages code repositories. It is capable of cloning repositories from various platforms and processing their content, extracting the source code and documentation for subsequent indexing and analysis. **Core Components,** contains optimization engines, routing engines, and integration clients. The elements that make up the core are: 1- *BERT Integration:* its main function is to generate embeddings (dense vector representations) of text and expand queries for hybrid search and to capture the contextual meaning of the text; 2- *Cache Manager:* it uses Redis as a caching tool to optimize backend performance, reducing request latency and decreasing the load on services and databases, 3- *Vector Optimizer:* optimizes the generated embedding vectors by applying strategies such as searching for chunks containing the most relevant keywords, ensuring that vector similarity searches are more accurate, 4- *Model Router:* routes incoming requests to the appropriate language model based on the corresponding agent's assignment, so that a response can be generated after contextualization; and 5- *GitHub/GitLab Client:* provides an interface for interacting with the GitHub and GitLab APIs, enabling the retrieval of repository information, issue management, and code cloning.

4.3 Databases & Services Layer

LAPLACE relies on a database infrastructure and external services to perform its indexing, analysis, and search functions. The components of this layer are: 1- *GitHub API and GitLab API:* bridges to source code repositories that facilitate access and in-depth, automated manipulation of their contents; *2- Weaviate:* a vector database with hybrid search capabilities. It stores and manages the generated embeddings, 3- *Redis[17]:* in-memory database that improves platform speed and responsiveness; 4- *PostgreSQL:* a relational database for storing metadata and structured data; and 5- *LLM Agents:* default platform models. It is configured so that if a user creates an Agent and does not specify an external API URL, it automatically uses a local model managed through Ollama[18].

[15] https://www.postgresql.org/
[16] https://weaviate.io/

[17] https://redis.io/
[18] https://ollama.com/

5 Features implemented by LAPLACE and validation tests

To validate the capabilities of the LAPLACE platform, test scenarios were defined based on real-world development interactions. These tests verify the effectiveness of the hybrid architecture (RAG and vector databases) and the Agents' ability to handle complex contexts. The scenarios outlined are: 1- Multi-Repository Analysis, 2- Documentation Generation and 3- Technical Debt Detection and Refactoring.

5.1 Scenario 1: Multi-Repository Analysis.

LAPLACE can search for, map, and understand code dependencies across multiple repositories (GitHub and GitLab) simultaneously. Following, the data source, the test prompt (in Spanish), the expected result, and the result of the test prompt are showed.

Data Source: a repository named A (AvalonLA/atelier) and other named B (nicoxroll/caro-psico)

Test Prompt: "*compara el Hero Section de atelier con el caro-psico. Compara los estilos de colores".*

Expected Result: it is expected to compare the hero section, specifically the colors

Test Prompt Result: Figure 2 shows the result obtained when running the test prompt.

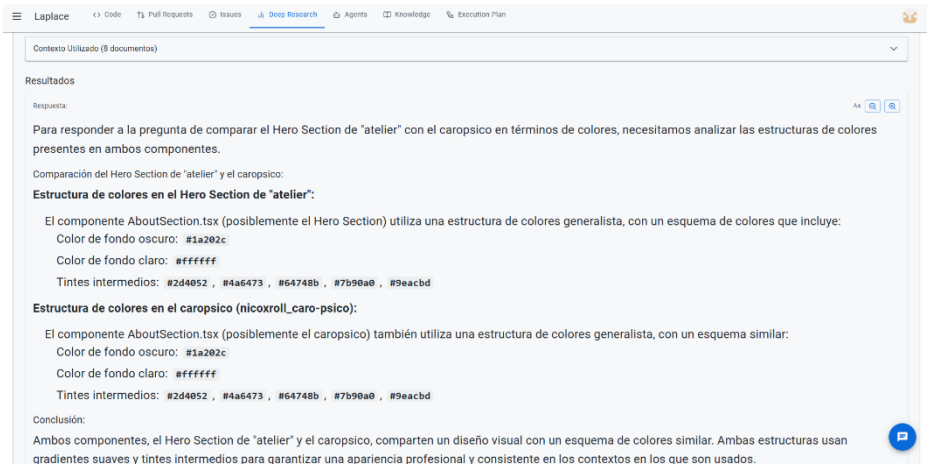

Fig. 2. - Results of the multi-repository analysis test prompt

5.2 Scenario 2: Documentation Generation.

LAPLACE can generate technical specifications or user manuals by directly "reading" and analyzing the source code. Following, the data source, the test prompt (in Spanish), the expected result, and the result of the test prompt are showed.

Data Source: nicoxroll/caro-psico repository.

Test Prompt: "*generar documentación basada en el repositorio caro-psico"*.
Expected Result: generate documentation in the correct format for a specific repository, even when there are multiple associated documents and/or repositories.
Test Prompt Result: for better readability, the prompt's result is split between Figures 3,4 and 5, visually.

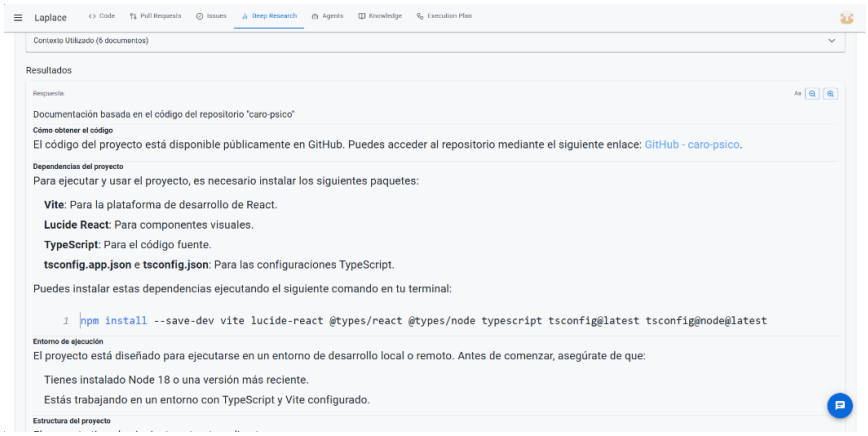

Fig. 3. - Results of Documentation Generation prompt - Part one

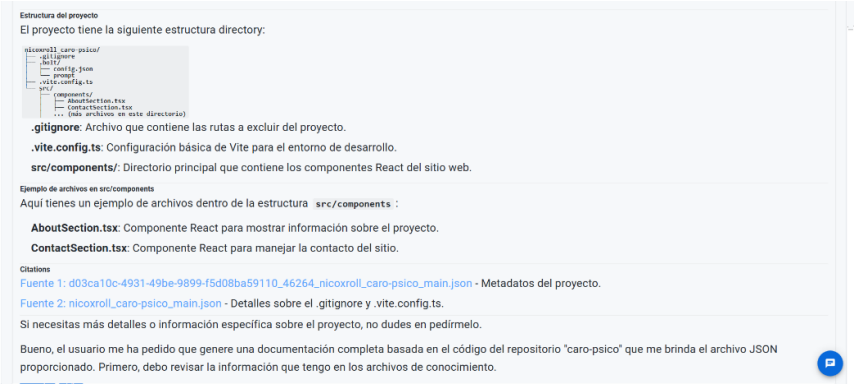

Fig. 4. - Results of Documentation Generation prompt - Part two

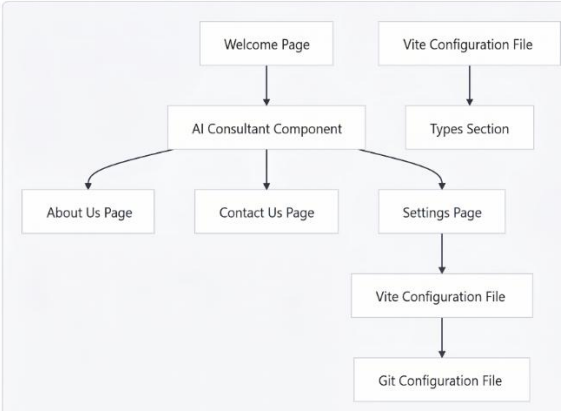

Fig. 5. - Result of Technical Mentoring and Onboarding with Visuals prompt

5.3 Scenario 3: Technical Debt Detection and Refactoring.

LAPLACE evaluates the quality of legacy code and suggests refactoring using the standards and design patterns provided in the knowledge base. Following, the data source, the test prompt (in Spanish), the expected result, and the result of the test prompt are showed.

Data Source: sample test code to simulate a dependency on legacy users from the nicoxroll/public-test[19] repository.

Test Prompt: "*Dada la arquitectura planteada docker-compose.yml de mi repositorio 'public-test'. ¿Qué servicios debería chequear para mantener healthy[20] esta aplicación?"*

Expected Result: the goal is to clearly and concisely identify the services listed in the manifest, which are: Api, Build, Redis, RabbitMQ, Weaviate, Nextauth, Github, Gitlab, bert-service and pythonunbuffered.

Test Prompt Result: for better readability, the prompt's result is split between Figures 6, 7 and 8.

[19] https://github.com/nicoxroll/public-test

[20] The application exposes a specific endpoint (usually /health, /status, or /ping) which, when queried by a monitoring system or a load balancer, returns a successful HTTP status code (such as 200 OK).

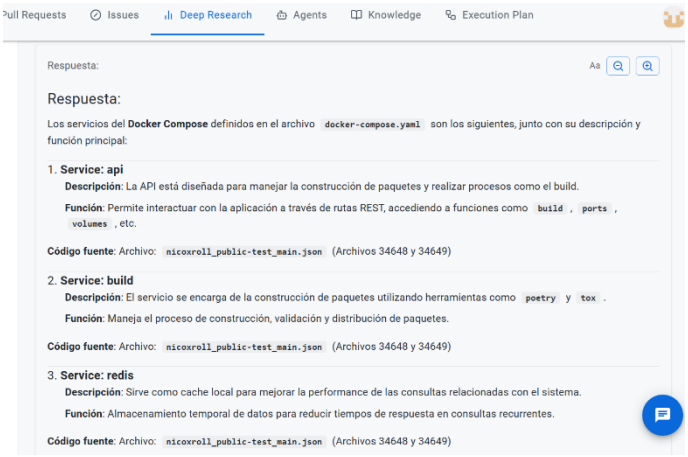

Fig. 6. - Result of Technical Debt Detection and Refactoring prompt - Part one

Fig. 7. - Result of Technical Debt Detection and Refactoring prompt - Part two

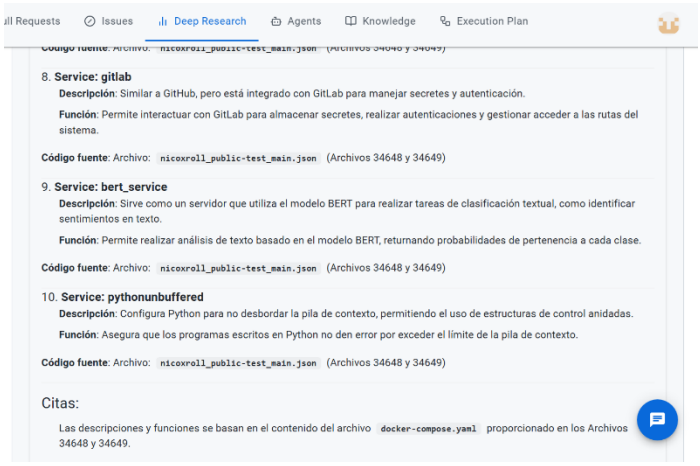

Fig. 8. - Result of Technical Debt Detection and Refactoring prompt - Part three

6 Conclusions and Future Works

This paper presents a platform based on Generative Artificial Intelligence that enables interaction with code repositories through natural language queries. On the one hand, this allows experienced developers to tackle tasks involving code repositories with which they are not fully familiar. On the other hand, it enables novice developers to tackle certain tasks in a guided manner.

LAPLACE's flexibility in configuring different language models allows it to be used in environments with varying characteristics. It also enables the use of more powerful models for tasks requiring greater analytical capability without the need to switch interfaces. LAPLACE is a centralized platform that addresses the issue of fragmentation caused by multiple language models in software development. It integrates multiple APIs (OpenAI, Anthropic, Google) and local models. It optimizes workflows by reducing the time spent managing separate tools. Custom AI agents and RAG-based architecture provide precise assistance tailored to the user's code. The microservices architecture (N-tier) with a FastAPI backend and a Next.js frontend ensures scalability. Additionally, the hybrid integration of relational (PostgreSQL) and vector (Weaviate) databases facilitates metadata management and large-scale semantic searches.

The scenarios presented therein enabled the analysis of multiple repositories, the detection of technical debt, and the generation of documentation tailored to the code.

While related projects focus on making it easier to write individual lines of code, LAPLACE is presented as a comprehensive platform capable of scaling organizational knowledge through the use of other software code repositories.

Based on the experience gained in the scenarios described, the next step will involve validating the proposed features with users (developers) to gather feedback and implement any necessary improvements. Additionally, in the future, we plan to expand the knowledge base by storing the results obtained. We also plan to add the capability for AI Agents to use external tools and store a log of each Agent's requests for future

optimization. We will seek to incorporate the creation of collaborative work environments, whether with other programmers or different Agents. Finally, we expect to expand the scope of the knowledge bases to other projects beyond software code.

References

GitHub. 2021. GitHub Copilot · Your AI pair programmer. https://copilot.github.com.

Kanmani, P., Prabha, K. C., Veerandeswari, J., & Manjula, M. (2026). Generative AI tools for accelerated software engineering. In *Advances in Computers* (Vol. 141, pp. 211-228). Elsevier.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in neural information processing systems, 33, 9459-9474.

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., & Lin, X. (2019). Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, *32*(8), 1475-1488.

Luo, J., Zhang, W., Yuan, Y., Zhao, Y., Yang, J., Gu, Y., ... & Zhang, M. (2025). Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*.

Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, *42*(4), 824-836.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

Nguyen, N., & Nadi, S. (2022, May). An empirical evaluation of GitHub copilot's code suggestions. In Proceedings of the 19th International Conference on Mining Software Repositories (pp. 1-5).

Reddy Vootukuri, N.S. (2025). GitHub Copilot on the Web. In: Vibe Coding with GitHub Copilot. Apress, Berkeley, CA. https://doi.org/10.1007/979-8-8688-2196-7_4.

Reimers, N., & Gurevych, I. (2019, November). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)* (pp. 3982-3992).

Robbes, R., Matricon, T., Degueule, T., Hora, A., & Zacchiroli, S. (2026). Agentic Much? Adoption of Coding Agents on GitHub. arXiv preprint arXiv:2601.18341.

Stuart, R., & Peter, N. (2010). Artificial Intelligence A Modern Approach Third Edition.

Schneider, J., Meske, C., & Kuss, P. (2024). Foundation Models: J. Schneider et al. *Business & Information Systems Engineering*, *66*(2), 221-231.

Sobania, D., Briesch, M., & Rothlauf, F. (2022, July). Choose your programming copilot: a comparison of the program synthesis performance of github copilot and genetic programming. In *Proceedings of the genetic and evolutionary computation conference* (pp. 1019-1027).

Tanenbaum, A. S., & Van Steen, M. (2017). *Distributed systems* (pp. 298-303). CreateSpace Independent Publishing Platform.