FLORIAN ALEXANDRU SERB PETRUSEL

Optimization of RAG Serverless Architectures with VectorDBs and Long-Context models

BACHELOR'S THESIS

Supervised by PEDRO ANTONIO GARCÍA LÓPEZ

Bachelor's Degree in Computer Engineering

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2026

Abstract.

The processing of long documents in Large Language Models (LLMs, artificial intelligence for text) has traditionally relied on Retrieval-Augmented Generation (RAG, which extracts relevant fragments from a vector database) to overcome their limited memory. Currently, long-context models allow processing complete manuals in a single request.

Processing the entire document prevents information loss due to fragmentation, but increases response time and the cost per consumption of basic text units (tokens). This poses an architectural challenge: at what point is it more efficient to use RAG versus long context?

To resolve this, this Bachelor's Thesis compares three architectures in a serverless environment (AWS Lambda): modular RAG (control over extraction and the vector database), managed RAG (indexing handled by the provider), and long context (sending the full document). Using a language model as an automated evaluator (LLM Judge), accuracy, latency, and operational cost were analyzed.

The results demonstrate that there is no single optimal architecture. As a solution, a smart router is developed to analyze each query in real time and route it to the appropriate architecture. This hybrid system ensures the high accuracy of long context for complex questions, leveraging the speed and low cost of RAG for all other operations.

Table of Contents

List of tables

List of figures

List of code snippets

List of equations

1 Introduction

The integration of large language models into software development requires systems capable of retrieving and processing vast volumes of information. Historically, the primary limitation of these models resided in their short-term memory, known as the context window. To overcome this constraint, the industry standardized the Retrieval-Augmented Generation (RAG) architecture. This technique fragments texts, converts them into a mathematical format, and stores them in vector databases, transmitting only the relevant fragments to the model during each query.

This approach presents structural limitations when dealing with complex documents, as fragmentation causes information loss by separating linked data across different pages. As an alternative, recent long-context models allow for the processing of complete manuals in a single request. Industry experts, such as Nicolas Bustamante in his article *The RAG Obituary* (2025) [14], argue that full access to the document improves reasoning and could potentially replace RAG systems. However, transmitting all the information significantly increases latency and computational cost due to the consumption of basic text units. Research such as LaRA [12] demonstrates that there is no single perfect solution when deciding between RAG and long-context approaches. Likewise, other studies [13] highlight the technical and performance challenges entailed in processing massive amounts of information. This poses a clear architectural problem.

To address this design challenge, this Bachelor's Thesis establishes the following main objectives:

- Evaluate a modular RAG architecture, integrating cloud-based vector databases (Pinecone) to maintain direct control over data fragmentation and retrieval.
- Evaluate a managed RAG architecture with Google Cloud (Gemini File Search), delegating the complete indexing and search cycle to the provider's infrastructure.
- Evaluate a long-context architecture, processing the information exhaustively by sending the complete document in each query.
- Audit the responses using the LLM-as-a-judge methodology, utilizing a language model as an automated evaluator to measure accuracy and detect fabricated results.
- Analyze the overall performance, extracting data and generating comparative graphs that allow contrasting the results of each architecture.
- Design an intelligent router capable of analyzing the user's query in real-time and routing the execution to the most efficient architecture, laying the groundwork for transitioning the system to a production environment.

To achieve these objectives, the technical implementation is proposed through a dual approach. In a primary phase, the workflows are modeled using a low-code automation platform (n8n) to evaluate the viability and deployment agility of the architectures. Subsequently, to ensure result accuracy and total control over the data, the definitive testing framework is fully deployed in a serverless computing environment (AWS Lambda). Operating natively through code ensures direct access to the provider's interfaces, avoiding the interferences or limitations of intermediaries and guaranteeing the acquisition of real-world results.

The experiment is based on a real technical document: the workshop manual for the BMW 3 Series E46 automobile. Additionally, different document formats will be

incorporated; specifically, a PDF without a table of contents, in order to verify whether the long-context model suffers from information loss in the middle of the document.

Finally, all tests will operate under real billing services to extract exact data regarding operational costs and latency times. This will allow for the establishment of a precise comparison, evaluating development convenience against infrastructure control and the actual consumption of resources.

2 Project overview

This chapter presents the contextual framework and technical motivations that justify the development of this Bachelor's Thesis. Before addressing the design of the proposed architectures, it is essential to understand the system's operational environment. The following sections analyze the current state of technical document management, define the technical challenges regarding the limitations of standard artificial intelligence systems, and, finally, justify the need to implement an intelligent router capable of optimizing computational resources.

2.1 Context and current state of information retrieval

The management of extensive technical manuals, such as workshop guides and maintenance protocols, presents a structural challenge due to their typical storage in static formats (such as PDF). Traditionally, information retrieval within these documents has relied on lexical search engines (exact keyword match searches, e.g., CTRL+F). This approach is inefficient and prone to errors, as it lacks contextual understanding. The user locates the word in isolation but is forced to review the preceding text or the section title to confirm whether that fragment corresponds to the procedure they were actually seeking, which hinders the rapid location of precise data.

The advent of LLMs (Large Language Models) enabled interaction with information through natural language. However, in their early stages, they presented two technological barriers: they lacked access to private documentation and possessed a highly limited context window (the amount of information the model can retain and process in its short-term memory during a query). To overcome this, the technology sector standardized the RAG paradigm. This architecture first retrieves the most relevant text fragments from the document and provides them to the model to draft a response. In this way, responses are grounded in real data, mitigating the risk of hallucinations (the generation of false, incoherent, or fabricated information by artificial intelligence due to its pre-training).

Recently, the technology has experienced a leap with the introduction of long-context models, such as Gemini 1.5 Pro [15], equipped with context windows capable of processing millions of tokens (the minimum unit of information a model processes, roughly equivalent to a syllable or word) simultaneously. This raises the possibility of dispensing with intermediate search engines and transmitting the complete technical manual in each query to maximize overall comprehension. Nevertheless, this approach introduces questions regarding computational cost and latency (the total waiting time until the system's response is received), thereby motivating a comparative analysis.

2.2 Reliability and optimization requirements

In the vehicle maintenance environment, the volume of technical documentation has grown to levels that are difficult to manage manually. Repair centers handle vast amounts of unstructured information distributed across workshop manuals and electrical wiring diagrams spanning thousands of pages. The traditional workflow is highly inefficient, as it forces technical personnel to halt their physical activity to search for highly specific data on computer terminals, such as the electrical diagram of a control module or the resistance values of a specific sensor for a particular model. This difficulty increases the mean time to repair (the average time required to identify and resolve a fault) and reduces the productivity of the facilities.

To mitigate this situation, the use of artificial intelligence-based assistants is an ideal solution, provided that maximum accuracy is guaranteed. In mechanical engineering, an error in interpreting a tolerance value or an assembly sequence can cause severe failures; therefore, the system must adhere to the principle of zero hallucinations (preventing the artificial intelligence from generating false or fabricated information). Consequently, the design relies on context grounding, a technique where the model is instructed to respond exclusively using the text fragments extracted from the manual. Furthermore, a fail-safe mechanism (a configuration that forces the system to admit it cannot find the information rather than fabricating it) is incorporated to prevent the model from resorting to its general prior knowledge.

Traceability (the ability to pinpoint exactly which part of the document the information originates from) is an indispensable requirement in this project. If the model responds correctly using data from its pre-training rather than extracting it from the provided manual, knowledge leakage occurs, compromising technical reliability. The objective is for the solution to be swift, reducing search times from minutes to seconds, while maintaining referential integrity to the manufacturer's original document. In this way, the acquisition of precise responses based strictly on the manual is ensured through a rapid and low-cost process.

2.3 Balance between resources and response quality

When designing an artificial intelligence-based information retrieval system, a technical tension exists between the use of architectures that divide the text into fragments and those that process complete documents. The use of RAG systems requires a more complex initial configuration to manage vector databases (information repositories that organize text according to its semantic meaning), such as Pinecone [16]. In return, this method allows for rapid responses with minimal processing expenditure. On the other end of the spectrum, long-context models allow the complete manual to be sent in each query, simplifying system design but considerably increasing response time and computational resource consumption.

Because there is no single ideal solution for all cases, this work is grounded in the necessity of conducting an evaluation using real data from the BMW workshop manual. The objective is to accurately measure the performance of each architecture to understand when each is most efficient. In a real-world environment, queries vary in difficulty: a question regarding a specific technical datum, such as tire pressure, does not demand the same processing effort as a complex diagnosis that requires correlating information across multiple chapters. Utilizing the model's maximum capacity for simple queries is inefficient, whereas using limited fragments for problems requiring a comprehensive overview may reduce the quality of the response.

As a solution to this problem, the project implements an intelligent router. This module analyzes the complexity of the query and automatically decides which path to take. Direct questions are resolved using the vector database to prioritize speed, whereas queries requiring deep analysis are processed using the complete manual to guarantee maximum comprehension. This hybrid architecture allows the system's operation to be optimized, ensuring precise and reliable responses with the lowest possible resource utilization.

3 Theoretical framework

To understand the design decisions evaluated in this study, it is necessary to analyze the operation of large language models from a technical perspective. The following sections describe the limitations of their working memory, the principles that enabled the standardization of Retrieval-Augmented Generation through the segmentation and mathematical conversion of data, and the advancements that have facilitated the emergence of models capable of simultaneously processing extensive volumes of information. Finally, the current methods for automatically measuring and validating the performance of these models are presented, concluding with the technical justification for the architectures selected for the comparative evaluation.

3.1 Memory limitations in large language models

At the core of any modern natural language processing architecture lies the process of dividing text into computational units known as tokens. Large language models do not process information letter by letter; rather, they fragment the user's input into these units, which may represent complete words, syllables, or characters depending on the complexity of the language. The fundamental limit of these systems' working memory is defined as the context window, which establishes the strict maximum number of tokens that the model can simultaneously retain, process, and correlate during a single execution. In terms of computer architecture, the context window acts similarly to the algorithmic model's RAM [17], limiting both the information provided in the query and the generated response.

Historically, the inability of these models to process extensive technical documents, such as workshop manuals spanning hundreds of pages, was not a storage problem but rather a structural mathematical limitation. Current models are built upon the Transformer architecture, originally introduced in 2017 [1]. The central component of this design is the self-attention mechanism, which enables the system to comprehend the meaning of a text unit by observing its relationship with all other units in the sequence. However, the computational cost of this original mechanism scales quadratically, which is represented by the complexity notation $O(n^2)$, where n is the length of the token sequence. This means that if the amount of input text is doubled, the processing and memory consumption of the graphics card are multiplied by four, creating a technical bottleneck when dealing with large volumes of data.

The first generation of commercialized models possessed highly limited context windows, typically ranging between 4,096 and 8,192 tokens, which is equivalent to approximately 20 pages of text. If a user attempted to input a technical manual, which can reach 300,000 tokens, the quadratic memory requirement of the attention matrix would collapse the infrastructure, returning a saturation error due to an exceeded context limit. This physical impossibility of processing extensive documentary bases in a unified manner was the technological catalyst that forced the development and adoption of architectures based on fragmented information retrieval, consolidating the standard of Retrieval-Augmented Generation [6].

3.2 Modular RAG architecture

To resolve the processing problems analyzed previously, the industry has standardized the Retrieval-Augmented Generation architecture. In its modular or custom variant, the engineer maintains full control over the information flow, independently selecting the conversion model, the database, and the generative language model. This scheme is not a

model in itself but a design pattern that divides the process into two stages: knowledge ingestion and meaning-based or semantic retrieval [18].

Because processing an extensive document in its entirety is unfeasible for the working memory of the models, the first phase consists of text chunking or division into blocks. This process divides the original document, such as a workshop manual spanning hundreds of pages, into small, manageable segments. To prevent information from being cut off and losing its meaning between fragments, overlapping techniques are applied. This allows the end of one block to contain the beginning of the next, ensuring that the continuity of structural information and context is maintained between the divided parts.

Once the document is fragmented, it is necessary for the computer system to be able to interpret the meaning of the text, since traditional databases operate through exact keyword matching, which is inefficient if the user employs terms different from those in the manual. To solve this, each text fragment is processed using an embedding model, which is a specialized neural network that translates human language into a mathematical language of numerical vectors. Technically, the text is projected into a continuous vector space represented as $\mathbb{R}^d$, where (d) is the number of dimensions. For instance, advanced models developed by OpenAI utilize high-density vectors with 1536 or 3072 dimensions.

Thus, text fragments with similar meanings are located close to each other within this multidimensional space [2]. To determine which fragment best answers a query, cosine similarity is used, a metric that calculates the angle between the vector of the user's question and the vectors of the manual's fragments using the following formula:

$$\text{Cosine similarity} = \cos(\theta) = \frac{A \cdot B}{||A||||B||} \qquad \textbf{(1)}$$

If the result of this operation is close to one, it means that the angle between the vectors is minimal and, therefore, there is a high coincidence in the meaning of both texts. Storing and comparing millions of these vectors in real-time requires the use of vector databases, such as Pinecone. Unlike conventional storage systems, these databases utilize approximate search algorithms, notably the Hierarchical Navigable Small World (HNSW) algorithm. Its operation is similar to searching for an address using a road network. The algorithm organizes all text fragments into different overlapping levels. The top level functions like a network of highways, connecting points that are far apart. This allows the system to make large leaps and quickly approach the area where the answer is located, discarding millions of irrelevant data points at once. Once in the correct area, the algorithm descends to the lower levels, which act like highly dense local streets, to navigate slowly among the most similar texts until the exact fragment is found. This tiered navigation structure is what allows information to be located in milliseconds without having to scan the entire database [3].

The combination of these elements is what allows the system to bypass the limitation of the quadratic complexity, $O(n^2)$, that affects language models. Instead of attempting to process the thousand pages of a manual simultaneously, which would collapse the system's memory, the vector database acts as a filter that extracts only a small number of relevant fragments, identified as (k). By sending only these specific paragraphs to the language model as supporting context, the volume of processed information remains small and constant. This reduces the computational effort and guarantees rapid responses, making the use of artificial intelligence on large volumes of technical documentation viable without incurring excessive hardware costs.

3.3 Managed RAG architecture

As the adoption of generative artificial intelligence accelerated within technological development environments, major technology providers identified that the barrier to entry for implementing a custom retrieval system was high. It required specialized data engineering teams to manage text fragmentation, mathematical conversion to vectors, and the maintenance of specialized databases. To solve this problem, integrated retrieval solutions emerged, operating under the software-as-a-service model. In this approach, the provider completely abstracts the complexity of the data processing workflow. The developer only needs to upload the original document, such as a text file or PDF, to the provider's platform, and the latter automatically manages the extraction of information, division into blocks (chunking), generation of numerical vectors, and their storage in an internal index that remains invisible to the system creator [19].

The primary strength of these architectures lies in their initial operational efficiency. By delegating the management of the underlying infrastructure to the provider, the engineering team reduces development time and eliminates the high initial investment costs for servers or specialized databases. Furthermore, these solutions typically incorporate conversational state management by default. This means that the system automatically remembers the information retrieved in previous interactions, allowing the user to formulate chained questions without the engineer having to manually program an intermediate memory module.

However, in industrial environments such as the automotive sector, this extreme simplification introduces a design risk: algorithmic opacity. These integrated solutions operate as a closed or opaque system for the developer [8]. By utilizing this approach, the software engineer loses absolute control over the system's operational settings. It is not possible to review or modify the exact size of the text fragments, the degree to which they overlap, the mathematical model used for vector conversion, or the exact number of fragments the system extracts to answer each query.

This lack of control becomes a critical obstacle during the troubleshooting phase. If, for example, the model is unable to find the exact data regarding the torque required to tighten an engine part, the developer lacks the tools to diagnose whether the failure occurred because the internal algorithm failed to interpret a table in the document, or if the mathematical similarity threshold was poorly configured by the provider. Finally, the exclusive use of packaged infrastructure generates a high level of technological dependency, which hinders the possibility of migrating the system and its knowledge bases to other open-source platforms in the future.

3.4 Long-Context architecture

As detailed in previous sections, the classic design of language models presented a primary limitation: the quadratic complexity increase in processing effort due to their internal mathematical attention mechanism. However, the field of artificial intelligence has recently achieved a notable breakthrough with the advent of long-context models, such as Google's Gemini family (1.5 Pro and later versions). These novel architectures have succeeded in expanding the historical reading limit from a few thousand pages of information to windows capable of processing between one and two million of these data units simultaneously [20]. From an engineering perspective, this achievement was not obtained merely by increasing computational power over the original design, but by applying advanced algorithmic innovations. The most recent models utilize an organizational structure known as Mixture of Experts. With this method, instead of activating the entire

neural network to analyze each word, the system routes the query exclusively to the most appropriate sub-modules for resolving that specific topic. Furthermore, the use of techniques such as ring attention allows the mathematical computation to be divided into blocks and distributed across multiple processors simultaneously, thereby preventing system collapse [7].

Despite these improvements in software design, the physical reality of data centers imposes significant constraints. When a model processes a complete document, such as inputting a 300-page manual in a single user query, the system must calculate and retain in its temporary memory the mathematical representations of all those words in order to generate the response. This temporary storage is known as the Key-Value Cache. For a request of one million information units, this memory can occupy tens of gigabytes per concurrently connected user. Given that these models require an exceedingly high data transfer rate to avoid halting the text generation process, providers are compelled to utilize highly specialized processing units and graphics cards [21]. These components rely on high-performance memory that is both costly and complex to manufacture, which implies that the capability to read a complete document at once relies on a physical infrastructure operating at extremely high levels of technical and energetic demand.

This reliance on advanced infrastructures introduces two major drawbacks in professional environments: latency and operational cost. On one hand, processing and loading hundreds of thousands of data points into memory delays the initial latency, that is, the time it takes the system to begin returning the first word of the response. Whereas a Retrieval-Augmented Generation system returns information in milliseconds, transmitting the complete manual can delay the start of inference by 10 to 30 seconds, which negatively impacts the user experience. On the other hand, high memory consumption drives up operational expenses. Providers bill for the use of their services based on the volume of information inputted; therefore, sending 300,000 fragments in each interaction of a conversation generates a substantially higher cost than sending only the 2,000 most relevant fragments, as the traditional modular architecture would do. These physical and economic barriers confirm that the exclusive use of long context is not always efficient, justifying that the combination of both technologies through an intelligent router is a technical necessity to guarantee the project's viability.

3.5 LLM-as-a-Judge testbench

Unlike classical software engineering, where unit tests verify fixed and binary results, such as a success or a failure, large language models operate under a probabilistic nature. The same user query can generate multiple responses that, although varying in their words or structure, are equally valid. This poses an architectural challenge: how to measure with scientific rigor the accuracy of a technical information retrieval system. To evaluate the accuracy of any predictive system, a reliable reference point is required, known as ground truth or reference response. In the context of an automotive manual, this reference is not the general knowledge the artificial intelligence might possess regarding mechanics, but rather the exact and literal text fragment extracted from the manufacturer's original document. Any deviation, data omission, or fabrication of external information (a phenomenon known as hallucination, where the model resorts to data from its pre-training), will be considered a critical system error, regardless of how well-written or confident the response may appear.

Historically, the validation of machine-generated texts relied on human evaluators. However, various studies have demonstrated that human supervision over extensive technical documents is difficult to scale and presents notable objectivity issues. One of the most detrimental to a technical audit is human bias [4]. This phenomenon causes evaluators

to insufficiently penalize data errors if the artificial intelligence responds with a formal and grammatically correct tone. Furthermore, when auditing hundreds of responses against a manual spanning hundreds of pages, the reviewer's mental fatigue rapidly degrades the consistency of the evaluation: a response scored positively in the morning could receive a deficient rating in the afternoon under the exact same criteria and by the same person.

To resolve this lack of objectivity and the high time cost of manual auditing, the industry has adopted a methodology that utilizes an advanced language model as a judge or evaluator. This system employs a state-of-the-art model configured with a zero degree of randomness, technically known as zero temperature, to maximize its level of determinism and ensure it acts exclusively as an analytical referee [5]. In this process, the evaluating model is provided with a strict grading instruction, the user's original question, the response generated by the system under test, and the ground truth extracted from the manual. The model mathematically cross-references the response with the reference text and outputs a numerical score accompanied by structured reasoning that justifies the grade.

This cross-evaluation architecture guarantees high scalability, a vastly reduced execution cost compared to the use of human auditors, and stable algorithmic consistency, currently consolidating itself as the standard method for automatically validating artificial intelligence systems prior to their deployment in production.

3.6 Justification of the technological comparison

Following the analysis of the theoretical foundations of information retrieval systems, a recurring challenge emerges in the development of such software: striking a balance between the ingestion of a complete document and the computational resources required for its processing. Architectures that divide text into small fragments have constituted the standard operational methodology; however, novel language models, capable of retaining an entire document within their memory, now offer viable alternatives.

Given the absence of a clear industry consensus regarding the optimal use cases for each approach, the necessity for this project arises: to conduct an empirical comparison. The objective is to evaluate the three primary options previously detailed, modular architecture, integrated architecture, and long-context processing, utilizing real-world technical manuals from the automotive sector.

This comparison will serve to examine the behavior of each technology under identical conditions. It should be noted that the quantitative results of this testing, alongside the operational costs and the final performance validation, will be presented later in the project's evaluation phase, subsequent to the system's construction.

The purpose of this preliminary test is not merely to observe which technology performs best, but to generate the requisite data for designing the core component of the software: the intelligent router. This component will be tasked with automatically determining the optimal technology to process each user query, based on the insights acquired during the preceding comparative analysis.

In order to systematically develop both the initial testing environment and the final router, it is first necessary to define the requisite system specifications and the profiles of its intended users. These specifications are detailed below in the requirements chapter.

4 Requirements

Having established the technological context and the necessity to compare the different architectures, this chapter defines the formal specifications of the software to be developed.

To encompass the entirety of the project in a systematic manner, this section is divided into two main blocks. The first block details the requirements of the testing environment, which is necessary to conduct the comparison among the different technologies. The second block establishes the requirements for the system intended for the production environment, focusing on the operation of the intelligent router. Through the identification of the actors and the specification of both functional and non-functional requirements for both phases, the guideline that will steer the project's design is established.

4.1 Comparative testbench requirements

To guarantee that the comparison among the different artificial intelligence architectures is fair and detailed, it is first necessary to construct an automated evaluation environment. The purpose of this environment is not to serve an end-user, but to subject the different models to an examination using the vehicle manual in order to record their performance. The specifications of this testing phase are detailed below.

4.1.1 Identification of actors

To model the interaction with the system during the testing phases and define the software's responsibilities, two primary profiles have been identified. Unlike traditional programs, where users are typically exclusively human, this project integrates logical systems that operate autonomously.

Research Administrator: This is the sole human actor in this phase. It represents the engineer responsible for designing the experiment. Their primary objective is to provide the list of technical questions regarding the automotive manual and to trigger the mechanism that initiates the automated testing. This actor does not interact with the responses individually but rather analyzes the consolidated final results.

Impartial Evaluator Model: This is an algorithmic actor exclusive to the experimentation environment. It is implemented through a language model from another provider, in this case, the GPT-4o model, configured under parameters of strict rigidity, eliminating its capacity for invention. Its function is to receive the responses generated by the architectures under examination, cross-reference them against the exact answer extracted from the manual (known as the ground truth), and issue an objective numerical score justifying whether the information is correct or contains errors.

4.1.2 Functional requirements

Functional requirements define the specific actions that the software system must execute to enable the comparative evaluation among technologies. They have been classified into three fundamental processes:

FR-01: Document Preparation Process: The system must be capable of processing the original technical document in PDF format. For the modular architecture, the system must extract the text, divide it into small fragments, convert these fragments into mathematical vectors using an external conversion service, and store this data in a vector database. For the integrated architecture, the system must allow the upload of the complete file to the primary provider's servers so that it can prepare its own internal search index. Conversely, the long-

context architecture does not require this prior preparation process, as the complete document will be attached and processed in real-time during each query.

FR-02: Automation of Response Generation: The system must allow the execution of tests in a sequential and block-structured manner. To achieve this, the program must read a list of twenty questions from a cloud-hosted spreadsheet. The process is not executed in parallel nor by alternating between technologies; rather, the system evaluates one architecture entirely before advancing to the next. First, the modular architecture block will be processed. For each iteration, the system will read a question, send it to the model, receive the response, and record the generated text directly in the corresponding column of the document before proceeding to the next query. Once this iterative cycle and individual saving for the twenty questions is completed, the system will repeat the exact same procedure with the integrated architecture and, finally, with the long-context architecture.

FR-03: Algorithmic Evaluation Module: Following the response generation phase, the system must enable the processing of results through code scripts programmed in the Python language. This code must connect to the spreadsheet to sequentially extract the original question, the ground truth that serves as a grading guide, and the response generated by the system. Next, the program must package this information along with a detailed instruction and send it to the impartial evaluator model so that it can rigorously analyze the result. Finally, the code must record the assessments dictated by the algorithmic judge, adding the results directly to the document using two specific columns: one for the numerical score and another for the textual reasoning that justifies said grade. This evaluation, extraction, and recording process will be executed sequentially and independently for each of the analyzed architectures.

4.1.3 Use cases

The use case model defines the high-level interactions between the actors and the software itself during the analysis phase. As shown in Figure 1, this diagram represents the boundary of the automated evaluation system and describes how external actors interact with the program's primary functions. In this schema, the research administrator acts as the engine that initiates the actions, while the impartial evaluator model is positioned as a necessary participant to complete the technical analysis of the responses.

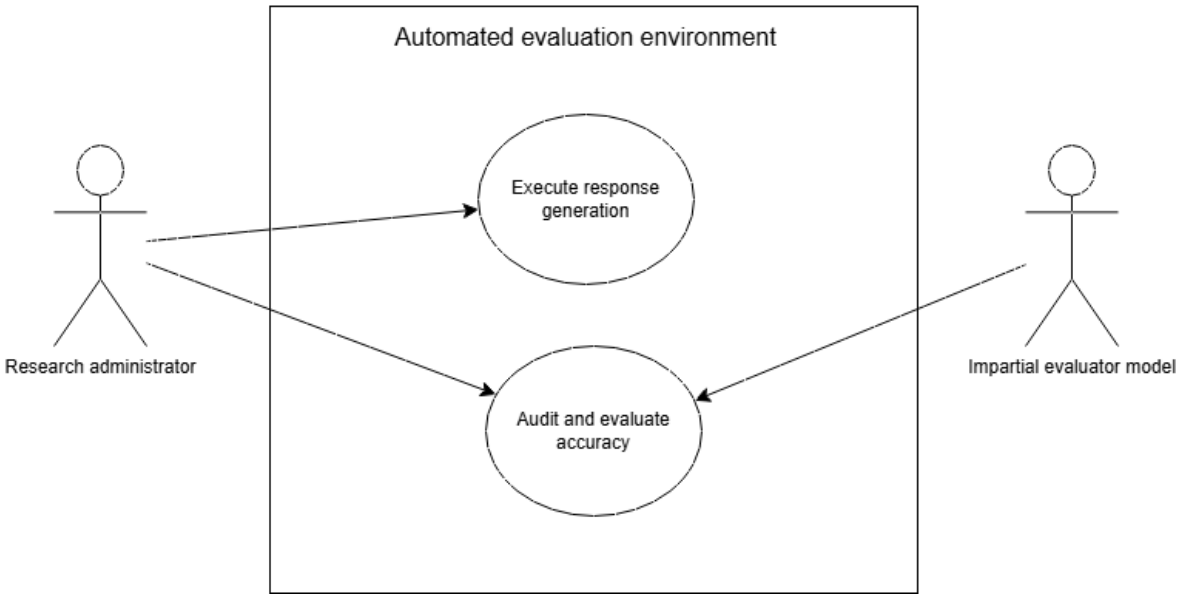

Figure 1. Automated evaluation use case diagram.

To understand the dynamic behavior of the system, the two primary workflows of this stage are documented below, detailing for each the responsible actor, the necessary preconditions, and the sequential execution steps:

UC-01: Generation Execution: This process serves to obtain the study data. The primary actor is the research administrator. As a precondition, the list of twenty questions must be prepared in the cloud spreadsheet, and the models must be operational. The main flow begins when the administrator initiates the process on the coordination platform. Subsequently, the system initiates the evaluation in a structured, block-by-block manner, beginning with the modular architecture. In a continuous cycle, the system reads a question, sends it to the model, receives the response, and saves it directly in the corresponding column of the control document before advancing to the next query. Once it finishes processing and individually saving the twenty questions for this first technology, it repeats the exact same complete cycle for the integrated architecture and, finally, for the long-context architecture.

UC-02: Accuracy Audit and Evaluation: This is an analytical process where the model assumes the role of a judge to calculate the reliability of the responses. The primary actor is the impartial evaluator model. As a precondition, the responses from the different architectures must have been generated and recorded in the previous step. The flow begins when the administrator executes the programmed code scripts. Previously, the system must have the spreadsheet document locally, and, sequentially for each architecture, it extracts the original question, the ground truth, and the response generated by the artificial intelligence. Next, the program packages this information along with a detailed grading instruction and sends it to the evaluator model. Finally, the latter issues its verdict, and the code records the result, directly adding to the spreadsheet a numerical score and a textual reasoning that justifies the obtained grade.

4.1.4 Non-Functional requirements

Non-functional requirements establish the constraints and quality characteristics that the testing environment must meet to ensure the validity of the results and the feasibility of the test.

NF-01: Determinism in Evaluation: To ensure the accuracy of the audit, the model acting as a judge must operate with a zero degree of randomness, a concept technically known as zero temperature. This configuration guarantees that, given the same response and the same reference text, the evaluator always assigns the identical score. This adjustment is fundamental for the system to be much more rigorous in detecting cases where the artificial intelligence fabricates data based on its prior knowledge instead of strictly limiting itself to the manual's content. By eliminating any margin for creativity or free interpretation, the judge can more precisely identify responses that are not supported by the original document, strictly penalizing any information the model has generated on its own that does not appear in the technical manual.

NF-02: Error Management and Fail-Safe Stop: The system design must prioritize data reliability over the uninterrupted continuity of the process. In the event that an error is detected, such as incorrect data in the source spreadsheet or a failure in the models' responses, the program must execute an automatic fail-safe stop. This strategy prevents the system from continuing to consume processing resources and time unnecessarily when the final result will no longer be valid. In this way, it is guaranteed that erroneous values are not saved in the control document, allowing the administrator to correct the problem and ensure the quality of the research before restarting the process.

NF-03: Traceability and Information Persistence: The system must ensure that all information generated during the experiment is permanently preserved and not lost in the program's temporary memory. In addition to the response texts and processing times measured in milliseconds, it is necessary to record the number of text units processed in each query and response. This data, which represents the volume of information the language model handles in each interaction, is necessary to subsequently calculate the approximate economic cost of the system's execution. All these parameters, along with the judge's assessments, must be persistently recorded in the spreadsheet. This allows the administrator to manually review the process and guarantees that the data is available for subsequent statistical analysis without the risk of loss.

4.2 Requirements of the intelligent router in production

The objective of this component is to act as an intelligent intermediary that receives user queries and automatically and transparently decides the optimal technology to resolve each question based on its complexity.

4.2.1 Identification of actors

Two primary profiles that interact to resolve technical queries regarding automotive manuals have been identified:

End User: Represents the human actor, typically a mechanic or an engineer. Their function is to formulate natural language questions regarding vehicle maintenance or diagnostics. Their expectation is to receive a precise and rapid response without needing to understand how this information was analyzed and extracted from the document.

Intelligent Router: This is the actor and the logical core of the project. It acts as an autonomous coordinator that receives the user's query, analyzes its intent, and decides which architecture the request should be routed to in order to obtain the best possible result. Its responsibility is to balance response accuracy with latency and resource utilization, even if the router determines it is necessary to attach the entire document for a more precise response at the expense of increased processing time and resource consumption.

4.2.2 Functional requirements

The functional requirements in this phase describe the behaviors the system must exhibit to efficiently serve the end user:

FR-01: Automatic Intent Classification: The system must receive the natural language query and evaluate its complexity using a classifier model. The router must categorize the question into one of three predefined states: direct technical queries, comparative analyses between sections, or complex problem resolution requiring the synthesis of multiple parts of the manual to achieve a comprehensive result.

FR-02: Conditional Flow Routing: Based on the aforementioned classification, the system must automatically route the query to the most appropriate architecture. If the question is straightforward, the modular path and vector database will be utilized; if it is comparative, the provider-managed search will be employed; and if it is procedural, long-context processing will be activated.

FR-03: Session Memory Management: The system must be capable of retaining the context of the user's previous interactions. This allows the operator to ask follow-up

questions without the need to reiterate all prior information, thereby facilitating a more natural technical dialogue.

FR-04: Processing of Attached Documents: In cases where the query is highly complex and the long-context architecture is required, the system must allow the user to attach the manual's PDF file. The program must verify the presence of the file and, if absent, issue a prompt requesting the document in order to proceed.

FR-05: Communication and Response Interface: The system must feature a chat interface that abstracts the internal complexity. All responses, regardless of the generating technology, must be presented in a unified and clear manner, ensuring that the end-user receives the technical information in a structured format.

4.2.3 Use cases

The use case model for the production phase defines how the user interacts with the software to resolve their queries. As detailed in Figure 2, this diagram establishes the boundaries of the intelligent router system and illustrates the direct relationship between the user's request and the program's internal dependencies.

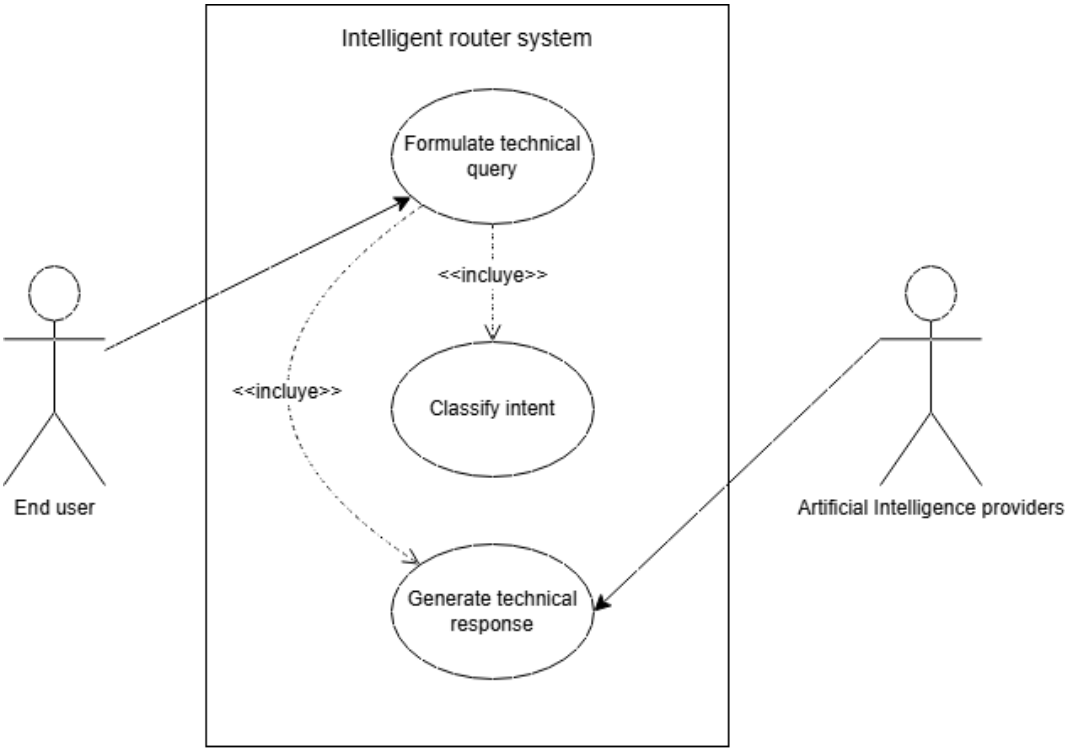

Figure 2. Use case diagram of the system in production.

To understand the system's dynamic behavior in a real-world environment, this primary workflow is documented below, detailing the responsible actors and the execution steps:

UC-01: Formulate Technical Query: This is the system's primary and sole outward-facing process. The primary actor is the end-user, and the secondary actor encompasses the artificial intelligence providers. As a precondition, the system must be operational and connected to cloud services. The flow begins when the user accesses the communication interface and inputs a technical question regarding the vehicle manual. To satisfy this request, the system internally executes two mandatory processes, represented in the diagram by inclusion relationships. First, the program includes the task of classifying the intent,

where the intelligent router analyzes the question to select the appropriate architecture. Subsequently, it includes the task of generating the technical response, at which point the system routes the request to the external service so that the artificial intelligence providers can process the information. Once the response is obtained, the system standardizes the text and displays it to the user through the conversational interface, thereby closing the query cycle.

4.2.4 Non-Functional requirements

These requirements establish the conditions of quality, performance, and security that the system must satisfy to be reliable and viable in a production environment.

NF-01: Optimized Response Time: The system must minimize processing times for the end-user. For direct queries, the response time must be as short as possible to maintain conversational fluency. Conversely, for procedural queries that require reading the entire manual, a longer time is acceptable due to the large volume of information to be processed.

NF-02: Reliability and Information Grounding: The system must prioritize technical accuracy over creativity at all times. The system must be configured to strictly utilize the information contained in the retrieved manual, blocking the use of the model's prior knowledge that is not validated by the vehicle manufacturer; if it utilizes its own knowledge, it could confuse the model or configuration, potentially leading to critical issues. In the event that the exact solution is not found within the document, the system must indicate that it does not possess the information, completely discarding the possibility of generating incorrect or unverified technical data.

NF-03: Security in Credential Management: All access keys to external artificial intelligence services and databases must be securely managed via environment variables on the server. No authentication credentials should be visible within the user interface code. This separation ensures that data traffic is transmitted securely and that access to third-party infrastructures remains fully protected against potential vulnerabilities.

NF-04: Virtualized Deployment and Resource Optimization: The system must be designed to operate stably within the memory and processing constraints of a virtual private server. Given that this infrastructure has limited resources, the router's architecture must be memory-efficient to prevent server saturation during simultaneous requests. Likewise, the execution environment must be packaged using software containerization technologies. This ensures that maintenance, backups, and system updates can be performed rapidly and in a controlled manner, maximizing service uptime.

NF-05: Economic Viability and Cost Control: The system's design must guarantee the project's financial sustainability, adapting its consumption to the execution environment. During the testbench phase, the system will utilize paid tiers for language processing services. This configuration allows the complete batch of sixty questions to be processed without interruptions due to rate limits, assuming a temporary spike in expenditure that is necessary to measure and record the true costs of each technology. Conversely, upon its transition to the production environment, the architecture must rely on the basic, no-cost service tiers offered by providers. By depending on the automatic constraints imposed by these platforms in their free tiers, the system passively ensures that sustained cost overruns are not generated in the face of a potential temporary increase in users. This dual strategy guarantees the acquisition of precise metrics in the research while simultaneously protecting the administrator from any unplanned economic impact in the operational environment.

5 Functional requirements analysis

This chapter serves as a bridge between what the system needs to do and how it will be technically constructed. The objective is to define how data is internally organized and how it flows through the program. To achieve this, the Unified Modeling Language (UML) is utilized. Given that the project entails two separate execution flows, the testbench environment will be analyzed first, followed by the system in production.

5.1 Conceptual data model

The conceptual data model defines the information structure within the project. Although the core configuration of the artificial intelligence technologies is strictly identical in both scenarios, the manner in which data enters and flows through the system differs. In the testbench, the flow bypasses the classification step and employs a repetitive cycle that reads questions from a document and saves the results directly back into the same document. Conversely, in the production environment, the system incorporates the intelligent router to evaluate and direct the request to the corresponding architecture. Due to this divergence in information handling, it becomes necessary to implement two separate schemas. The data implementation for each of the scenarios is detailed below.

5.1.1 Testbench conceptual model

The data model for the testing environment elucidates what information is handled by the program installed on the workstation. This program is responsible for automatically testing the various artificial intelligence systems. In this scenario, there are no individuals asking live questions; rather, there is an automated process dedicated solely to reading lists of queries and writing the obtained results directly into the spreadsheets.

The primary pieces of information and how they interconnect are described below. All these elements and their relationships can be observed in detail in Figure 3.

The control document is the spreadsheet that functions as the main file. In this design, the document privately stores the name of the technology being examined in that specific sheet. In the schema, its relationship with the questions is represented by a solid diamond. This signifies that the document is the absolute owner of the questions; that is, if the file is deleted, all the queries contained within it disappear with it.

For its part, the test question stores the exact text of the query that the program will send, one by one, to the artificial intelligences to observe their responses.

The ground truth is the perfect and correct answer extracted manually from the technical manual. The straight line linking it to the question in the diagram serves to ensure that each query always has its own assigned grading template, which acts as a guide.

The generated response is what the system returns. As seen in the schema in Figure 3, there is a one-to-one relationship between the question and the response. This is because each spreadsheet focuses on a single technology, thus each query corresponds to exactly one generated solution. Here, the response text, the latency taken by the machine to answer, and the text units consumed are stored, which serve to measure speed and economic cost.

Finally, the judge's evaluation is the end result of the grading process. Since the tests are conducted first, the file containing all the obtained responses must be downloaded and saved in the same directory where the evaluating program operates. To assign a grade, the judge utilizes a connection drawn in the schema as hollow diamonds. This indicates that the evaluation functions as a dossier: it gathers the machine's reply alongside the ground truth,

compares them, and, based on that reading, assigns a numerical score and explains the reasoning behind its decision.

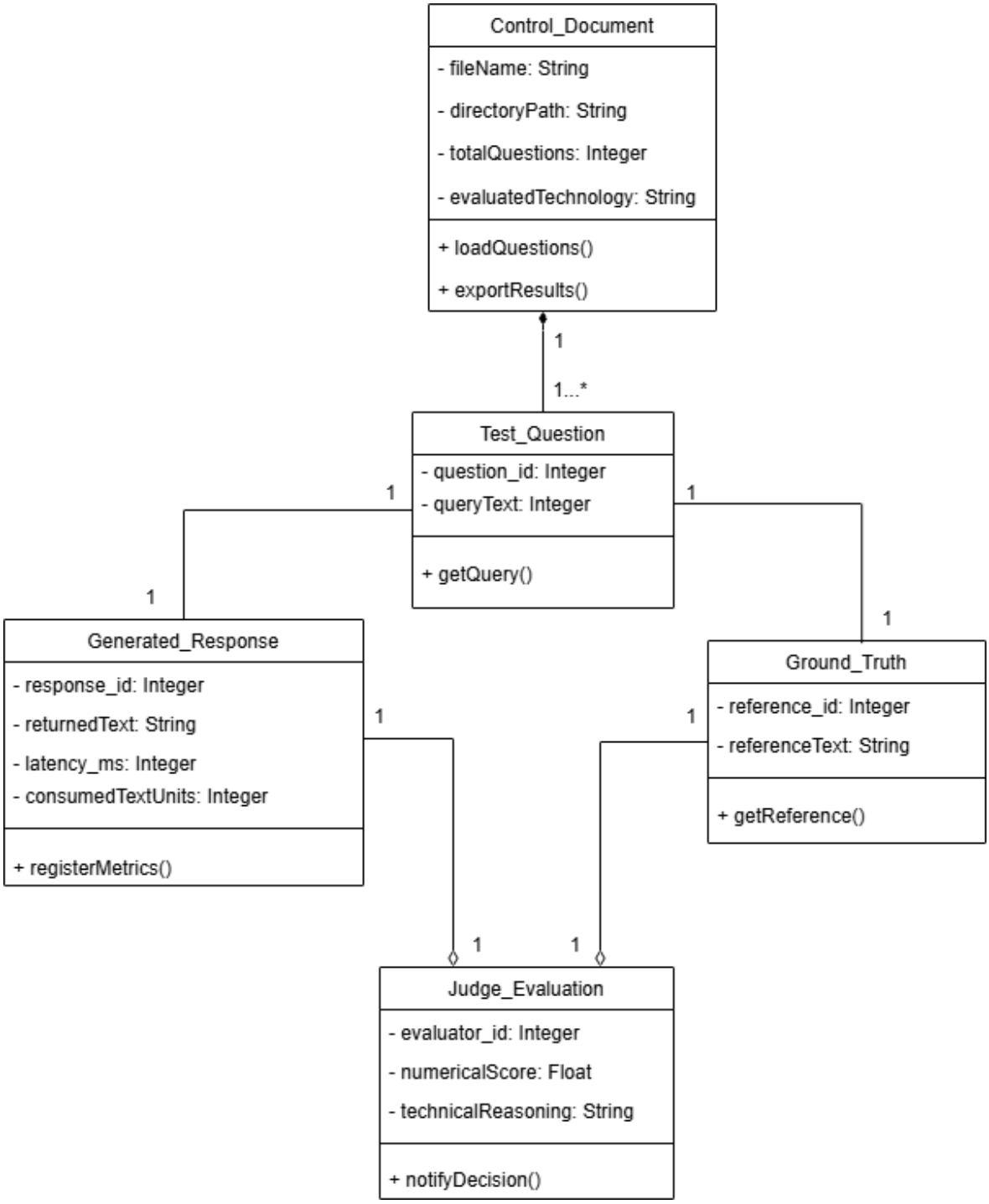

Figure 3. Class diagram of the testbench in the local environment.

5.1.2 Production router conceptual model

The data model for the production environment describes how information is organized when the system is operational for real users. Unlike the testbench, the objective here is to address individual queries in real-time. As we can observe in Figure 4, the structure is more dynamic and features a central component that acts as a brain to decide the optimal path for each query.

The entire process commences with the user query, which is the box located at the top of the diagram. This entity stores the query text, a timestamp to record when it was made, and a boolean data type named pdfAttached. In this instance, it serves to notify the program whether the user has submitted a file alongside their query.

This information travels directly to the classifier router. This component is tasked with analyzing the user's intent using a predefined list of options, technically termed an enumeration (enum). Thanks to this, the system knows whether to provide a simple response, perform a comparison, or act as a complex agent to resolve difficult queries. Once the decision is made, the router communicates with the inference engine. In the diagram in Figure 4, this engine is denoted as an interface. This template ensures that any type of artificial intelligence we utilize always knows how to process the query and return a generated response.

Beneath this general template, we find three specific architectures that connect via arrows with a hollow triangle; these inherit from the primary engine. In modeling language, this represents inheritance, meaning these three boxes are specialized versions of the main engine. The first option is the decoupled Retrieval-Augmented Generation engine, known by the acronym RAG. This name refers to a technique where the machine retrieves external information to enhance its response. To operate, this engine connects with mathematical vectors, which are arrays of numbers representing the semantic meaning of phrases. These vectors are linked to fragments (chunks), which are the text snippets extracted from the original technical manual. In the drawing in Figure 4, we see a solid black diamond at these junctions, indicating that the vector belongs to the fragment, and the fragment belongs to the manual. It is crucial to highlight that, for this system to respond, the manual must have previously undergone a preparation process called data ingestion. In this pre-production phase, the program reads the file, divides it, and stores all the numerical data in a vector database. If this information loading is not performed beforehand, the system will have no stored knowledge, and therefore will not be able to retrieve any text snippets or find mathematical meanings to resolve the operator's queries.

The second option is the managed engine, which delegates all the search and response work to the artificial intelligence provider's own internal infrastructure. In the schema in Figure 4, this architecture connects directly with the document via a one-to-one relationship. This is because, operating as a black box, a concept referring to a system where we observe the input and output, but not the internal workings, the programmer has no control over how the information is divided. The only indispensable prerequisite for this system to function is that the manual's PDF file must have been previously uploaded to the provider's file search tool. In this way, the engine simply identifies the complete document that the external company already has stored to find the necessary solutions within it, without our intervention in the internal search process.

Finally, the third option is the long-context engine. As seen by the line traversing Figure 4, this engine bypasses the snippets and numbers to connect directly with the binary file of the complete document. This allows it to read the entire manual at once to answer questions that require a holistic view of the entire technical book.

This entire structure is designed so that private data, marked with a minus sign, remains protected within each box, while public functions, marked with a plus sign, allow the components to systematically pass information to one another until delivering the final response to the user.

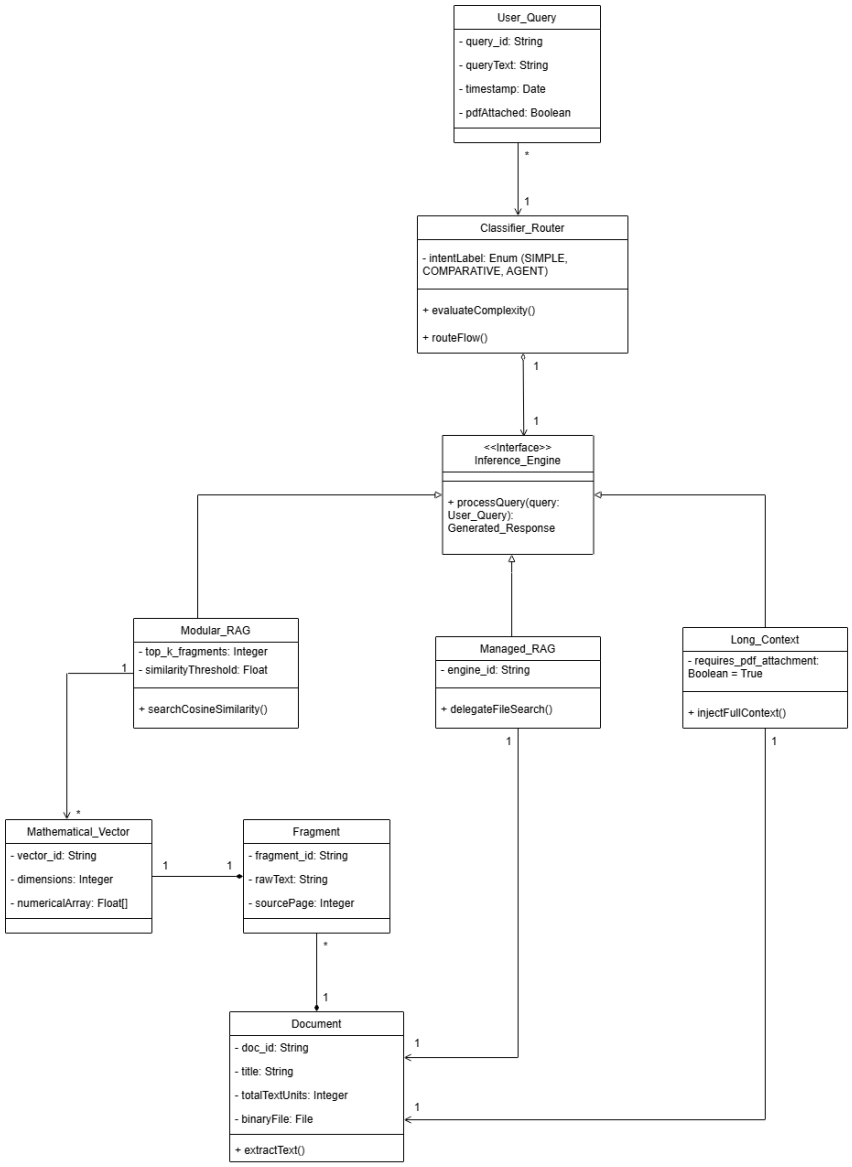

Figure 4. Class diagram of the system structure in production.

5.2 Dynamic model

While the conceptual model defines the fixed organization of information, the dynamic model describes how data moves over time. To explain this behavior, several sequence diagrams have been designed. A sequence diagram is a schema that shows the order of messages and calls made between the different parts of the system, from the moment a question originates until the solution is delivered.

The movement of data is analyzed in two distinct scenarios: the testbench, where everything is automated and controlled by a technician, and the production environment, where the system responds to real operators.

5.2.1 Workflow in the testbench

This section details the behavior of the program during performance testing. In this scenario, an administrator initiates an automated process that reads questions from an organized list. The objective here is to measure how long each architecture takes to answer and what quality of response it offers.

5.2.1.1 Sequence diagram: modular RAG architecture

This section describes the flow of the Retrieval-Augmented Generation architecture—a technique where the system searches for information in the vector database before answering—in its modular version. This latter term indicates that the different phases of the process are separated, and the program must manage them one by one. As shown in Figure 5, execution begins when the administrator gives the start command. From that moment, an automatic repetition component, known as a loop, takes control to continuously evaluate each question from the control file.

To process each question, the testbench first sends the text to a programming interface, which functions as a communication bridge between different programs, with the objective of performing vectorization. This step consists of converting the words of the question into lists of mathematical numbers. Using these numbers, the system queries a specialized database to locate and extract the text units from the technical manual that best fit the problem.

Subsequently, the program delivers the original question along with these information snippets to the artificial intelligence engine so that it can draft the final response. After organizing the received data into a common format and saving the result in the corresponding row and column, the cycle starts over with the next question. This schema allows for a straightforward visualization that, by having the steps separated, the system needs to perform several information exchanges over the internet to resolve a single query.

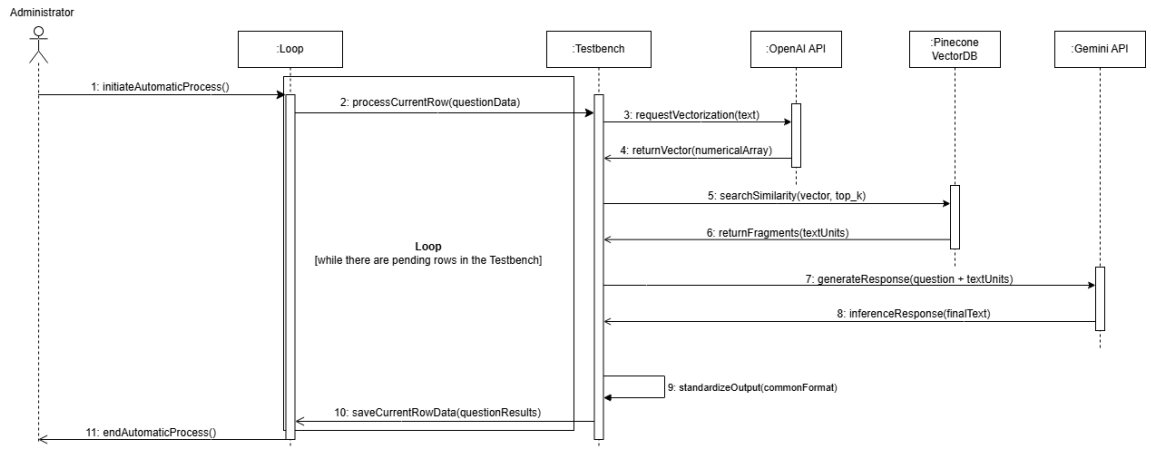

Figure 5. Sequence diagram of the modular RAG architecture.

5.2.1.2 Sequence diagram: managed RAG architecture

This section details the flow of the Retrieval-Augmented Generation architecture in its managed version. Unlike the previous model, this configuration operates as a black box, a concept indicating that the system delivers data and receives a result without knowing the internal steps the machine has taken to achieve it. As shown in Figure 6, following the administrator's start command, the automatic repetition component begins to evaluate each question.

In this case, the testbench takes the original query and automatically attaches an identifier code for the technical manual. It is crucial to note that the person who inputs the original query does not need to know or provide this code, as the internal system itself is responsible for referencing the correct document on its own for each evaluation. Once the question and the reference are combined, the program sends the package to the provider's programming interface.

From there, this external service performs an opaque search on its own servers to find the information, draft the solution, and return the final text without displaying the algorithmic procedure. After saving the results of that row, the cycle advances to the next question. This design demonstrates how internet calls are reduced by delegating all the complexity of the search to the provider company's infrastructure.

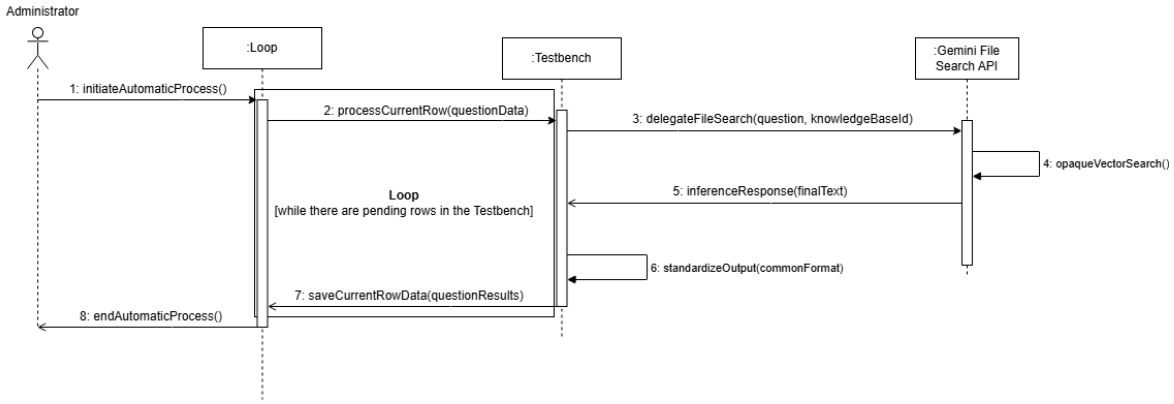

Figure 6. Sequence diagram of the RAG architecture in its managed version.

5.2.1.3 Sequence diagram: long context architecture

This final section describes the flow of the long context architecture. This configuration is characterized by not requiring prior processes of text segmentation or information search in databases. As observed in Figure 7, following the administrator's start command, the automatic repetition component evaluates each question on the list. For each case, the testbench simultaneously sends the user's query and the complete technical manual file to the artificial intelligence engine's programming interface. Upon receiving this package, the external system must process millions of text units at once. These units are the minimal fragments into which the machine divides the document in order to read the entire content simultaneously and comprehend the overall context. After generating the drafted response, the program organizes the data, saves the row's results, and restarts the cycle. This diagram demonstrates the great simplicity of communication over the internet, which is reduced to a single large transmission, in exchange for transferring the entire reading effort to the provider's server.

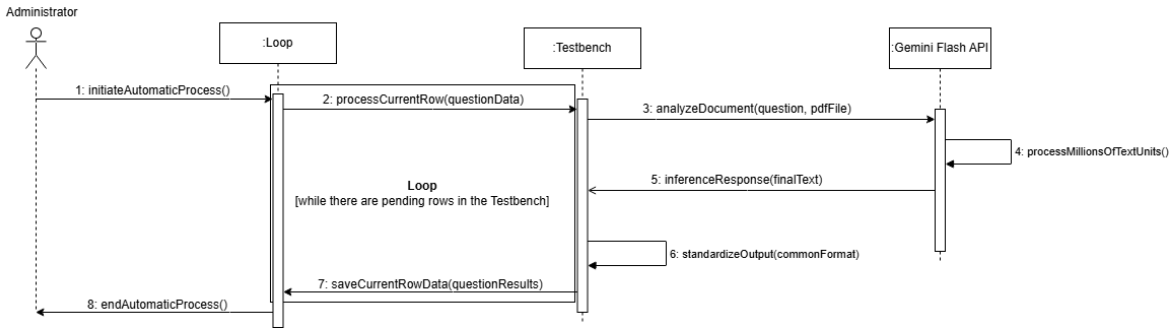

Figure 7. Sequence diagram of the long context architecture.

5.2.2 Workflow in production

This section describes how data moves on the actual server when an operator makes a query from their workstation. Here, the entity in charge of organizing all the traffic is a coordinator.

5.2.2.1 Sequence diagram: modular RAG architecture

This section describes the flow of the RAG architecture in its modular version within the production environment. As observed in Figure 8, the process is linear and is triggered by an end user's request, without requiring automatic repetition cycles. The query reaches the central coordinator, which is responsible for directing all communications.

To resolve the query, this program first sends the text to an external mathematical conversion interface to transform it into a list of numbers.

Using this numerical result, the coordinator queries a specialized data store to retrieve the most relevant text units from the technical manual.

Subsequently, the system sends these supporting units along with the original question to the artificial intelligence engine to draft the solution. After adjusting the format of the obtained information, the coordinator delivers the final response to the user's screen. This design demonstrates how the central system must make multiple external calls, step by step, to address a single real-time request.

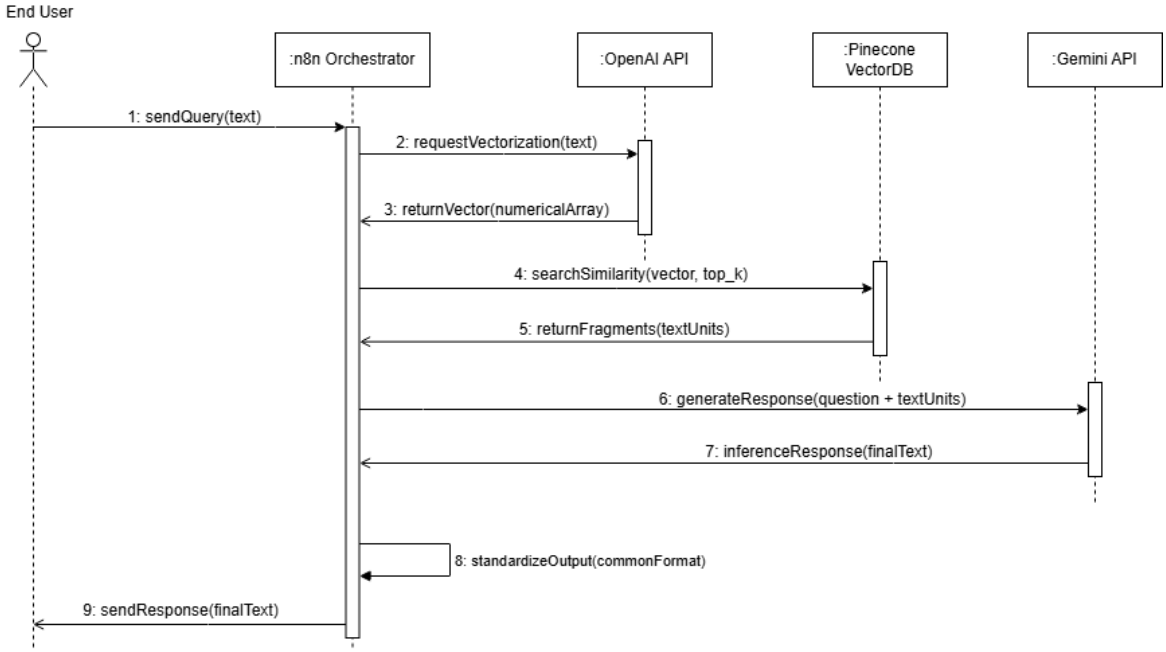

Figure 8. Sequence diagram of the modular RAG architecture in production.

5.2.2.2 Sequence diagram: managed RAG architecture

This section describes the flow of the Retrieval-Augmented Generation architecture in its managed version within the production environment. As shown in Figure 9, this model greatly simplifies data management by operating under the black box concept, a dynamic where the system delivers a request and receives a result without having to program or manage the internal steps. The process begins directly when an end user sends their query to the central coordinator.

Instead of processing the information in parts as in the previous model, when the end user types their question, it is the central coordinator who takes that query and automatically appends an identifier code for the technical manual. It should be noted that the person making the query does not intervene in this step nor do they need to know said identifier, as the internal system is responsible for referencing the correct document on its own. Once the question and the code are combined, the coordinator sends the request to the provider's

programming interface. From that moment, the external service executes an opaque search, internally taking charge of locating the necessary information among its documents and drafting the solution without displaying the algorithmic procedure. After receiving the already generated final text, the coordinator adjusts the data into a common format and returns it directly to the user's screen. This diagram demonstrates how data transfers across the internet are reduced by delegating all the search complexity to the external provider's infrastructure.

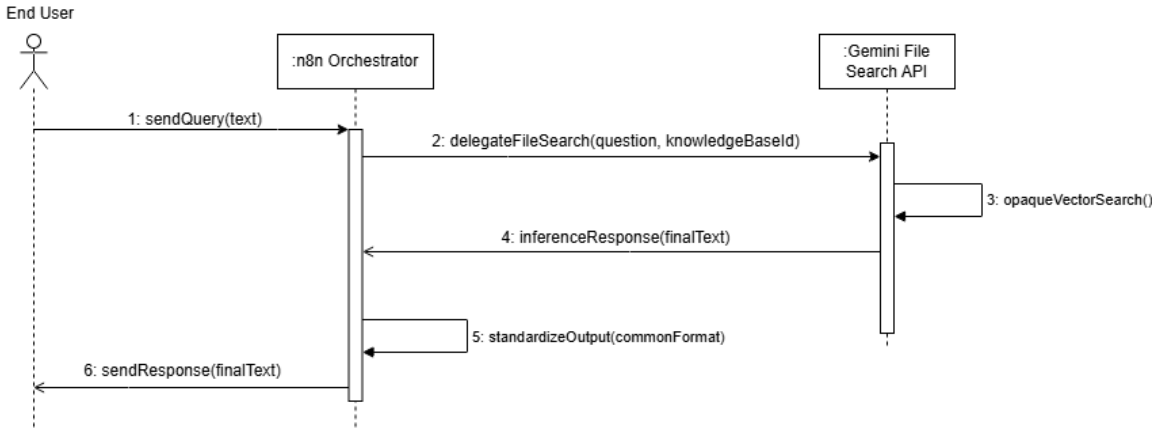

Figure 9. Sequence diagram of the managed RAG architecture in production.

5.2.2.3 Sequence diagram: long context architecture

This final section describes the flow of the long context architecture within the production environment. As detailed in Figure 10, this model stands out for the great simplicity of its communications. The process begins when the end user sends their question along with the complete technical manual file to the central coordinator. Without the need to perform prior searches or information segmentation, the coordinator transfers this complete package to the artificial intelligence engine's programming interface. Upon receiving the complete document, the external service assumes the task of processing millions of text units at once to comprehend the overall content and draft the solution. Once the system generates the response, the coordinator adjusts the format of the data and delivers it directly to the user's screen. This schema reflects how the need to consult additional databases is eliminated, transferring the entire effort to the provider server's extensive reading capacity through a single transmission of information.

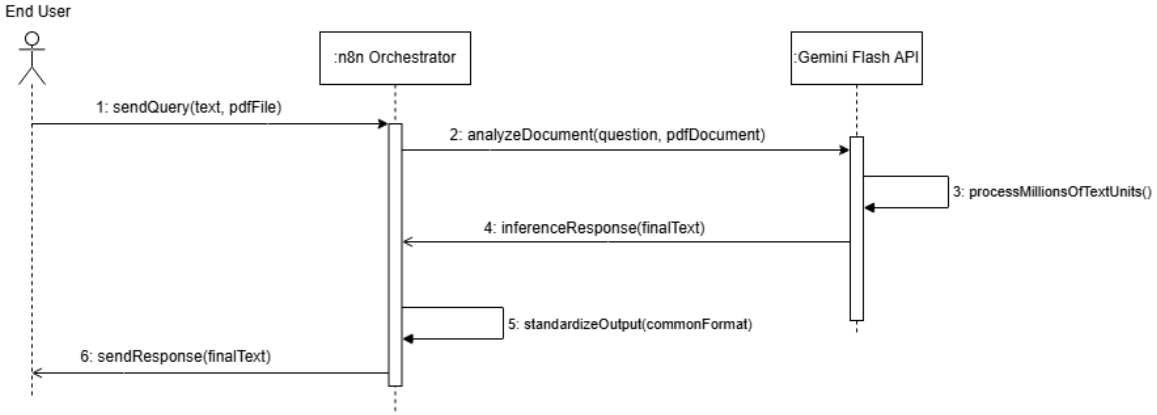

Figure 10. Sequence diagram of the long context architecture in production.

6 Design

This chapter documents the technological decisions adopted to meet the functional and operational requirements analyzed in the previous phase. While the conceptual model explained the solution in a theoretical manner without delving into programming details, the design phase defines the actual structure of the system. The following lines justify the selection of tools, platforms, and working methods. The main objective of this section is to demonstrate how a stable, adaptable, and easily auditable infrastructure has been built, allowing the performance evaluation to be executed under conditions very similar to those of a real corporate environment.

6.1 Testbench design

For the automatic experimentation phase, the system requires an engine capable of processing a large quantity of consecutive questions without requiring human attention. As shown in Figure 11, the design of this testing laboratory has been built using n8n, a visual automation tool. This type of program allows for the creation of processes by connecting graphical blocks on a screen instead of writing traditional programming code, acting as the central engine that directs the entire continuous workflow.

Data storage and reading for these tests have been designed using cloud-hosted spreadsheets. This document acts as the experiment's control center. As the diagram reflects, the process operates through a repetitive cycle: the program checks if there are pending questions, reads a row from the sheet to obtain the question, and sends it to the single architecture that the administrator has activated for that test. After performing the queries, the system reconnects to the document to write the obtained response, the wait time, and the text unit consumption in the corresponding columns, repeating the process until the list is completed.

This spreadsheet-based design was chosen for its high compatibility and simplicity. By saving the information in this common format, it facilitates future direct data reading by other analysis programs. In this way, once the automatic data collection is finished, it is highly straightforward to process the results to calculate quality scores and generate performance graphs without having to design complex storage structures.

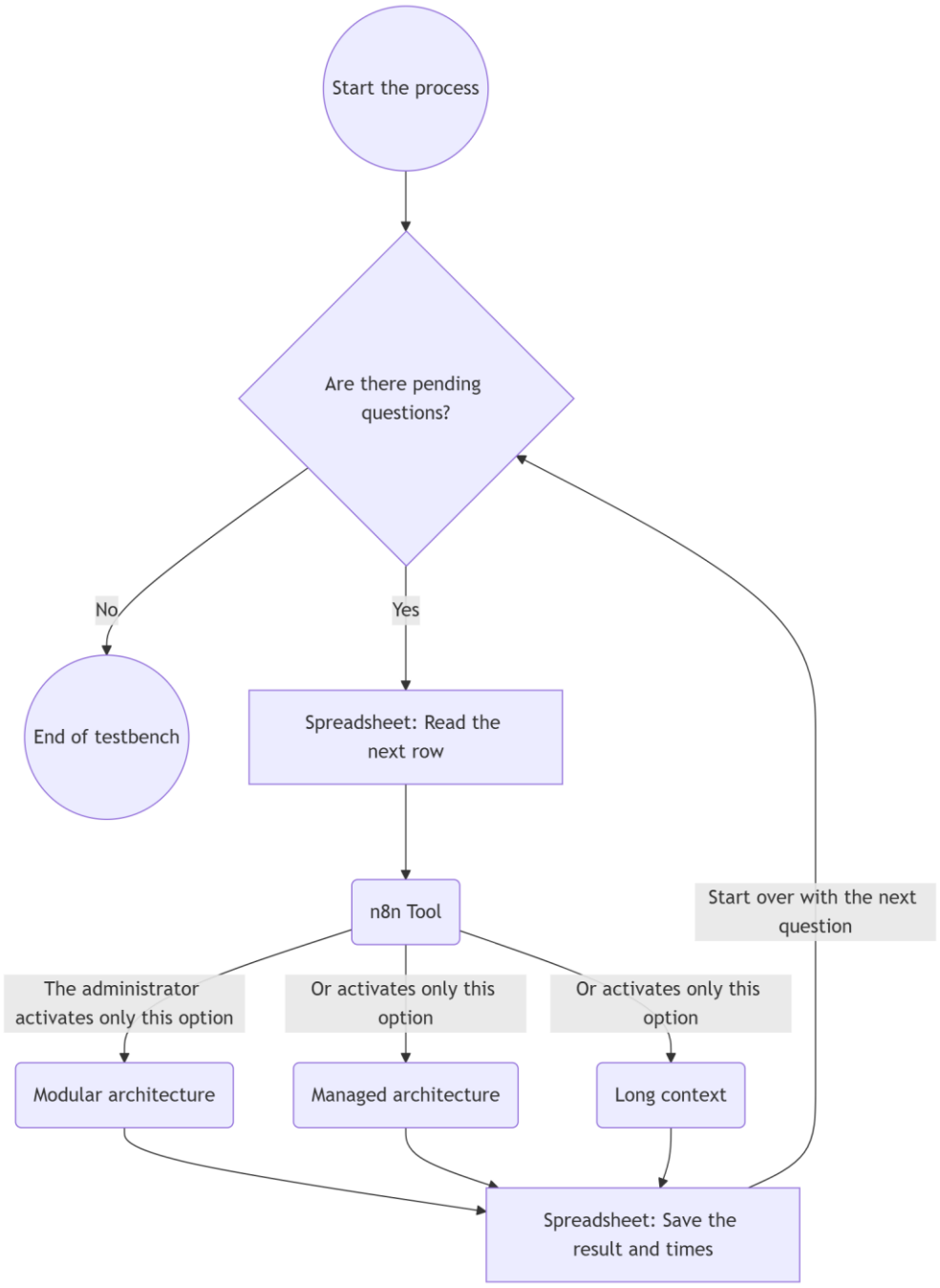

Figure 11. Testbench architecture. The schema details the repetition cycle.

6.2 Design of the production environment

The system has been designed entirely under a cloud services model. Outsourcing the artificial intelligence infrastructure and vector databases eliminates potential local hardware performance issues, allowing this study to focus exclusively on response times and operational costs.

As detailed in Figure 12, the architecture is divided into layers interconnected through secure programming interfaces, known as APIs.

The interactive presentation layer is developed with the Next.js framework and deployed on a virtual private server. A public access approach without registration has been chosen, applying the privacy by design principle to avoid storing personal information.

Conversation continuity is temporarily managed via Firebase. Given that it is volatile storage—as the session is destroyed upon reloading the page—an export module was integrated, allowing the user to download their history as a plain text document, thus delegating total custody of the data to their own device.

The core relies on the coordination layer, built with the n8n automation tool. This intermediary system receives encrypted web requests and executes a conditional router, which is responsible for evaluating and directing the query to the corresponding architecture.

Depending on the route selected by this router, the processing layers come into play. The query goes directly to the modular system with Pinecone, to the provider's integrated solution via Gemini File Search, or to the full reading model, ultimately returning the generated response to the user through the secure channel.

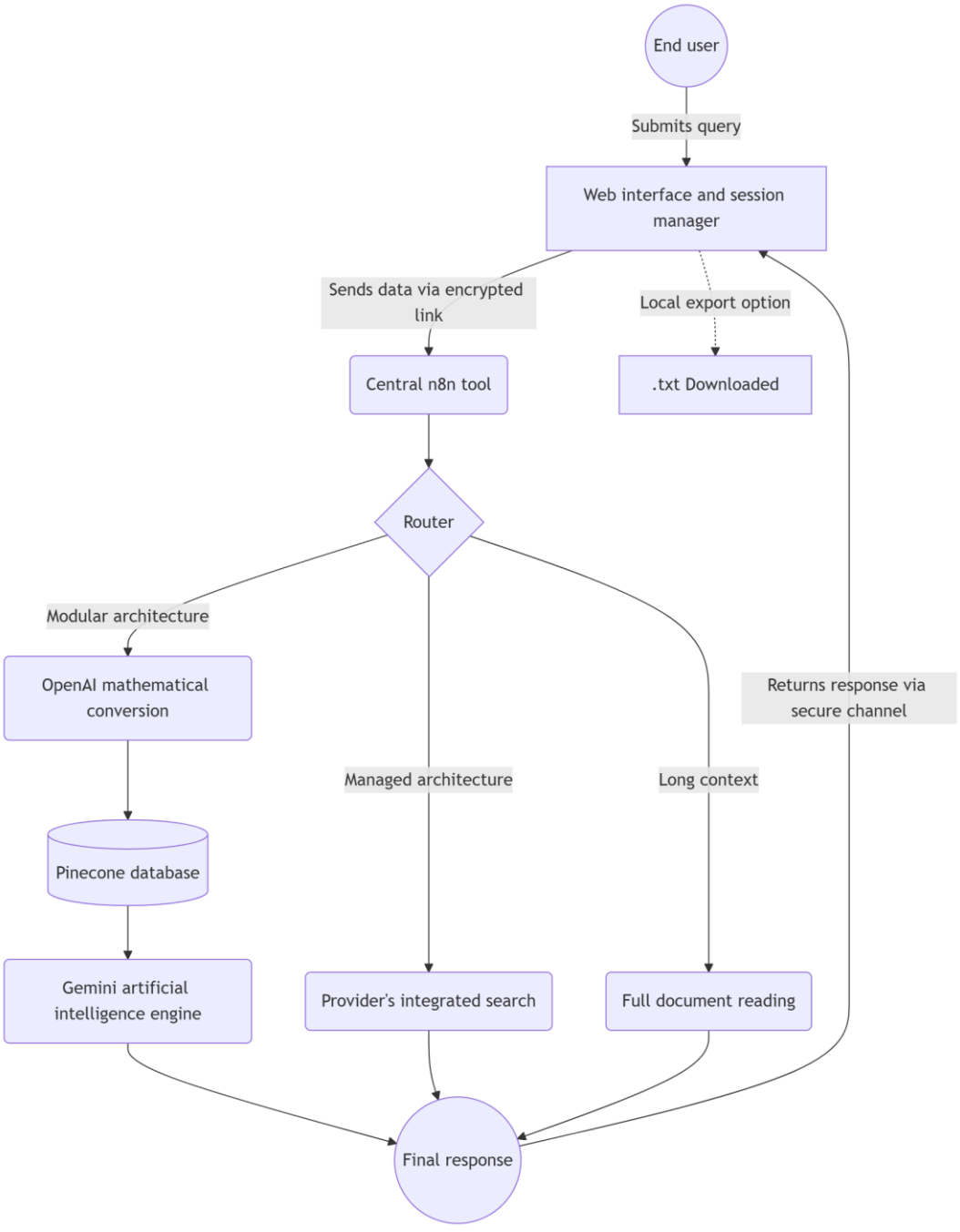

Figure 12. Production environment architecture.

6.3 Design of the external services and the interface

This section describes the configuration of the external tools that support the system's logic. Both operational parts, the human interaction as well as the testbench, communicate via data packets in JSON format, a lightweight information exchange standard, which travel across the internet using encrypted links to ensure confidentiality at all times.

In the real-world usage environment, the web-exposed application includes an active security layer in the browser based on three pillars to protect the central coordinator.

First, it applies text sanitization filters to prevent the entry of malicious code.

Second, it inspects the integrity of attached files by analyzing their internal binary codes to ensure they are authentic PDF documents, applying a 65-megabyte limit to prevent server saturation.

Third, it relies on the volatile session management and local export detailed in the general architecture, guaranteeing strict compliance with user privacy.

Regarding the internal workings of the queries for the modular architecture, persistence is delegated to Pinecone. Unlike traditional databases known as SQL, this tool locates fragments of the technical manual by comparing their mathematical proximity within a high-dimensional space. The index stores the numerical representations of the text, called embeddings, created by the OpenAI programming interface. To find the most relevant information, the system utilizes cosine similarity, a metric that calculates the mathematical distance between the user's query vector and the indexed fragments, retrieving the original text so that the artificial intelligence system can process it.

Finally, regarding the experimentation phase, the cyclical data flow managed by n8n and the spreadsheets functions structurally as an extract, transform, and load model. This modular design allows the collection of metrics to be separated from their subsequent analysis. The recorded information will be consumed by external evaluation programs based on the Python programming language. This component will act as a logical bridge between the data and an artificial intelligence model that will serve as an evaluating judge to score the fidelity of the results and provide technical justifications. The implementation of these logics and the code snippets will be detailed in chapter 7, while the visual representation and discussion of the results will form the core of chapter 8.

7 Implementation

The implementation is detailed below, following the guidelines of the previously proposed design. This section documents the detailed configuration of each workflow, each node that constitutes the workflow, and the rationale behind choosing this strategy and structure, both in the testbenches and in production. Finally, this section concludes by discussing the most prominent problems encountered and the design decisions made to resolve them.

7.1 Technologies used

For the construction of this system, a set of tools has been selected, prioritizing compatibility, serverless scalability, and data processing efficiency. They are briefly detailed below.

Next.js. Framework used to develop the website. It facilitates the creation of a secure and agile interface, integrating automatic protections against malicious code injection in the browser.

Firebase. Cloud storage service implemented to maintain the temporary memory of the conversation. It guarantees that the data disappears upon reloading or closing the page.

Nginx and Linux. Web server and operating system hosting the user interface and workflows. Their configuration allowed for expanding the attachment file size limit and avoiding connection timeouts during prolonged wait times.

n8n. Visual automation tool acting as the system's central engine. It allows structuring workflows and connecting programming interfaces without the need to write traditional code.

JavaScript. Programming language used to resolve data loss issues in some of the workflows. Its use allowed for the recovery of the original documents in binary format before sending them to the artificial intelligence.

Google Gemini 3.0 Flash. Main artificial intelligence engine utilized for its high speed and large reading capacity. It is responsible for processing complete documents and drafting the project's final responses.

OpenAI API (text-embedding-3-large). Tool used to convert text into mathematical representations. It provides a high-precision reference point for preparing the text before saving it in the database.

Pinecone. Cloud vector database that stores the mathematical coordinates of the text. It allows information to be located by semantic proximity in fractions of a second without requiring physical server maintenance.

Google Workspace. Suite of cloud tools, highlighting spreadsheets, used as the main registry. It allows real-time saving of performance and quality metrics for their subsequent analysis in section 8.

Python. Programming language employed to create external statistical analysis programs. It reads data from the spreadsheets, executes calls to the automatic evaluator, and generates the results graphs.

OpenAI GPT-4o. Artificial intelligence model used as an automatic judge in the auditing phase. It is responsible for comparing the generated responses with the correct data to issue objective fidelity scores.

Git and GitHub. Version control systems used to save the program's change history. They have allowed for the secure separation of the analysis environments, the coordination, and the user interface.

7.2 Implementation of document preparation and loading

This section details the necessary process for the artificial intelligence system to access the content of the vehicle manual. Given that each of the three evaluated options processes information differently, the preparation of the files requires specific programming approaches. The three methods developed for the project are explained below: the transformation of text into mathematical coordinates for the vector database, the uploading of the document to the provider's servers for the managed solution, and the direct reading of the file for the long context architecture.

7.2.1 Ingestion for the modular architecture

For the modular RAG architecture to function, a preliminary phase of extraction, transformation, and loading of information regarding the vehicle manual is necessary. This process, commonly known as data ingestion, converts the original static document into a mathematical knowledge base that can be queried in real time. To implement this process without relying on small, isolated local programs, an automated workflow was designed using n8n.

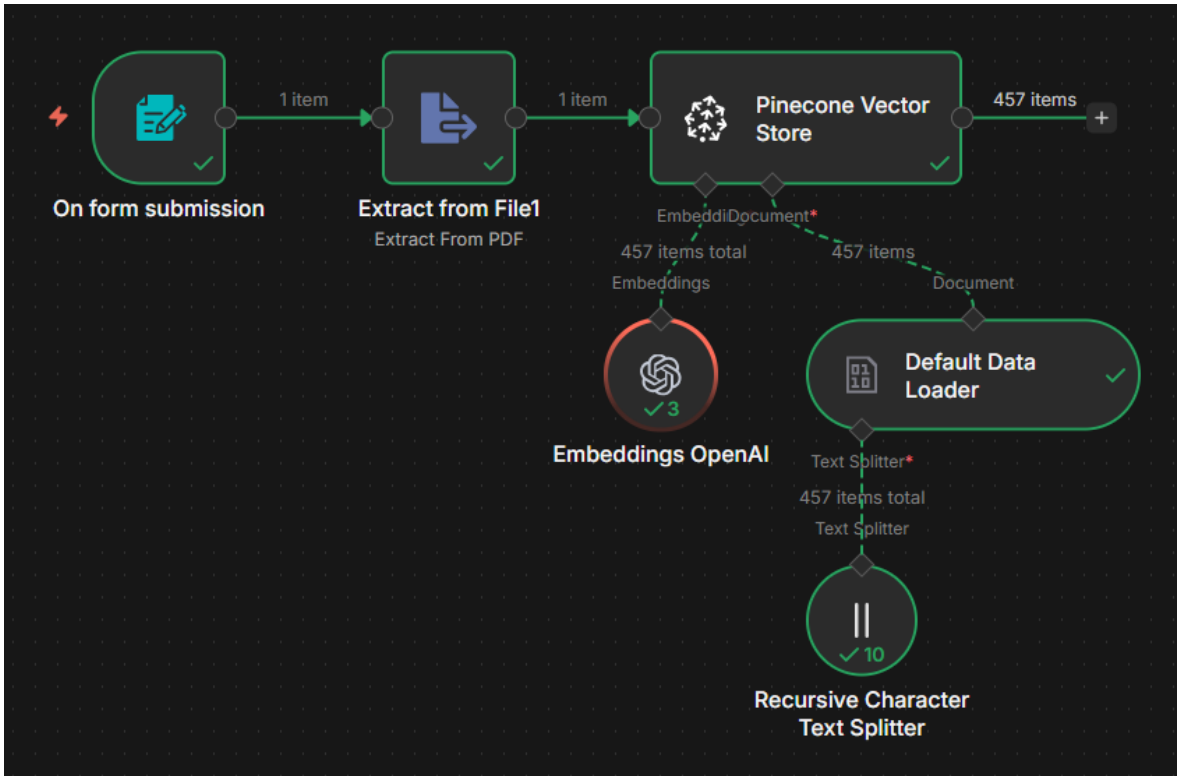

Figure 13. Workflow implemented in n8n for the data preparation phase of the modular architecture.

As observed in Figure 13, the preparation flow is divided into three phases. The process is initiated by an input event that receives the vehicle's workshop manual in binary format. Subsequently, an extraction node processes the data stream and retrieves the entirety of the content in plain text format. In this phase, unreadable elements and visual formatting details are eliminated, generating a large text string containing all the technical knowledge of the manual.

Given that the extracted text exceeds standard reading capacity and the limits of the mathematical conversion interface, the document is then fragmented. To this end, a data loading tool coupled with a recursive character text splitting algorithm is integrated. The selection of this splitter over those that cut by a fixed number of letters is highly important. The recursive algorithm attempts to keep paragraphs and sentences intact, dividing the information by logical line breaks before forcing a definitive cut by length limit. This prevents technical values, such as "the tightening torque of a part", from being severed in half. The exact configuration of the fragmentation parameters for this project was established at a chunk size of 1000 characters, with an overlap of 200 units. This overlap ensures that the end of one chunk contains the beginning of the next, maintaining context between the different sections of the manual.

Once the chunks are generated, the central node connected to the Pinecone database, configured in document insertion mode, is responsible for directing the call to the OpenAI interface. Through this link, each text chunk is sent to the high-capacity mathematical model. This system returns a numerical representation of the text in the form of a vector, that is, a list of floating-point numbers in a high-dimensional space. Once this value is calculated, the program transmits it to Pinecone, where it is permanently stored.

To understand the system's behavior and the processing cost, it is necessary to analyze the mathematics of the splitting algorithm. When the workflow processes an excerpt of the manual of, for instance, 103 pages, the tool does not return 103 results; instead, it generates a volume of 457 independent vectors. This data expansion responds to the text density and the configured parameters. By setting the size at 1000 characters and the overlap at 200, which represents a twenty percent margin, the algorithm advances in blocks of 800 new characters each time. Added to the flexibility margin of the recursive splitter that seeks logical line breaks, the system yields an average of slightly more than four chunks for each physical page processed. This fine-grained division is what guarantees high search precision, as it isolates each technical procedure into its own independent mathematical coordinate.

To verify the correct execution of the workflow, the actual data stream was reviewed. After processing a document, the system generates a structured output containing a unique identifier, the numerical array, and the original text content saved as additional metadata. The final confirmation of this operation was performed through a direct review on the Pinecone control panel. As shown in Figure 14, the administration console confirms the creation of the index and the successful storage of all vectors, ensuring that the database is prepared and ready to receive queries in real time.

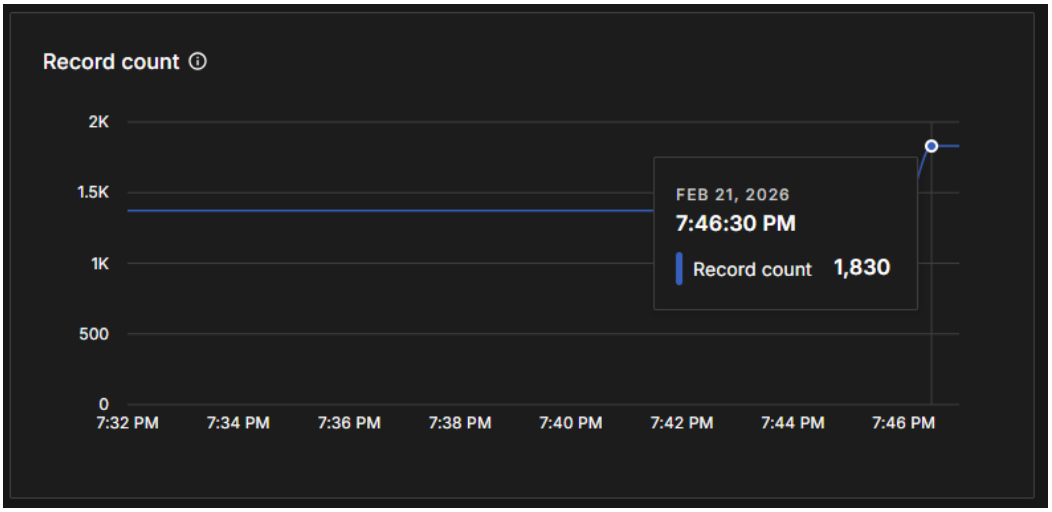

Figure 14. Pinecone administration panel confirming the successful saving of the technical manual vectors.

7.2.2 Ingestion for the managed architecture

Unlike the rigorous extraction, transformation, and loading phase required by the modular architecture, the implementation based on Google's integrated ecosystem relies on an infrastructure abstraction model. The objective of this approach is to evaluate the performance of a solution entirely managed by the provider, a computing concept commonly known as a black box system. In this model, the cloud provider company assumes and completely conceals the complexity of the data flow, taking charge of fragmentation, mathematical conversion, and indexing without the programmer's intervention.

For its implementation, the preparation process requires a fundamental preliminary step. Before being able to send queries, it is necessary to establish the database on Google's servers. First, a specific file search space must be created within their environment, which functions as the container where the ingestion and storage of all information will take place. Upon creating this store, the system provides a unique identifier code (e.g., fileSearchStores/chatmotor-12a2raoth0lx). Figure 15.

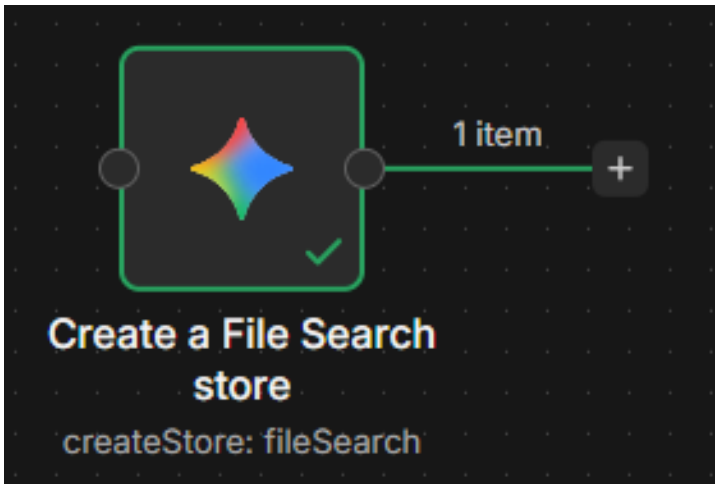

Figure 15. Pre-setup of the managed file interface.

Subsequently, once the container is established, the technical manual is uploaded to this newly created space. To automate this step, as presented in Figure 16, a workflow has been designed in the n8n tool that communicates with the provider's programming interface. Before sending the document to Google's servers, the system performs a fundamental action: it links the upload request with the identifier code obtained in the previous step. This initial configuration is mandatory to indicate to the platform exactly in which data store it must save the information.

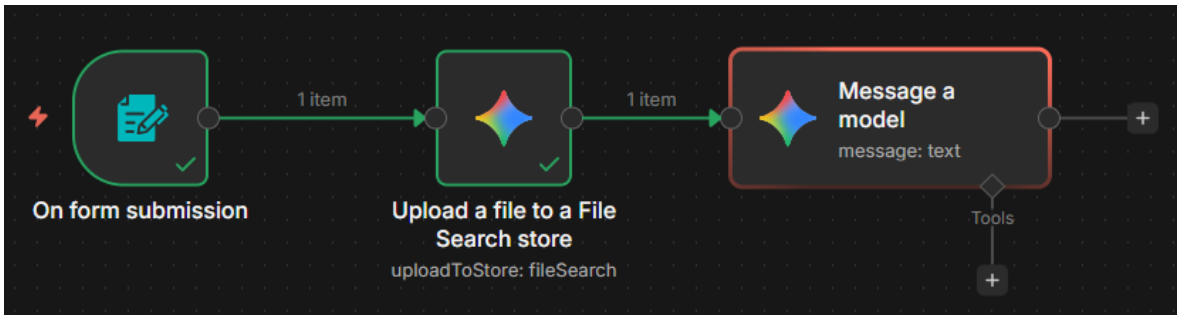

Figure 16. Preparation flow for the managed architecture. The schema details the document upload.

After establishing this destination route and linking the document with its container, the file upload is executed. Given that the platform requires time to automatically read and

process the information on its own equipment, the system waits. Finally, it is confirmed that the document has been uploaded and is now ready to be used.

This structural simplicity considerably accelerates the project's setup, as it entirely dispenses with the creation and maintenance of external databases. However, it imposes significant technical and operational limitations right from the preparation phase itself.

First, by delegating all the loading effort to the provider, the developer loses control over how the information is stored internally, remaining unaware of the exact size of the generated chunks or the mathematical metric employed to organize them. This loss of traceability from the moment of upload confirms the opacity of this black box model and justifies the need to conduct this comparative study.

Next, during development, a significant disadvantage related to processing capacity was identified. Unlike the modular architecture, which allowed for the continuous assimilation of extensive hundred-page manuals, the integrated system presented restrictions during loading. To ensure the platform accepted the information without displaying errors, it was necessary to shorten the technical documents and perform the uploads by dividing the files into smaller, fifty-page blocks. This restriction forces a preliminary partitioning effort, demonstrating that the convenience of delegating the infrastructure entails tangible barriers when working with real documentation.

7.2.3 Direct loading for the long context

The third evaluated architecture represents the processing of complete documents. Unlike the modular and managed retrieval systems seen in the previous sections, this option completely lacks a prior preparation, extraction, or indexing phase. Its operation does not require creating databases or saving text chunks in advance. Instead, it relies on sending the entire manual and the user's question simultaneously during each individual query.

To materialize this flow in the n8n tool, a workflow designed to dynamically handle large files was structured. As observed in Figure 17, the process begins with reading the original raw file. In the test environment, this is done through an input event that retrieves the document.

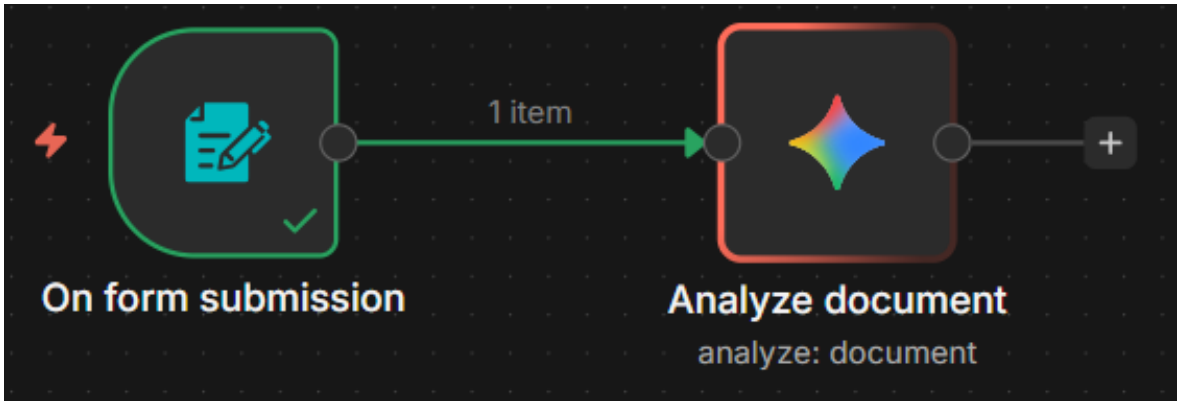

Figure 17. Coordinated workflow in n8n for the extensive context architecture.

The core of this architecture lies in the document analysis node. In its internal configuration, the incoming file is linked, and the main artificial intelligence model is selected. To ensure precision, the system is configured to receive the user's question accompanied by a strict anchoring directive: to analyze the attached document in its entirety and answer the query based exclusively on its content. This approach forces the foundational

model to process, calculate, and store hundreds of thousands of text units in its temporary memory for each independent interaction. This enables a global understanding of the manual, but entails a natural penalty in both the response wait time and the increase in cost.

Given that this approach does not require a traditional ingestion phase into a database, the document preparation configuration concludes at this point. The detailed analysis of how this system actually works in practice, as well as the evaluation of its enormous capacity to simultaneously correlate technical information to resolve breakdowns, will be explained later in the document, during the results analysis phase.

7.3 Implementation of the automated testbench

The testbenches implemented for each architecture are detailed below. The validation and extraction of metrics for the three technological options required an automatic processing approach. For the design of this testing laboratory, the use of small, isolated local programs was discarded, opting to integrate the coordination of the experiments directly within the n8n tool.

Three parallel workflows were designed, one for each evaluated architecture, acting as a continuous process of data extraction, transformation, and loading. It is important to highlight a fundamental decision in the design of this validation phase: the three architectures are not executed simultaneously. Given that the three workflows consume the same question bank and write the results to different sheets within the same spreadsheet document, a joint execution would cause saturation in data input and output, write locks on Google's servers, and potential errors due to cross-information. Consequently, the design establishes a single manual start button, forcing the researcher to completely execute and finish the test of one architecture before initiating the next, thereby guaranteeing data integrity.

7.3.1 Execution of the tests for the modular architecture

This workflow aims to evaluate the performance of the artificial intelligence agent when interacting with the external vector database. Being the option that generates the most additional data, due to having to retrieve and analyze the stored text chunks, its information processing procedure is the most comprehensive from a tracking perspective. As presented in Figure 18, the repetitive execution processing queries against the database and saving the obtained metrics in the spreadsheets can be observed.

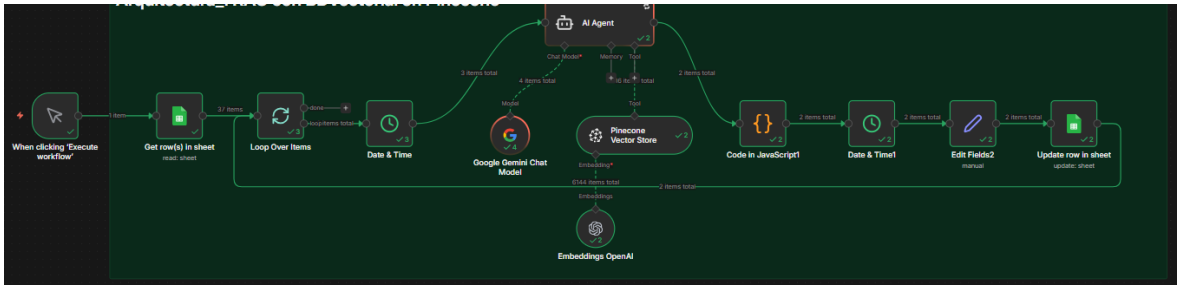

Figure 18. Coordination workflow for the modular architecture testbench.

The lifecycle of this automated process is structured into a series of nodes that operate in a strictly sequential manner. The first step consists of extracting the dataset. The flow initiates by connecting to the spreadsheet interface located in the cloud. The tool is configured to precisely read the source document and extract all available rows. In this

project, the platform retrieves a total of 37 elements, composed of 20 evaluation questions and 17 technical notes.

The second step applies a repetition control. To avoid saturating the network connections and to process each metric individually, a node is utilized that isolates each row of the document. Its primary function is to force the system to execute the remaining operations one by one, sending a single question per execution cycle, which guarantees that the responses do not get mixed.

The third step establishes the initial time tracking. Just before calling the artificial intelligence engine, a clock tool is used to capture the exact system time measured in milliseconds. This value acts as the starting point for calculating the response time. Subsequently, during the results analysis phase, these values will be transformed into seconds to facilitate their visual comprehension.

The fourth step executes the information query and represents the core of the test. The question isolated in the previous step is dynamically introduced into the primary instruction received by the generative model. To limit the operational cost and mitigate the risk of the model hallucinating data and entering an infinite loop attempting to reason about non-existent information, safety restrictions were applied. The agent's maximum repetitions parameter was limited to ten attempts. Similarly, the database was limited to retrieving a maximum of eight text chunks per query.

The fifth step proceeds to extract metrics via code. Given that this architecture conceals its reasoning within a complex output format, it was necessary to program a small script in JavaScript to extract the estimation of consumed text units, a key factor in calculating the system's cost. As detailed in Code 1, the program performs a mathematical approximation of the unit volume based on sentence length. The standard metric formula has been established, where one text unit is approximately equivalent to four characters.

```
// 1. Obtaining the information generated by the system
const agentData = $('AI Agent').last().json;
let question = $('Loop Over Items').item.json.Question || "";
let answer = agentData.output || "";
// 2. Estimation of text unit consumption (1 unit equals 4 characters)
// Input = System instruction (approx. 500 characters) + Question
const estimatedInputTokens = Math.ceil((500 + question.length + contextText.length) / 4);
// Output = Final generated response
const estimatedOutputTokens = Math.ceil(answer.length / 4);
// 3. Return of the calculated data to continue the flow
return {
  json: {
    ...agentData,
    Calculated_Input_Tokens: estimatedInputTokens,
    Calculated_Output_Tokens: estimatedOutputTokens,
    Chunks_Retrieved: chunksUsed,
    Arch1_Response: answer
  }
```

```
};
```

Code 1. Tracking program for the modular architecture.

The sixth step performs the final time calculation. Following the execution of this code, a second temporal tool captures the completion timestamp. Immediately after, a field editing step assumes the role of organizing the final structure. It retrieves all the processed information and performs a mathematical subtraction operation between the end time and the start time to obtain the exact wait time for the current question.

Finally, the seventh step performs the automated saving. As the cycle's closure, the flow reconnects to the spreadsheets to record the results. The program is configured to update the data ensuring exact correspondence, utilizing the question's identifier code as the primary key. This prevents overwriting data in incorrect rows and ensures that the wait time, the consumed text units, and the final response are stored exactly in the columns of the evaluated question. Once this saving transaction is completed, the flow automatically returns to the repetition node to process the next question on the list.

7.3.2 Execution of the tests for the managed architecture

This second automated workflow aims to evaluate the performance of the integrated solution provided by the Google environment. Given that this model completely delegates information retrieval to the provider, the main challenge of this workflow was not the coordination of the different programs, but rather the extraction of metrics from a system characterized by its total opacity. As presented in Figure 19, the flow of requests towards the provider's closed ecosystem and the use of custom code to successfully capture the necessary data are evident.

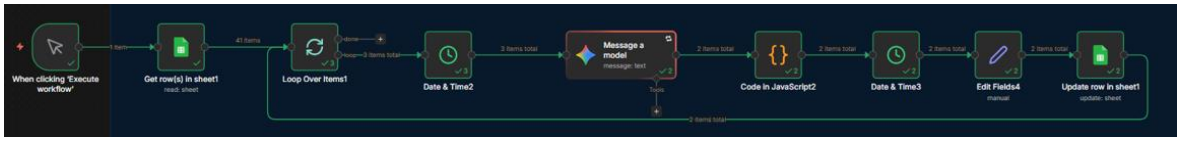

Figure 19. Coordination workflow for the managed architecture testbench.

The lifecycle of this process maintains the same sequential philosophy as the previous architecture, adapting its internal nodes to the particularities of this integrated engine. The process is divided into the following steps:

The first step consists of extracting the dataset. Identically to the first architecture, the flow initiates by connecting to the spreadsheets located in the cloud to download the original question bank.

The second step applies a repetition control. This node isolates each row of the data list, forcing strictly individual processing to prevent blockages due to overloading the server with simultaneous requests.

The third step establishes the initial time tracking. The clock tool is used to capture the exact system time measured in milliseconds prior to sending the information, establishing the point of origin for calculating the response wait time.

The fourth step executes the query to the model. In this phase, the isolated question is introduced into the request. The artificial intelligence model relies exclusively on the file space generated and linked during the prior preparation phase (Section 7.2.2). Likewise, strict behavioral instructions are maintained to guarantee that the response is based solely on the attached manual and is not contaminated with external information previously learned by the model.

The fifth step proceeds to extract metrics via code. Unlike the first architecture, the Google interface dynamically alters the structure of the response data depending on how its internal algorithms process the document. Being an opaque model, the system often does not return the amount of original text it has read to generate the response. To guarantee metric capture and prevent fatal errors in the program, a heuristic search and estimation calculation code was designed in JavaScript. As Code 2 shows, the program searches for the final response by navigating through the different layers of the returned data. If the opaque model conceals the amount of text it has consulted, the program activates an alternative safety plan, performing a mathematical estimation based on the length of the question and the response multiplied by a standard margin to deduce the cost in text units.

```
    // 1. Deep tracking of the final response within the Google data
layers
    let answerText = "";
    if (modelData.text) {
        answerText = modelData.text;
    } else if (modelData.output) {
        answerText = modelData.output;
    }         else         if         (modelData.predictions         &&
modelData.predictions[0]?.content) {
        answerText = modelData.predictions[0].content;
    }
    // 2. Extraction of the context read by the opaque model
    let contextTextLength = 0;
    if                    (modelData.groundingChunks                    &&
Array.isArray(modelData.groundingChunks)) {
        modelData.groundingChunks.forEach(chunk => {
            if (chunk.retrievedContext?.text) {
                contextTextLength += chunk.retrievedContext.text.length;
            }
        });
    }
    // 3. Mathematical calculations (Standardized estimation: 4 characters
= 1 text unit)
    let calculatedOutput = Math.max(Math.ceil(answerText.length / 4), 5);
    let calculatedInput = 0;


    if (contextTextLength > 0) {
        // If the provider returns the amount of read text, the calculation
is exact
        calculatedInput      =      Math.ceil((questionText.length      +
contextTextLength) / 4);
    } else {
        // Alternative scenario: If the provider hides the data, a
baseline estimation is applied
```

```
        calculatedInput = Math.ceil(questionText.length / 4) +
(calculatedOutput * 10);
    }
    // 4. Return of the calculated data to continue the flow
    return {
      json: {
        ...modelData,
        Arch2_Input_Tokens: calculatedInput,
        Arch2_Output_Tokens: calculatedOutput,
        Arch2_Response: answerText
      }
    };
```

Code 2. Metric extraction program for the managed architecture.

The sixth step performs the final time calculation. Once the data extraction and cost calculation are overcome, a second temporal tool records the completion timestamp. Immediately after, the field editing node assumes the organization of the variables and calculates the mathematical difference to obtain the total wait time value in milliseconds.

Finally, the seventh step performs the automated saving. The standardized information is transmitted to the spreadsheets. Unlike the first architecture, this tool is configured to exclusively update the columns belonging to the second test sheet, utilizing the row's original identifier to prevent mismatches in the data table. After successful writing, the cycle returns to the beginning to process the next question.

7.3.3 Execution of the tests for the long context architecture

This final workflow corresponds to the evaluation of the system that processes complete documents. In this model, the artificial intelligence engine does not retrieve small text snippets, but rather reads the entirety of the manual for each request.

During initial testing, a significant design flaw was detected. Originally, the tool for downloading the file from cloud storage was placed within the repetition loop. However, the restrictions of the Google Drive interface's free tier caused blockages due to request saturation, causing the system to fail after a few attempts to continuously download the file. The solution consisted of moving the download command outside the loop and saving the file temporarily in the system's memory. As presented in Figure 20, the final design resolves this bottleneck and ensures smooth execution.

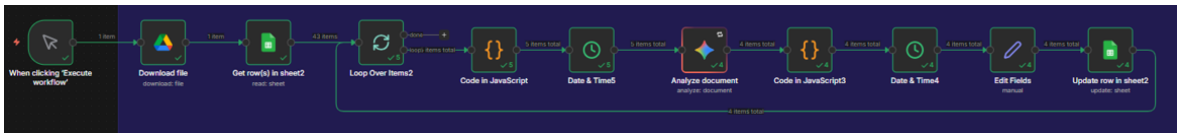

Figure 20. Coordination workflow for the testbench of the extensive context architecture.

The execution flow of this process is structured into the following consecutive steps:

The first step consists of downloading to memory and extracting the data. The flow starts by downloading the manual from the cloud provider a single time. Subsequently, it connects to the spreadsheets to retrieve the list of questions pertaining to the third test.

The second step applies the repetition control. As in the previous approaches, a node isolates the requests to process each question in a strictly individual manner.

The third step executes the injection of the temporary memory via code. To ensure that each question is accompanied by the entire document without having to download it again, a small JavaScript program was implemented. As Code 3 shows, this program retrieves the file saved in the first step and attaches it directly to the current question, saving resources and time.

```
// 1. Access to the previously downloaded file outside the loop
const downloadedFile = $('Download file').first().binary;
// 2. Identification of the data container's name
const dataKey = Object.keys(downloadedFile)[0];
// 3. Injection of the file directly into the current question
$input.item.binary = {
  data: downloadedFile[dataKey]
};
return $input.item;
```

Code 3. Memory injection program. It allows attaching the entire manual to each iteration.

The fourth step establishes the initial time tracking. The exact system time is captured in milliseconds prior to the artificial intelligence call.

The fifth step performs the complete document analysis. The node receives the injected file and the user's question. This step operates under the strict instructions defined to prevent the model from hallucinating responses, forcing it to read the entire attached document.

The sixth step proceeds with the metric extraction. Just as in the previous architectures, the output from the Google interface is processed to calculate the cost in text units. In this case, the algorithm prioritizes reading the official usage data returned by the provider. A design decision was implemented in the mathematical safety formula of this step. Since the cost of sending the full manual represents a very high constant value, close to one hundred forty thousand text units (detailed in Section 8 evaluation on how these 150,000 text units or tokens are obtained), it was decided that this tool should only capture dynamic data. Therefore, if the interface fails and omits the official consumption data, the fallback code exclusively calculates the length of the user's question. The addition of the document's fixed cost is delegated to the subsequent external analytical processing phase in the Python programming language, thus keeping the data clean within the coordinator. Code 4 details this simplification.

```
// 1. Obtaining the official usage data from the provider
let inputTokens = modelData.usageMetadata?.promptTokenCount || 0;
let outputTokens = modelData.usageMetadata?.candidatesTokenCount ||
0;
let answerText = modelData.text || "";
// 2. Alternative safety calculation in case of provider failure
if (outputTokens === 0) {
    outputTokens = Math.ceil(answerText.length / 4);
}
```

```
    if (inputTokens === 0) {
        // Only the question is calculated. The large cost of the document
is added in the analysis phase.
        inputTokens = Math.ceil(questionText.length / 4);
    }
    // 3. Return of the calculated metrics to the main flow
    return {
      json: {
        Arch3_Input_Tokens: inputTokens,
        Arch3_Output_Tokens: outputTokens,
        Arch3_Response: answerText
      }
    };
```

Code 4. Data extraction and protection program.

The seventh step performs the final time calculation. The completion timestamp is recorded, and the mathematical difference is calculated to obtain the wait time value in milliseconds.

Finally, the eighth step performs the automated saving. The cycle finishes in a standardized manner by sending the calculated information to the spreadsheets. The tool saves the data exclusively in the designated columns of the third sheet, completing the execution and returning to the beginning for the next question.

7.4 Implementation of the automatic evaluator

Objectively evaluating the quality of the responses generated by the three architectures posed a significant challenge. The manual review of sixty technical responses proves to be a slow process prone to human cognitive bias. To overcome this obstacle and scale the testbench, the language model as a judge design pattern was implemented. This workflow pattern utilizes a reasoning artificial intelligence model to objectively evaluate the performance of other systems. The complete Python source code for this evaluation process, as well as the programs for generating the visual graphs presented in the following chapter, are publicly available in the project's official repository (referenced in Section 11 Annex).

For this task, OpenAI's flagship model was selected, operating through a coordinating program. The evaluation cycle consists of three consecutive phases, beginning with the engineering of the evaluating instruction.

The most determining factor in the accuracy of an automated judge is the strict definition of its evaluation criteria. Therefore, a core structure was programmed, designed for the model to adopt the role of a mechanical engineer, forcing it to compare the system-generated response against the ground truth extracted from the original manual. As observed in Code 5, the instruction defines a scoring rubric from zero to ten, heavily penalizing the invention of data and demanding that the return of information be strictly in a structured data format containing only the reasoning and the numerical grade.

```
    You are an expert mechanical engineer. Evaluate from 0 to 10 the
accuracy of the generated response by comparing it with the actual response
from the workshop manual.
    10: Perfect response, contains all exact technical data.
    7 to 9: Correct response, but omits some minor detail.
    4 to 6: Partially correct response, but omits the key data.
    0 to 3: Incorrect response or hallucinates technical data.
    Return exclusively a structured data format with two keys: reasoning
and score.
```

Code 5. System directive for the automatic evaluator. Strict rubric to penalize invented responses.

The second phase corresponds to the data processing and structuring engine. To guarantee that the evaluating model would not alter its scores between different executions, the creativity parameter, also known in the programming of these systems as temperature, was configured to a value of zero. This configuration forces a completely deterministic behavior, ensuring that the system always gives the same grade for the same response. Furthermore, a specific functionality of the programming interface was used to force the judge to return the metric in a predictable structure that could be processed by the main program. Code 6 shows the fundamental part of this connection, where the output format is forced and the mathematical score and the reasoning text are directly extracted, eliminating the rest of the excess code.

```
    # Call the interface by forcing structured format and deterministic
response
    answer = client.chat.completions.create(
        model="gpt-4o",
        response_format={ "type": "json_object" },
        messages=[
            {"role": "system", "content": system_instruction_,
            {"role": "user", "content": user_instruction_
        ],
        temperature=0.0
    )# Automatic grade extraction and justification
    score = float(structured_result.get("score"))
    reasoning = str(structured_result.get("reasoning"))
```

Code 6. Key segment of the coordinating program. Highlights the temperature restriction to zero.

The third and final phase consists of the bidirectional data flow. The reading of the testbench and the subsequent writing of the grades was coordinated using a typical Python programming language data analysis library. The program automatically iterated through all the sheets of the results file, sending each question and answer pair to the automated judge. To guarantee the integrity of the testbench and not overwrite the wait times and text consumption previously obtained in the visual tools, this program's writing engine was configured in append mode. This allowed the code to dynamically add two new columns in the corresponding spreadsheets, one for the numerical grade and another for the judge's textual justification. In this way, it was possible to consolidate a complete dataset, prepared and ready for its analysis in evaluation section 8.

7.5 Implementation of the hybrid workflow

This section details the design of the final product that is operational in the real-use environment, a phase known in software development as the production environment. This stage represents the consolidation of the project, as it has allowed for the unification of the three technological options evaluated in the testbenches into a single, comprehensive system. This final system has been strictly bounded to respond solely and exclusively to requests related to the workshop manual of the BMW model E46 vehicle, limiting its knowledge base to the information contained in this specific document to guarantee greater mechanical accuracy and avoid out-of-context responses.

The logical core and main technical contribution of this project reside in the creation of the dynamic coordinator, also known as a router. As justified in the analysis phase, forcing all user queries through a single architecture is highly inefficient in terms of costs, especially when processing complete documents, or very limited in terms of performance, as occurs with the modular search when attempting to resolve complex breakdowns involving several steps that intertwine different pages of the manual.

To overcome this obstacle, an intelligent decision flow was implemented within the coordination tool. This system has the capacity to intercept the user's request, evaluate the complexity of its meaning, and automatically direct it towards the most appropriate technological alternative for that specific case.

7.5.1 Implementation of the router

As presented in Figure 21, the execution logic of this coordinator is divided into three consecutive steps. The pathway captures the external request, standardizes the data, evaluates the intention of the question using artificial intelligence, and splits the path through a decision component.

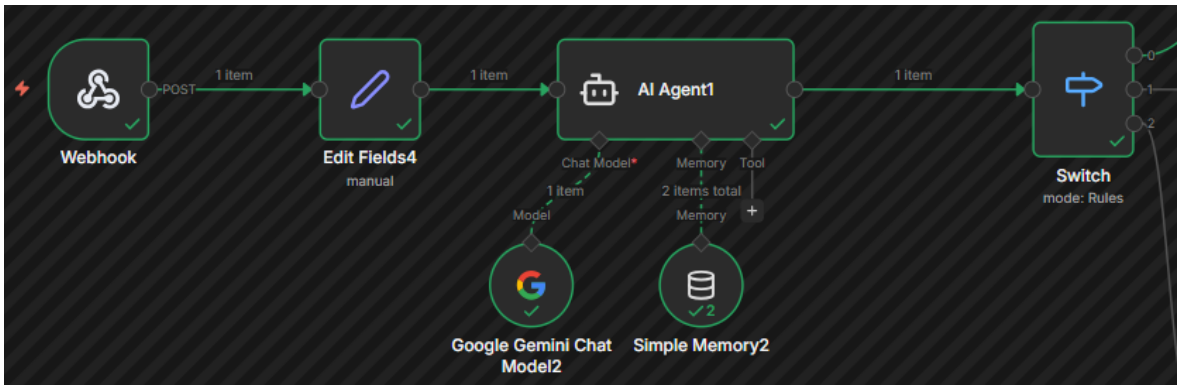

Figure 21. Network schema of the hybrid coordinator.

The first step consists of data reception and standardization. The cycle begins when the user interface issues a secure information transmission request. An initial endpoint, designed to receive these web requests, captures the signal and transfers it to a field editing tool. This component acts as a normalizer, organizing the input variables, such as the question text and the user's session identifier, to ensure that the information travels with a clean and predictable structure towards the classification engine.

The second step corresponds to intention classification, functioning as the brain of the system. The decision process is not based on simple keywords, as human language presents significant ambiguity. Instead, an artificial intelligence agent was implemented. In this phase, the model does not answer the user's mechanical query, but acts exclusively as an

automatic classifier. To achieve this analytical behavior, instruction design techniques were applied, strictly limiting the output format. As Code 7 details, the injected directive forces the model to return a single exact word based on the type of question received.

```
Act as an intention classifier. Your only output must be one of these three words: SIMPLE, COMPARATIVE, or AGENT.

1. SIMPLE: The user wants a direct technical fact, a tolerance value, or a definition.

2. COMPARATIVE: The user wants to see differences or compare two parts and systems.

3. AGENT: The user describes a problem, a symptom, or asks for a step-by-step procedure.

Formatting rules: Return exactly one word. Do not use punctuation marks or filler text. If in doubt between SIMPLE and AGENT, choose AGENT.
```

Code 7. System rules injected into the router.

Additionally, to provide the coordinator with coherence in long conversations, a memory module linked to the session identifier was integrated. This allows the classifier to remember the context of the conversation. If a user first asks how to disassemble the water pump and, in a second message, adds a question about what the tightening torque is, the memory prevents the system from losing focus on the analyzed part, allowing this second question to be correctly classified as a direct technical query.

The third and final step is conditional routing. Once the agent evaluates the query and issues its verdict, the data reaches a decision node configured with logical rules. This component evaluates the exact text string returned by the artificial intelligence. If the output is the keyword for simple queries, the flow is directed towards the first architecture supported by the external vector database. If the word indicates a comparison, it is routed towards the second architecture using the Google provider's integrated search engine. Finally, if the output requires complex reasoning to resolve a symptom, the request is transferred to the third architecture to perform a complete analysis of the document.

This organization precisely guarantees that computing resources with higher costs and processing times are only used when the complexity of the mechanical diagnosis truly demands it.

7.5.2 Implementation of the modular architecture

When the coordinator classifies the user's query as a request for direct technical data, such as tolerance values, tightening torques, or maintenance fluid capacities, the logical decision routes the data packet towards the first output, triggering the flow of the modular architecture. Unlike traditional linear processes, this pathway was not built as a simple question-and-answer chain; rather, it was designed under the concept of autonomous agents—that is, artificial intelligence programs with the capacity for reasoning, decision-making, and tool utilization.

To understand the transactional operation of this pathway, it is fundamental to recall its dependence on the document preparation phase explained in previous sections. Execution in the real environment requires that the technical manual has been previously divided into small text blocks, converted into mathematical coordinates using OpenAI's technology, and permanently stored in the external vector database, thus generating the index upon which the search engine will operate. As presented in Figure 22, the use of this artificial intelligence agent directing the information retrieval can be observed.

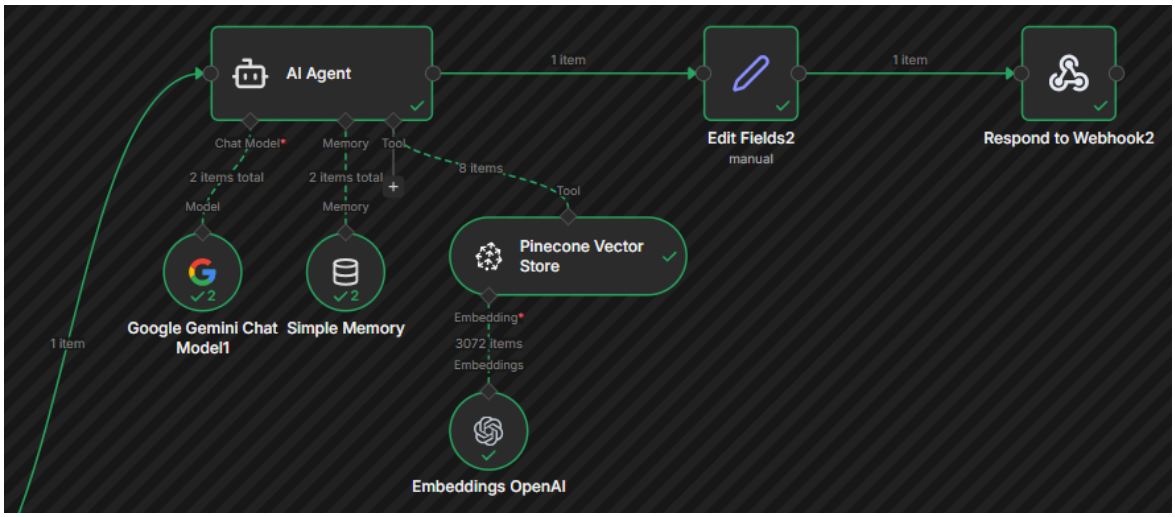

Figure 22. Transactional execution flow for the modular architecture in the real environment.

The behavior of this ecosystem is governed by the configuration of its internal components. The analytical brain is powered by Google's primary model. To guarantee technical rigor and nullify the creativity bias that causes the system to invent non-existent data, the temperature parameter was forced into a strictly conservative state, assigned a value of zero.

To maintain the conversational thread, a transactional memory module dynamically linked to the user's session identifier was integrated. This persistent memory was configured with a retention limit of the last five interactions, allowing the user to make chained queries. For instance, the user can ask for the capacity of the oil pan (the lower reservoir responsible for storing engine oil) and, in the following message, ask what type of oil it takes, without losing the vehicle reference or forcing the system to start from scratch.

The foundation of the agent's behavior was defined through a strict role directive. As shown in Code 8, these instructions compel the model to prioritize retrieved information and categorically reject the use of knowledge previously acquired during its general training.

```
Role: Expert automotive technical assistant.

Objective: Answer the user's maintenance questions EXCLUSIVELY using the provided database.

Rules:

1. Ground truth: Every technical response must be based on the retrieved data.

2. Behavior: Be precise and technical. Cite specific values or warnings found in the text.

3. Exceptions: Only if the tool returns empty results, respond that you cannot find that specific information in the uploaded manual.
```

Code 8. Main agent instruction for the modular architecture, establishing operational boundaries.

The true potential of this system resides in how it connects with the mathematical database. This external storage tool does not act as a passive search engine to which text is simply sent; instead, it is integrated as an active utility at the agent's disposal. In its internal configuration, the index was parameterized to retrieve the eight most relevant chunks per query, processed in real time. For the agent to understand when and how it should use this database, a semantic description was programmed into the tool's properties, as detailed in Code 9.

```
MAIN KNOWLEDGE BASE. Use this tool to search for ANY technical query.
IMPORTANT: When searching, extract only the main keywords from the
user's question. Do not search using the complete conversational sentence.
```

Code 9. Semantic description of the database tool.

Finally, once the agent extracts the keywords, queries the database, retrieves the pertinent chunks, and synthesizes the final response, the information flow converges in a field editing tool. This component collects the generated text and structures it into a standardized data format. As a final step, a web response component closes the secure communication cycle, sending the formatted data packet back to the visual interface so that it can be read by the end user, thus concluding the request.

7.5.3 Implementation of the managed architecture

When the semantic classifier determines that the user's query requires a system comparison, the evaluation of differences, or the analysis of advantages and disadvantages between components—for instance, when comparing two different engine models—the decision flow routes the request to the coordinator's second output, triggering the architecture. Unlike the modular approach, this pathway completely dispenses with external databases and intermediate agents, delegating information retrieval directly to the cloud provider's integrated infrastructure.

As presented in Figure 23, the design of this solution demonstrates a simplified and linear pathway composed of the call to the main model, data editing, and the final response to the interface.

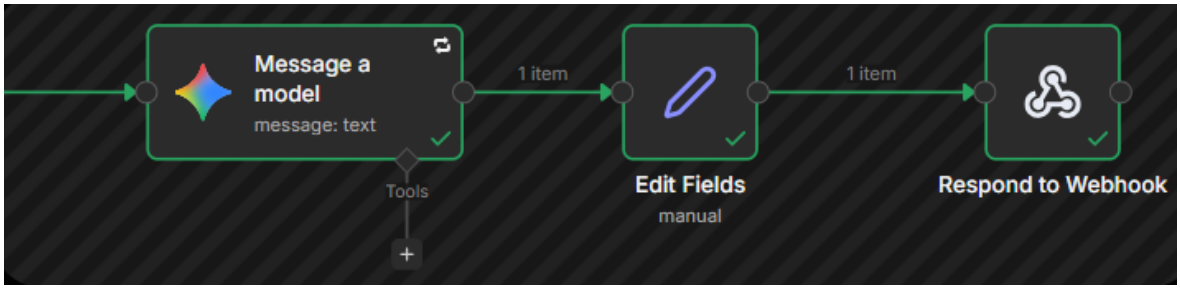

Figure 23. Execution flow for the managed architecture.

To understand the operation of this flow in the real-world usage environment, it is necessary to recall its dependence on the prior preparation phase. Execution requires that the technical manual has been previously uploaded to the provider's servers via its communication interface, generating a unified document store and a unique identifier code.

Query processing is organized through three fundamental steps. First, the connection with the main engine and the data store is established. Instead of an autonomous agent, a direct communication tool configured with Google's artificial intelligence model is used. The user's question is dynamically introduced by retrieving the standardized variable at the beginning of the flow. The key element that transforms this conversational tool into a document search system is the activation of the integrated file reading functions. The exact identifier code of the knowledge base generated in the preparation phase is introduced, which forces the system to search for the data within that specific document hosted in its cloud before drafting the final text.

Next, the behavioral instructions are configured. Given that the provider's internal retrieval operates as a closed and opaque system, known as a black box, where the developer cannot fine-tune the size of the read text chunks or the mathematical similarity metric,

quality control relies entirely on the provided commands. A strict message was configured to mitigate the risk of data fabrication and force high mechanical precision. As Code 10 shows, the instructions impose rules and compel the model to adhere exclusively to the context.

```
You are a technical documentation assistant for mechanics. Your objective is to answer based SOLELY on the provided context.

CRITICAL RULES:

1. STRICT RELEVANCE: Answer only what is asked. Do not add unrequested extra information.

2. PRECISION: If the context contains specific measurements, such as capacities or gaps in millimeters, you must use them exactly as they appear in the original manual.

3. UNKNOWN: If the answer is not in the text, clearly indicate that you cannot find that information in the provided document.

Response format: Provide a direct response in plain text.
```

Code 10. System instruction for the managed architecture.

Third and finally, the standardization and secure return of the information are performed. Once the server processes the hidden search and returns the generated text, the flow advances towards the output through two data manipulation steps. On one hand, a structural editing tool exclusively extracts the final drafting of the response, discarding the provider's internal diagnostic information, and assigns it to a new clean variable. On the other hand, the return data packet is constructed. To guarantee the integrity of the information during its transit over the network, especially when handling texts with multiple line breaks or special characters typical of a technical manual, a standardized text conversion function is applied. This function packages the content, ensuring correct readability, which allows the website to receive and display the response to the user without visual errors or formatting breaks, securely closing the communication.

7.5.4 Implementation of the long context architecture

When the hybrid coordinator detects a query based on the resolution of complex problems, breakdown symptoms, detailed technical procedures requiring consecutive steps, or steps that require intertwining several points of the manual, the system routes the request to the third output, activating the long context architecture. Unlike the previous options that operate by searching for fragments in organized databases, this technology processes the information through a total analysis, introducing the complete manual for each question. Because the system operates under a model without inter-session file memory, the platform does not save the document permanently. This introduces an operational requirement: the user must attach the file in the same message in which they make the technical query.

To manage this need, the flow is divided into an initial validation phase and two possible execution routes. As identified during development, the n8n coordination tool tends to lose track of the file when passing through decision components. For the artificial intelligence engine to be able to analyze the document, it was necessary to integrate an intermediate program in the JavaScript language that retrieves the original file from the entry point. As detailed in Code 11, this program acts as a bridge, restoring the document in the data packet and creating a logical control variable to confirm whether the file is present or not.

```
// 1. Find the file at the starting point
const inputData = $('Webhook'). first();
const fileData = inputData.binary;

// 2. Verification of the existence of the document
if (fileData && Object.keys(fileData).length > 0) {
  // If there is a file: restored and marked as validated
  $input.item.binary = fileData;
  $input.item.json. hasFile = true;
} else {
  // No file: is marked as unvalidated
  $input.item.json. hasFile = false;
}
return $input.item;
```

Code 11. File validation and restoration program.

Following this validation, a decision component evaluates the result. If the response is true, the successful execution route is activated, presented in Figure 24. In this pathway, the system transfers the complete manual and the user's text to the Google Gemini document analysis tool.

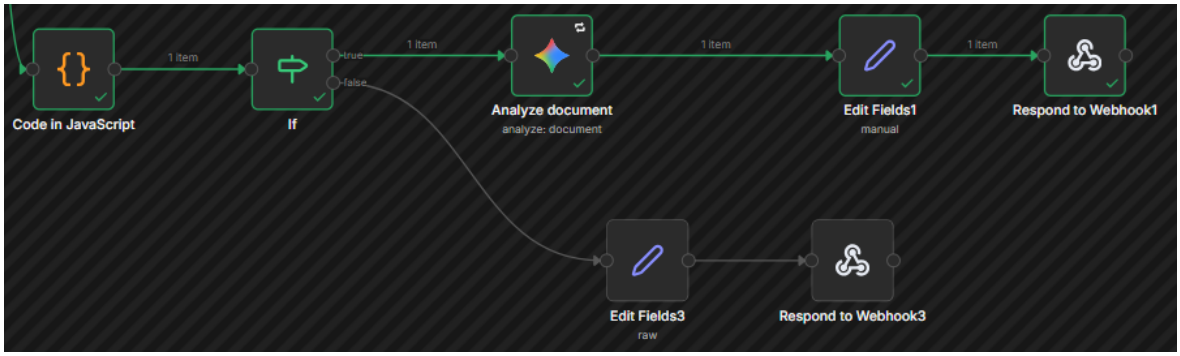

Figure 24. Main execution route for the extensive context architecture.

At this point, the artificial intelligence engine is configured to read the dynamic file and receives a system instruction designed to prioritize professional technical diagnosis. As Code 12 shows, fidelity restrictions to the original text are imposed, and the use of text formatting, such as asterisks or bolding, is prohibited to facilitate a clean display of the response on the final webpage.

```
You are a high-precision mechanical diagnostics specialist. Your only source of truth is the provided manual.

Rules: Ignore your general knowledge and use exact measurements from the text. Do not add filler information.

Format: Always respond in English. Use only plain text, without bolding or formatting symbols. Separate sections with natural line breaks.
```

Code 12. System instruction for the extensive context analysis.

Conversely, if the user sends a diagnostic query but forgets to attach the required file, the system activates the error interception route shown in Figure 25. Instead of allowing the

process to advance and cause a server failure or unnecessary expenditure on the artificial intelligence tool by not finding the document, the coordinator blocks the call and activates an alternative response.

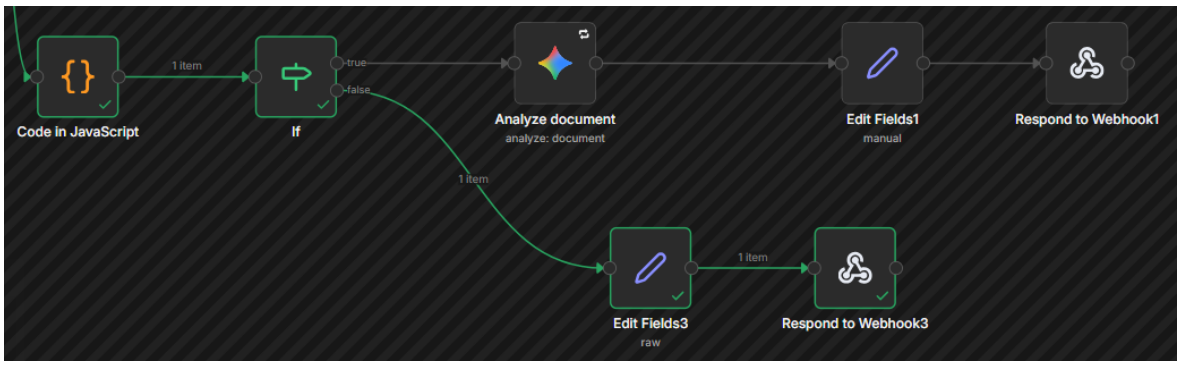

Figure 25. Error correction route. Upon detecting the file's absence, the system halts the process.

This mitigation route utilizes an editing tool to overwrite the output with a predefined warning message. As observed in Code 13, a simple data packet is generated that informs the user of the need to upload the document before continuing.

```
{
  "output": "We have not received any attached file. Please ensure you upload the manual so that we can process your query correctly."
}
```

Code 13. Data structure for the missing file warning.

Finally, regardless of the route taken, the flow concludes by sending the information back to the webpage via a secure connection, thereby closing the communication cycle and guiding the user in their interaction with the diagnostic tool.

7.6 Security plan

The design of a robust technological structure does not end with the implementation of its main functions, but rather requires protection and stability. During the installation phase on the virtual private server, a significant risk was identified in the infrastructure: by default, the n8n platform stores all workflows in an internal database located within a virtual container.

Under this initial configuration, the server and its storage system act as a single point of failure. If the internal database is corrupted or the server experiences a severe crash, all completed work would be lost, including node configurations, agent instructions, and testbenches.

Given that n8n internally structures all its workflows in JSON format, the solution to prevent information loss consisted of dynamically extracting this code and automatically synchronizing it with a private GitHub repository. This integration goes beyond a simple backup. By keeping the project under version control, any failure caused by a VPS server crash or errors in the n8n service can be quickly recovered by reverting to a previous commit. As can be seen in Figure 26.

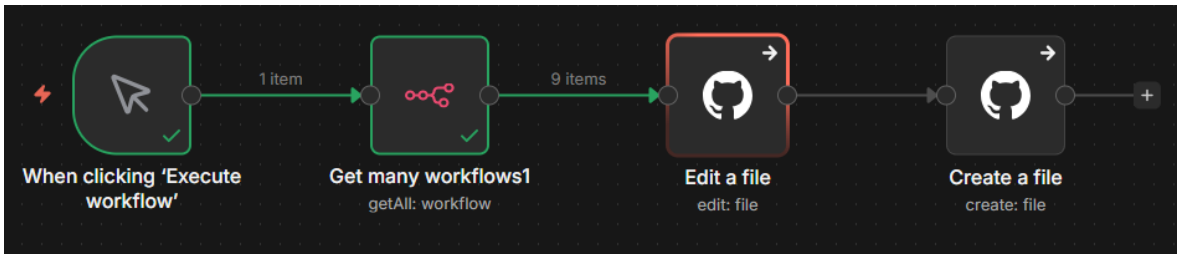

Figure 26. Security and backup process.

The execution cycle of this security plan is organized into four consecutive phases. The first step is the manual trigger. The workflow is designed to execute on demand, requiring direct action from the developer. This decision allows controlling when a new security version is generated, ensuring that data is saved exactly when the system is stable. In this way, it guarantees that the external repository always reflects correctly functioning versions, avoiding the accidental saving of incomplete processes.

The second step consists of extracting the baseline information. Instead of attempting to directly access the server's internal files, the workflow utilizes an internal connection known as an application programming interface (API). This tool queries the platform and downloads a dataset containing the technical definition of all existing workflows.

The third step corresponds to the continuous update module. To avoid file duplication or the loss of change history, the workflow follows an intelligent synchronization logic. The system attempts to perform a commit on the corresponding file in the remote repository. If the workflow already existed, the system records only the differences in the code, keeping the evolutionary history of that specific process intact.

The fourth step is the new file creation module. To manage the system's growth, the flow includes a control route for newly created processes. If the previous update step fails because the file does not yet exist in the external repository, the system does not stop. Instead, the workflow advances to this second step configured for creation. This component is responsible for generating the new code file for the first time.

Thanks to this design organized into update and create steps, the system guarantees that the state of the external backup is always a secure reflection of the actual installation on the server. This sequential execution logic was chosen because, in the version of the tool with which the project was initiated, there was no direct automatic update or creation function. This operation is common in the field of databases to modify a record or generate it if it does not exist, a concept technically known as an upsert (a term resulting from the combination of the English words update and insert). Lacking this option natively, the necessity was resolved through this coordinated execution of tasks. In this way, the risk of losing information in the event of a technical failure is eliminated, and all the project's work is completely protected.

7.7 Deployment in production and final product

As a result of the development phase, the system based on the hybrid router has been deployed in a publicly accessible production environment. This deployment allows real-time interaction with the mechanical assistant through the project's digital space (link in Section 11 Annex).

For the real-world usage environment, a conversational-style web interface has been designed, implemented under the Next.js development framework and stylized with modern

visual design tools. The aesthetic aspect prioritizes ease of use and the reduction of mental effort for the operator, incorporating a central messaging area, support for attached documents, and visualization modes adapted to lighting.

It is necessary to highlight the temporary nature of information management in this interface. Although the system momentarily relies on a cloud database to maintain the conversation thread, due to privacy by design principles, the lifecycle of this history is tied exclusively to the active browser session. Upon reloading the page or closing the window, the information is irreversibly erased from the server, guaranteeing the privacy of user data.

Regarding infrastructure and networks, the internal system is hosted on a self-managed virtual private server. To guarantee the confidentiality of data in transit, communications between the client and the server strictly operate under secure encryption using the HTTPS protocol, backed by security certificates. Unlike what was done during the testbench, where the paid modality was used to obtain stable results in a controlled environment, the final product in production operates under the free tier of the Google Gemini interfaces. This decision was made because the open usage environment does not allow total control over the volume of queries; therefore, the restriction of twenty daily requests offered by this zero-cost option has been chosen to prevent unexpected resource usage.

Given its deployment as a publicly accessible web application, the interface does not act merely as a presentation layer, but as the system's first line of defense. To protect the integrity of the coordinator and prevent the improper consumption of resources, four fundamental security vectors have been implemented:

- Network saturation mitigation: For the analysis of extensive documents to function without putting the server at risk, a strict limit of 65 megabytes was established for uploaded files. This stops at the source any attempt to overload the system's memory or wait times through excessively large files.
- File identity validation: To prevent the entry of camouflaged malicious programs, the system does not rely on the file extension. Instead, the system inspects the binary header, known as the file's DNA, to certify that it is a valid PDF document before processing it.
- Code injection prevention: Strict sanitization of user inputs is applied using regular expressions before processing the information. The system neutralizes and eliminates any attempt to introduce tags or commands that could execute programs in the browser in order to reach the server's internal information.
- Defense against instruction manipulation: In the face of attempts to deceive the artificial intelligence into ignoring its rules or responding to topics unrelated to the project, the architecture maintains its robustness. The strict anchoring to the technical manual ensures that the system rejects requests outside the domain of automotive mechanics.

To guarantee the adoption of the system in a real industrial environment, the user interface has been designed under the principle of minimal difficulty of use. The objective is that any operator, regardless of their computer literacy, can interact with the model seamlessly. The visual organization of the application is divided into several functional components.

First, the central area for interaction and quick access. Upon launching the application, the user is greeted by an uncluttered screen that includes suggestion cards with predefined queries. By clicking on any of them, the text is automatically entered into the typing bar,

streamlining the most common diagnostic tests. As presented in Figure 27, the clean design and the cards configured to activate the coordinator's different pathways can be appreciated.

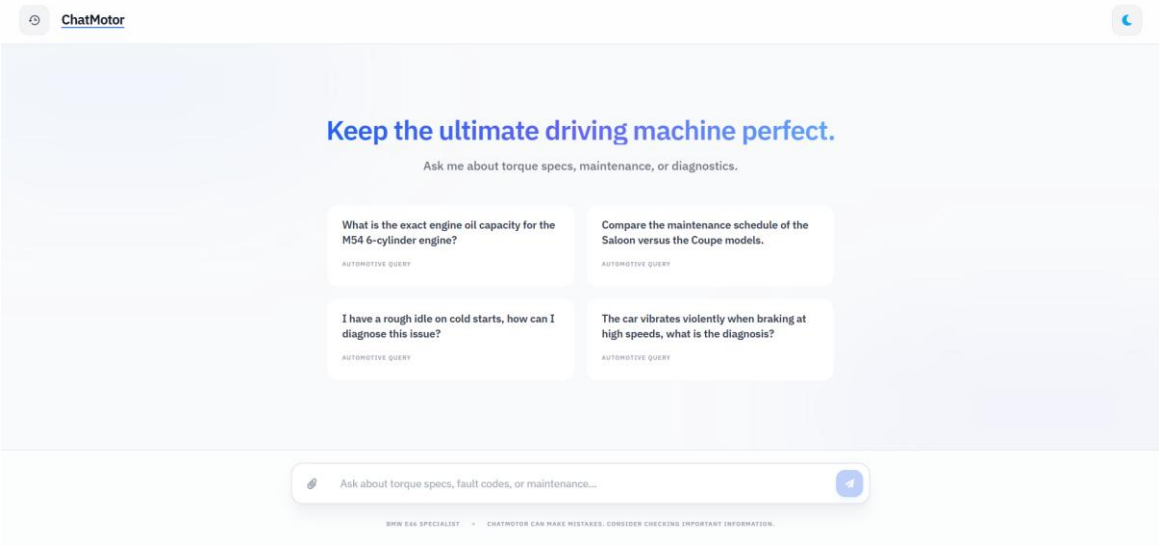

Figure 27. Main view of the mechanical assistant interface.

Second, the restricted input bar located at the bottom. This allows formulating queries in natural language. For security reasons and to maintain the focus of the project, the attachment button limits uploads exclusively to files in PDF format, blocking any other extension from the user side even before performing internal security validations.

Third, the side panel for conversation management. The system features a menu that acts as a control center for temporary storage. Within this panel, the user has tools to initiate new dialogues with a blank context, a search engine to filter active conversations, and the ability to delete specific threads that are no longer necessary. As observed in Figure 28, this menu centralizes the management of active queries.

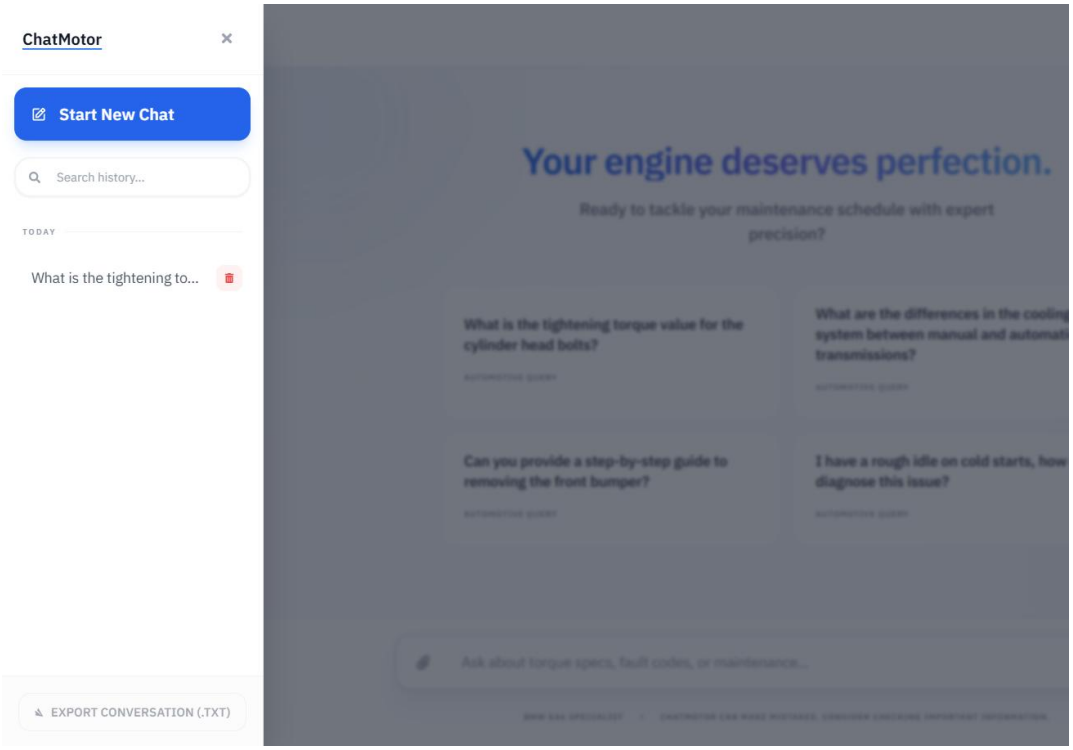

Figure 28. Deployment of the side menu.

Fourth, the knowledge export module. Following the principle of temporary storage, the side panel allows downloading the complete history of a diagnosis to the user's device. This function packages the entire conversation into a plain text file, guaranteeing that the user does not lose the technical information after closing the session on the server.

Finally, to ensure visual comfort in different environments, such as the variable lighting of a mechanical workshop, the interface integrates a quick switch to toggle between light mode and dark mode, reducing the user's visual fatigue. Figure 29 shows the adaptability of the design for low-light environments.

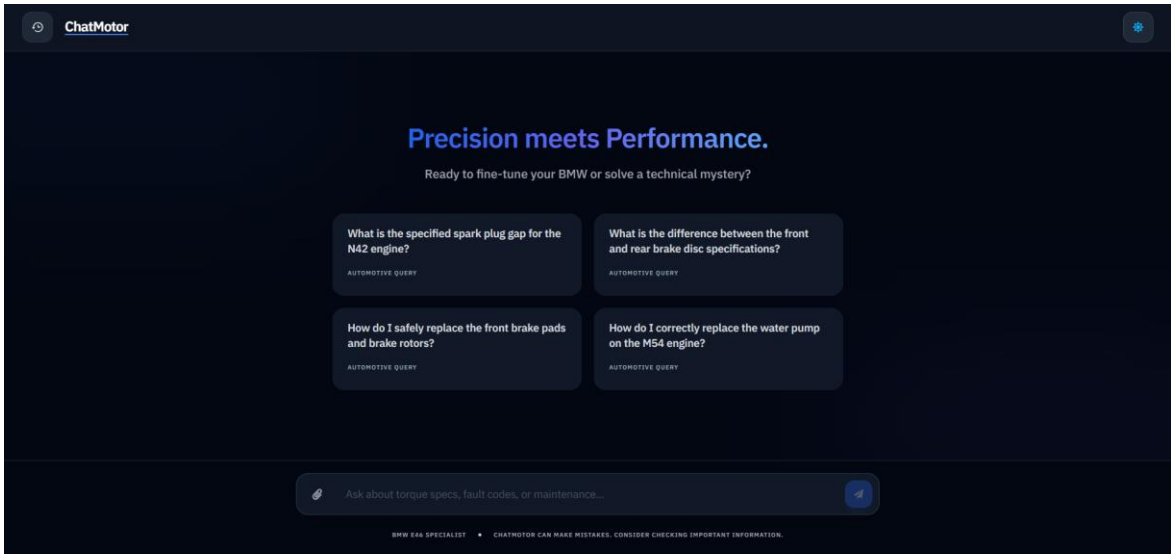

Figure 29. Adaptability of the visual design.

7.8 Challenges and technical solutions

The development of this project was a constant struggle against limitations and errors. Both when building the testbench and when taking the system to production, provider blockages and network bottlenecks arose. To meet the objective of having a robust program that covered both parts of the analysis, I had to make several design decisions. Below, I explain the most severe obstacles and the solutions I implemented.

Problem: When automating the battery of questions to evaluate the three architectures, the process would abruptly stop. The free accounts for the artificial intelligence models had a strict limit of twenty queries per day or blocked access if they detected requests that were too frequent. This made it impossible to execute the complete testbench without suffering mandatory halts.

Solution: I had to register a payment method on the provider platforms to unlock the professional usage tiers. This eradicated the requests-per-minute restrictions and allowed me to launch hundreds of queries continuously, measuring the real performance of the system without artificial bottlenecks.

Problem: One of the most prominent problems while implementing the testbench was that I attempted to send all 20 questions at once to the AI, assuming it would handle answering each question separately, but this was not the case.

Solution: The solution was to redesign the testbench workflow and add a loop, strictly dedicated to sending each question separately; until the cycle was completed and the results

were correctly persisted in the spreadsheet, it would not send the next question for evaluation.

Problem: My initial intention in the coordination tool was to build the entire project using solely Google's technology. I wanted both the conversational model and the text-to-numerical conversion system to be from the same company. However, this proved unfeasible. When attempting to process the document to store it in the vector database, the provider's system constantly blocked me due to its strict size limits, preventing the upload of the complete file.

Solution: After researching how the developer community resolved this step, I noticed that almost all used OpenAI's technology. Initially, I looked for other free alternatives, as this platform requires a minimum initial deposit of five euros, and my intention was to avoid additional expenses. But, just as occurred with the testbench, I understood that to obtain reliable performance metrics, I needed to operate in a real, unrestricted environment. Upon activating the paid account and integrating this new model, the data was uploaded to the Pinecone database on the first attempt, and I experienced no further reading failures for the remainder of the project.

Problem: Once in the real environment, when sending the complete manual of over three hundred pages to the full reading model, the system took almost forty-five seconds to generate the response. The web server severed the connection at thirty seconds by default, returning a timeout error to the user. Furthermore, the baseline configuration blocked uploading files larger than one megabyte, and my document weighed sixty-five.

Solution: I accessed the internal configuration of the virtual machine. I increased the timeout to one hundred and twenty seconds and removed the size restriction on the visual interface. With that, the connection held until the model finished reading all the text.

Problem: The program uses a model to decide where to route each question. Sometimes it fabricated the classification and sent easy queries down the most expensive path. Worse still, sometimes it responded with normal text and other times with a broken data structure, causing the spreadsheet saving process to fail.

Solution: I rewrote the router instructions with very strict rules so it would not improvise. I also programmed an intermediate filter that reviews the response live. If the format arrives corrupted, it applies a quick conversion function to salvage the data before saving it.

Problem: I needed to record the consumed text units to calculate the money spent on each query. Working with closed programming blocks that manage data internally made it impossible to extract the exact consumption returned by the provider in real time. The system operated blindly, and I could not know the cost of the operations.

Solution: First, I added an intermediate step in the code to calculate an estimation of the text size before sending it. Then, upon reviewing the cloud provider's official invoice, I applied reverse engineering to adjust the expenses. The main objective was to isolate and discover exactly how many text units the three-hundred-page manual consumed. With that clear figure, it became much easier to calculate the cost of the modular, managed, and full reading architectures, where the greatest economic impact came from having to upload the entire file for every question. I repeated this same manual calculation technique to measure the expenses with the smaller documents in the rest of the analyses.

8 Evaluation

This chapter presents the analysis of the real data obtained upon testing the three architectures. The three architectures were executed in a serverless computing environment (AWS Lambda), discarding the n8n platform used in previous phases. This technical decision was indispensable after detecting that the low-code intermediary tool altered text extraction and added its own latencies, thereby distorting the actual performance of the artificial intelligence. Isolating the code by operating directly against the official APIs guaranteed the procurement of reliable results.

Once the results were obtained, a program in the Python programming language was developed to function as a data extraction, transformation, and loading system. Through the use of graphical libraries, this code is responsible for cleaning, organizing, and visually projecting the performance metrics. All the analysis code and the files containing the final data are available in the project's repository (in section 11 Annex).

8.1 Testing methodology

To guarantee the validity, the reproducibility of the experiment, and the objectivity of the results, a standardized testing method was designed. The three architectures—namely, the modular one with an external vector database, the provider-managed one, and the long-context one—were subjected to exactly the same operational conditions and the same dataset.

8.1.1 Dataset and question bank

To simulate a real-world usage environment and evaluate the system's performance within its area of specialization, the official manual of the test vehicle (BMW 3 Series E46) was utilized as the documentary knowledge base. From this document, a set of twenty structured technical questions was developed. These questions were designed to encompass different levels of complexity and difficulty in information retrieval, including requests for exact numerical data, such as tightening torques, comparisons between components, and reasoning regarding maintenance steps.

For each question in the testbench, the correct and verified response was manually extracted from the original manual itself. This text constitutes the system's reference response, functioning as an absolute ground truth against which the text generated by the artificial intelligence in each test is measured and compared.

8.1.2 Evaluation metrics

To obtain a comprehensive overview of the coordinator's performance and the different search alternatives, the evaluation was organized around several fundamental metrics in the design of these systems.

The first metric is the response time, which accounts for the execution seconds of each strategy in isolation. It is calculated by recording a first timestamp just before the query enters the artificial intelligence engine and a second timestamp immediately after receiving the final word of the generated response. This method excludes the prior processing time of the router, allowing for the evaluation and comparison of the actual speed of the different engines.

The second metric is the operational cost, which represents the economic expense derived from the process of generating the response. It is measured by analyzing the

consumption of text units, both in the submitted question and the received response, applying the official rates published by the cloud providers.

The third metric is accuracy and quality, which evaluates the level of fidelity, the degree of detail, and the technical correctness of the generated text in comparison to the ground truth.

The fourth metric is the retrieval success rate, consisting of a pass or fail evaluation entirely independent of the language model. It measures whether the search system was capable of locating and delivering the exact fragment of the manual containing the original answer. By cross-referencing this data with the accuracy score obtained from zero to ten, it is possible to diagnose the system's behavior, classifying the results into four possible scenarios:

- The first scenario is grounded success (context found + high score): considered the ideal situation. The search engine locates the correct fragment, and the model drafts a precise response strictly based on the data it has just read.
- The second scenario is a false positive or knowledge leakage (context not found + high score): here, the search engine does not find the information, but the model responds correctly using the data it memorized during its prior training. Although the answer is correct, this represents a critical risk in professional systems, as it demonstrates that the artificial intelligence ignores its constraints and could hallucinate data uncontrollably in the future.
- The third scenario is a reasoning failure (context found + low score): this occurs when the system locates the exact text, but the model becomes confused, drafts the response poorly, or omits key data despite having the correct information in front of it.
- The fourth scenario is a cascading failure (context not found + low score): where the search engine fails to locate the information and the model, lacking both the supporting text and prior knowledge about the breakdown, fails to respond.

The fifth metric is global efficiency, a measure that intersects intellectual performance, that is, average quality, with the economic impact or total cost. This allows for establishing the efficiency frontier, determining which architecture offers the best balance between technical precision and economic viability for production use.

Finally, the sixth metric is the global performance footprint. This measure groups all the aforementioned operational variables, such as quality, cost, speed, and reliability, projecting them onto a normalized scale from zero to ten. Its visual representation through a spider web chart allows the suitability profile of each architecture to be evaluated at a single glance. This perspective is fundamental for decision-making, as it reveals which system is more balanced, avoiding the error of solely enhancing speed at the expense of sacrificing precision or economic cost.

8.1.3 Evaluation method

To grade the accuracy metric and the technical quality of the responses, a manual human review was initially proposed, cross-referencing the generated texts with the reference response from the manual. However, during the initial tests, a clear presence of human bias was detected. Upon receiving well-written responses with good structure and a confident tone from the models, the natural tendency of the human evaluator was to assign near-perfect scores of nine or ten out of ten, thereby masking the actual precision differences between the various architectures.

To provide the study with maximum rigor and eliminate this perception error, the use of an artificial intelligence model as an automated evaluator was adopted. This concept represents the current standard for grading large volumes of responses without human intervention.

The primary model from one of the leading providers was utilized through its official interface, configured with a strict system instruction and a zero creativity parameter to ensure fixed and predictable responses. The model acted as an impartial computational judge, comparing the system-generated response with the reference response and applying a severe penalty for the slightest fabrication of data or omission of critical information, such as variations in tightening torques or engine fluid capacities. This adjustment allowed for much more critical, detailed, and realistic scores, clearly revealing the deficiencies and hidden data fabrications in the managed architectures compared to the modular solutions.

Finally, this automated grading method demonstrated high operational efficiency. The total cost of executing the evaluation instruction, processing the sixty cross-responses generated by the three architectures alongside their respective reference responses, was an expense of barely twenty cents. This objective data proves that using a language model as a judge not only eliminates human biases and allows the testbench to scale easily, but it does so with a minimal economic impact compared to the hours of manual auditing required by a professional.

8.2 Comparative analysis with extensive document

This section presents the results obtained following the execution of the automated testbench on the three evaluated architectures using the complete manual. The metrics analyzed in this initial phase correspond to the operating cost and the response time. These data have been extracted directly from the billing records of the programming interfaces of the utilized providers.

To guarantee the total accuracy of the data and prevent any blockage due to usage limits, the free tier of these services was completely deactivated during this stage. Consequently, all tests were executed under the platforms' official paid plans, generating real commercial expenses that were directly covered with private funds. This decision ensures that the consumption figures presented reflect the authentic economic cost that the system would incur in an unrestricted professional environment.

8.2.1 Analysis of operating costs

The operating cost analysis determines the system's viability in a production environment. The evaluation focuses exclusively on the consumption of basic text units billed by the language model (Gemini 3.0 Flash Preview) when processing the sixty queries of the testbench (twenty questions for each architecture). The economic calculation is performed by applying the provider interface's official rates to the processed volume, set at 0.41706 euros per million input text units and 2.68168 euros per million output text units. The results obtained for the three architectures are detailed in Figure 30.

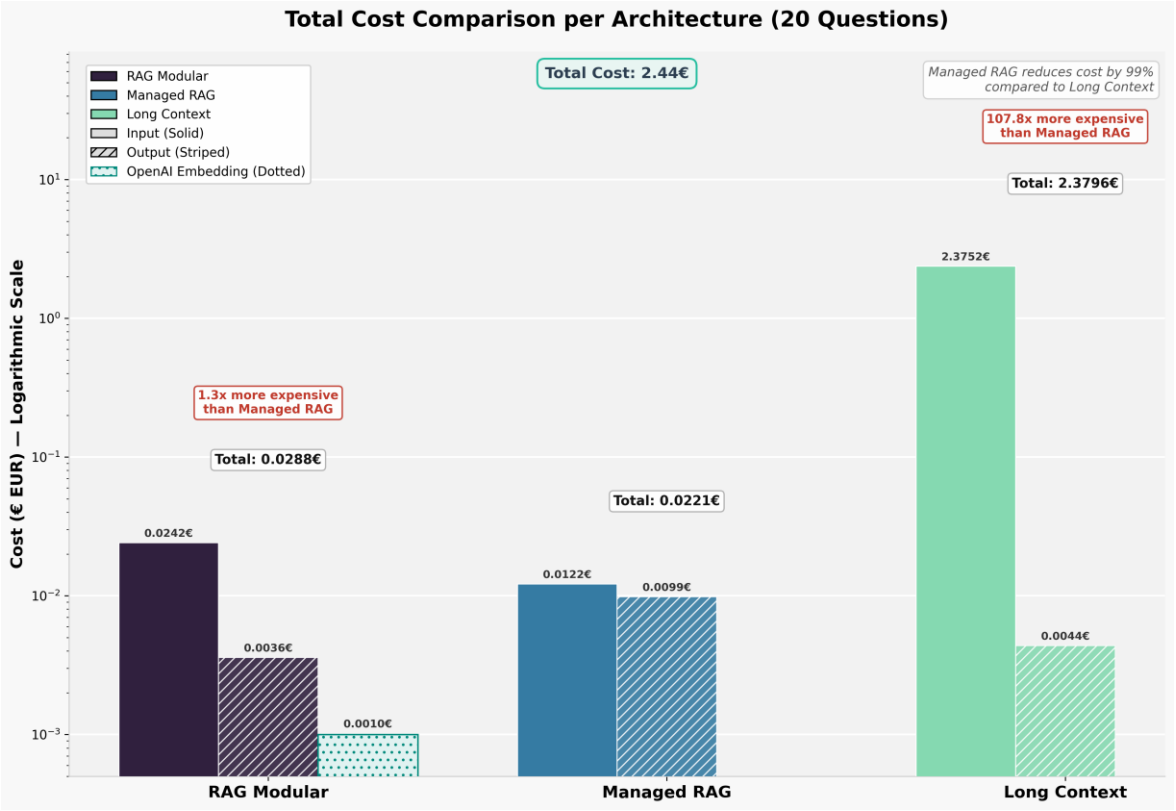

Figure 30. Comparison of total operating cost on a logarithmic scale for the processing of twenty queries.

The data show a highly pronounced difference in consumption among the three approaches. The managed RAG architecture presents the lowest operating expense in the comparison, with an approximate total cost of 0.02 euros. By delegating the indexing process to the provider's infrastructure, it internally optimizes the quantity of text units retrieved to formulate the response. The cost derived from the prior data ingestion is minimal and does not significantly affect the total calculation.

The modular RAG architecture presents a slightly higher cost than the managed approach. This increase is due to the requirement of performing data ingestion into the external vector database (Pinecone), which introduces an initial expense when using an OpenAI vectorization model. Although this additional cost occurs solely at the beginning, during the ingestion process, it exceeds the expense of the managed architecture. Furthermore, the consumption of input text units is greater during queries, given that the system must retrieve the text fragments from the vector database before the model analyzes the information and generates the response.

The long context approach records the highest cost, as it requires sending and processing the complete 300-page document in every request. This large volume of text units limits the architecture's viability in real-world environments. If a 300-page manual generates an approximate expense of 2.40 euros, processing documents of 1000 pages or more would escalate the computational cost to inefficient levels.

The results confirm that sending the complete document in each request is the least viable alternative for production environments with extensive manuals. Conversely, the implementation of managed RAG systems proves more efficient and facilitates operability for companies, assuming as a tradeoff the loss of direct control over the search algorithm.

8.2.2 Analysis of the response times

Response time is a determining factor in user experience. For this evaluation, a five-second limit is established as the standard to maintain fluency in conversational assistants. The obtained results show significant variations among the architectures, as detailed in Figure 31.

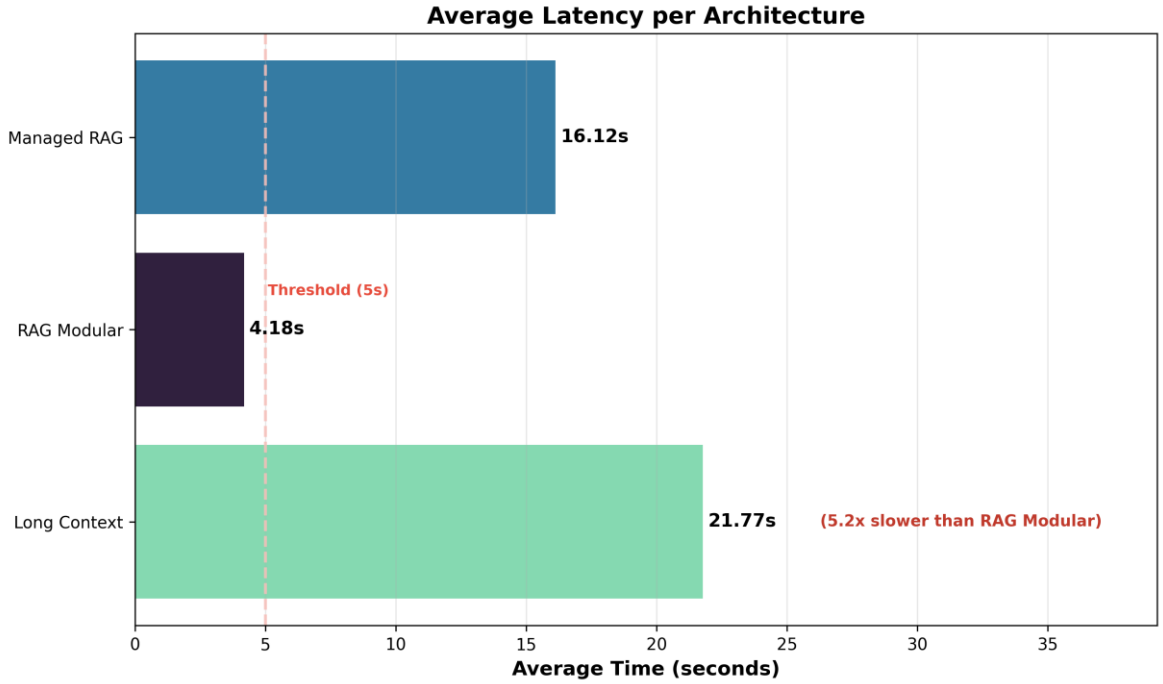

Figure 31. Average response time per architecture, in seconds.

The modular RAG architecture is positioned as the only solution that complies with the fluency standard, recording an average time of 4.18 seconds. This performance validates the efficiency of the design, demonstrating that the orchestration between the external vector database (Pinecone) and the inference engine minimizes retrieval latency. Despite the communication between services, it manages to retrieve the information rapidly and within the tolerance margins.

The managed RAG presents a latency of 16.12 seconds. This increase triples the established tolerance limit. The delay is attributed to the provider's internal computational overhead when performing the search and ranking cycle opaquely. These times limit the use of this architecture in applications that require real-time interactivity, as the processing of the managed system severely penalizes the agility of the conversation.

The long context approach records the highest response time, with an average of 21.77 seconds, when processing the 300-page technical manual in its entirety during each interaction. This latency is critical for commercial deployment, as the user must wait more than twenty seconds to receive each response. Unlike retrieval-based architectures, the extensive processing of input text units prevents maintaining an acceptable operational latency in large documents.

In conclusion, the performance analysis confirms that the modular RAG architecture is the most balanced option for a production environment. While the long context systems and the managed solutions introduce latencies that degrade the user experience, the modular design guarantees the fastest response.

8.2.3 Quality and reliability

Unlike the measures of cost and latency, the evaluation of the technical accuracy of the texts requires an automated analytical method. The presented data is obtained through a language model acting as a computational judge. This evaluator operates with a zero creativity parameter, guaranteeing a strict and neutral analysis when comparing the generated responses with the reference response extracted from the technical manual, assigning a score from zero to ten.

8.2.3.1 General performance analysis

The first step of the evaluation consists of analyzing the average capacity of each architecture to formulate a correct response to the twenty questions of the testbench. The results of this cross-audit are represented in Figure 32.

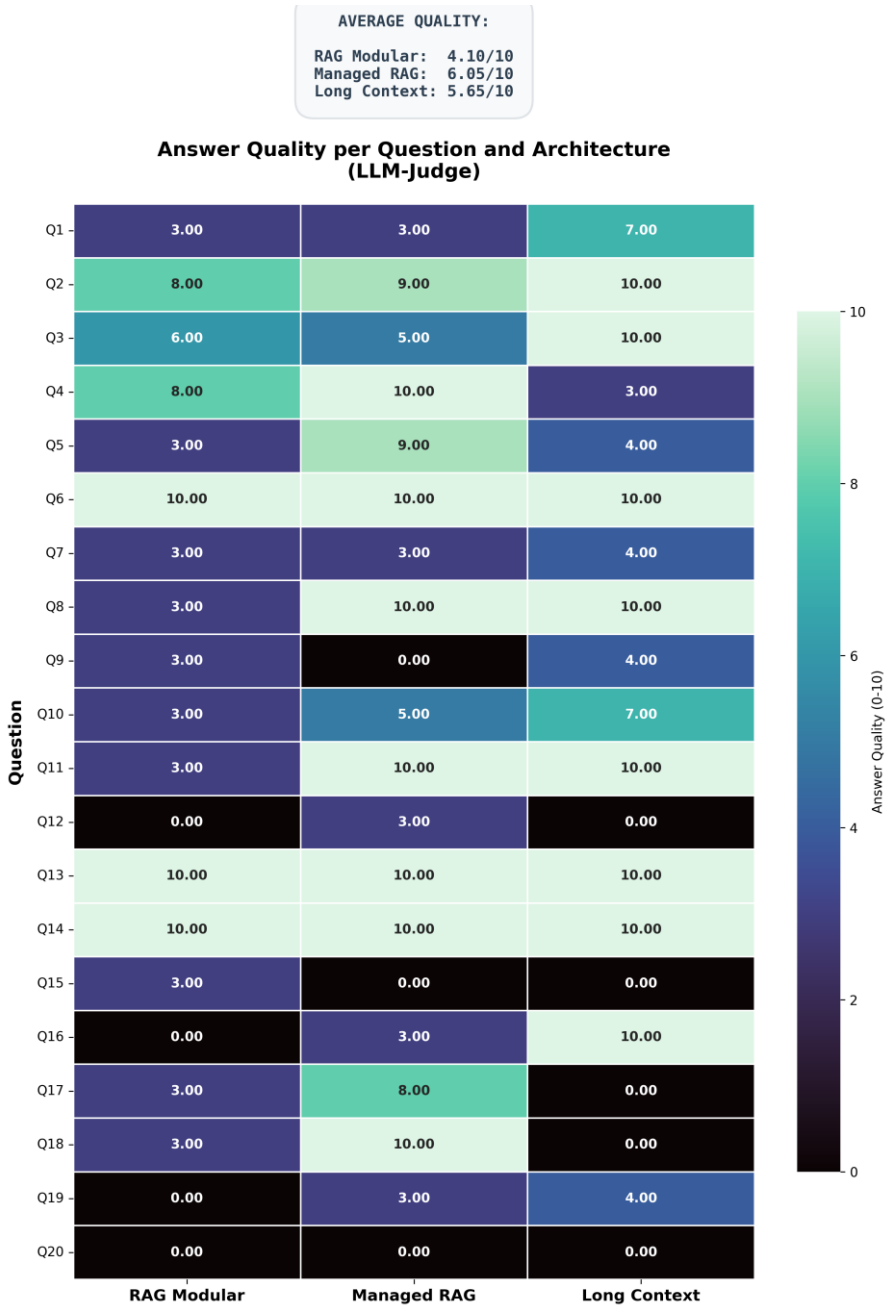

Figure 32. Heat map representing the quality of the responses, with a score from 0 to 10.

The extracted data shows a functional behavior different from what was expected. The managed RAG architecture obtains the highest average rating of the study, with a 6.05 out of 10. The provider's infrastructure demonstrates high efficiency in indexing the structure of the workshop manual. Its internal search algorithms succeed in locating and retrieving the appropriate technical context with greater consistency.

The long context approach is positioned in second place with a 5.65. Although the model receives the complete document in each request and does not depend on a retrieval engine, the score reveals a technical degradation. This performance indicates that forcing the artificial intelligence to process hundreds of pages simultaneously causes saturation in reasoning, hindering the localization of the exact data compared to systems that deliver already filtered information. Processing 300 entire pages in each query generates context fatigue in the model; during the final questions of the test, data accumulation causes a loss of precision that results in retrieval failures.

The modular RAG architecture presents the poorest performance, dropping to an average of 4.10. The heat map evidences multiple queries where the system obtains a null score. This metric demonstrates that the fragmentation and vectorization strategy configured in the external database (Pinecone) presents limitations in capturing the semantic relationship in complex queries. By restricting the system to a single search round where it only retrieves the ten fragments with the highest similarity, the model runs the risk of receiving insufficient or irrelevant information. If the architecture were optimized by configuring multiple iterations (for example, raising it to eight search rounds while maintaining the limit of ten fragments per cycle), the system would query the vector database repeatedly. This would allow it to progressively accumulate up to eighty text fragments, expanding the injected knowledge base and providing the necessary data to formulate a correct response, although, as the number of rounds increases, so does the cost and latency; this must also be taken into account.

The quality analysis confirms that the design of the search engine is the most critical factor of the system. In this scenario with technical documentation, delegating the indexing to a managed RAG solution ensures a higher success rate compared to a modular development. Likewise, the data demonstrates that providing total access to the document does not guarantee the maximum score, showing clear symptoms of attention fatigue in the language model.

8.2.3.2 Search engine hit rate and data reliability

The quality of the responses does not guarantee the reliability of the system in a technical production environment. It is fundamental that the generated information proceeds exclusively from the provided knowledge base. To audit this behavior, the automated judge's score is cross-referenced with the search engine's success rate. This procedure identifies whether the model utilizes the manual or external knowledge. The analysis is detailed in Figure 33.

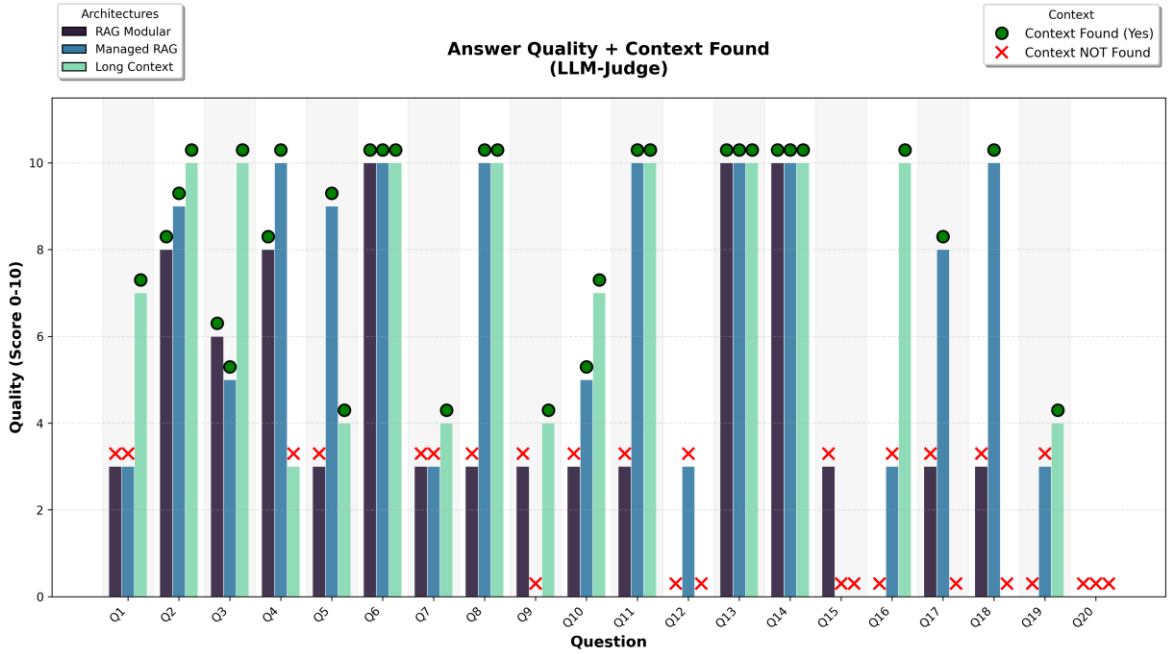

Figure 33. Cross-analysis of response quality versus search engine success.

The modular RAG architecture presents a direct correlation between search failure and a low score. The presence of error marks coincides with null quality values in the majority of the queries. This behavior defines a fail-safe mode: upon failing to locate the data in the vector database, the system lacks the information to respond and avoids the fabrication of content.

The managed RAG exhibits a superior context retrieval rate. In cases where the search engine fails, as observed in question nine, the obtained score is zero. This result confirms that the system respects the security constraints and does not resort to data external to the original document, maintaining the integrity of the response.

The long context approach records a decreasing performance as the evaluation progresses. During the initial queries, the model manages to maintain the context and provides precise responses. However, as it approaches the final questions, a degradation of the responses can be observed. Forcing the model to process hundreds of pages simultaneously in each request generates context fatigue. Although isolated retrieval peaks are recorded in the final phase, the system suffers a clear loss of attention, causing it to fail in the majority of the latter queries.

The obtained results confirm that the managed RAG architecture provides the best overall performance. It achieves the highest hit rates and strictly adheres to not fabricating information if the algorithm fails to retrieve the appropriate context, thus guaranteeing a fail-safe mode fundamental for production environments.

8.2.4 Efficiency

In the previous sections, an isolated analysis of critical variables such as cost, latency, technical quality, and reliability has been conducted. In software architecture design, viability requires the joint optimization of multiple factors. This section evaluates economic performance by mapping operational variables for the processing of the 300-page manual.

8.2.4.1 Cost-performance analysis

To visualize the trade-off relationship between technical performance and economic expenditure, the operating cost is projected against the average quality assigned by the automated evaluator. This relationship is objectively measured using the following economic efficiency formula:

$$Efficiency = \frac{\boldsymbol{Average\ Quality\ (0-10)}}{\boldsymbol{Total\ Cost\ (euros)}} \qquad \textbf{(2)}$$

This value represents the economic performance of the system, indicating the quality points obtained for each euro invested. Under this model, the optimal scenario is located in the upper left quadrant of the graph, where the score is maximized while minimizing expenditure, indicating the most sustainable topology for a real-world deployment.

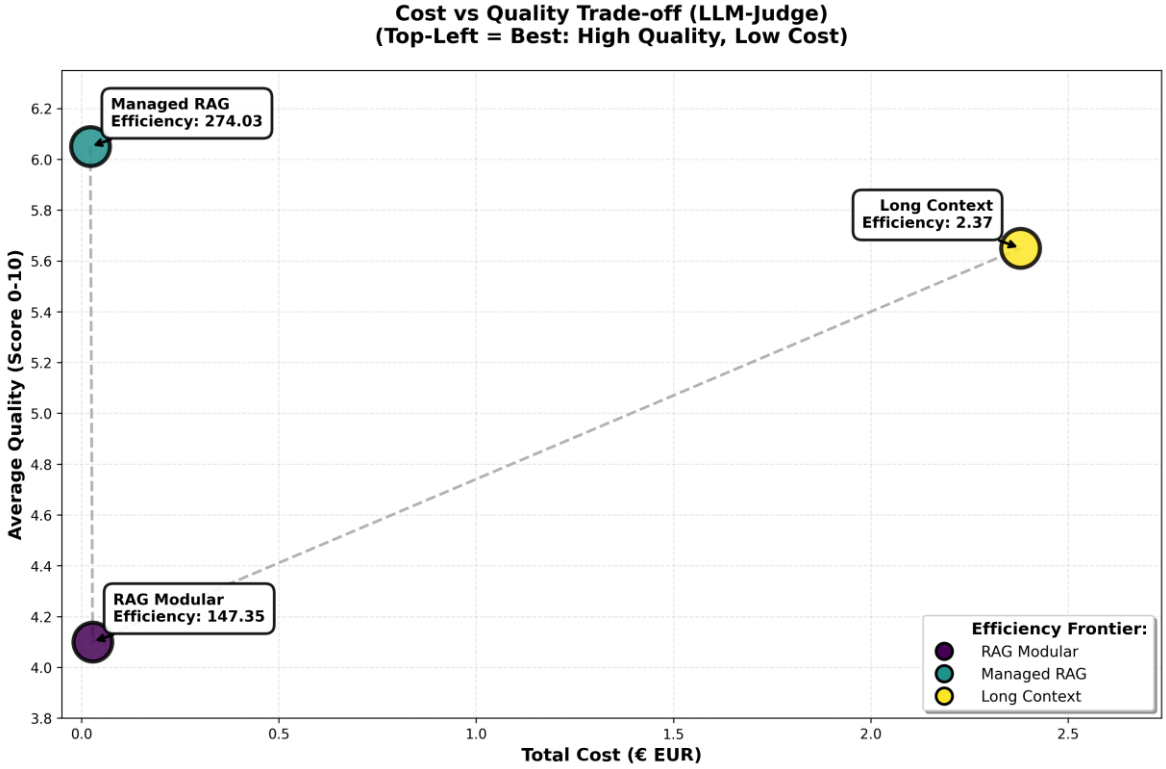

Figure 34. Scatter plot showing the trade-off relationship between total cost and average quality.

The analysis of Figure 34 reveals notable conclusions regarding the viability of the systems.

The managed RAG architecture dominates the comparison, positioning itself exactly at the ideal break-even point (upper left quadrant). By recording the lowest operating cost of the study (approximately 0.02 euros) and the highest average quality (6.05), its efficiency rate surpasses the other architectures by a wide margin. This solution demonstrates that delegating indexing to Google maximizes the return on investment.

The modular RAG architecture is positioned as the second most efficient design, as it is located in the lower left section, but with an average quality that provides it with low reliability.

The long context architecture is the worst option in terms of efficiency; this is due to the high costs of having to process hundreds of pages each time. Conversely, it is the second best in providing reliability, with 5.6 points.

The cost-performance analysis determines that the managed RAG architecture is the only financially optimal solution. The utilization of the provider's closed retrieval algorithms breaks the cost barrier, offering the best performance with minimal economic impact compared to the long context or modular RAG architectures.

8.2.4.2 Multi-variable analysis

To unify the four operational dimensions evaluated during the study (quality, economic efficiency, response speed, and context reliability), the results have been normalized on a scale from zero to ten. The projection of these metrics onto a spider web chart allows the strength or weakness of each design to be evaluated as a whole at a single glance.

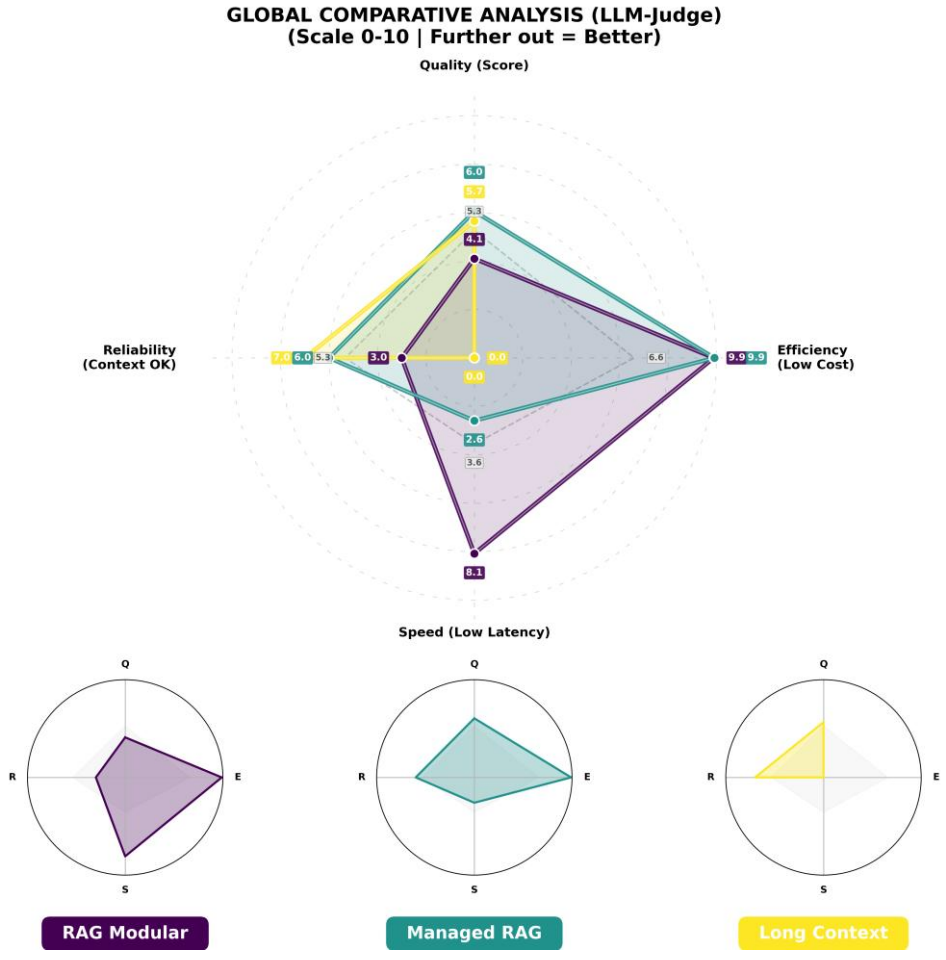

Figure 35. Global comparative analysis using a spider web chart on a normalized scale from 0 to 10.

The observation of the resulting shapes in Figure 35 serves as the final test of the study with 300 pages.

The most deficient profile corresponds to the long context architecture. Its visual representation exposes a very poor shape. Although it manages to maintain acceptable metrics in quality (5.7) and reliability (7.0) by having to send the complete document, it

obtains a value of 0.0 in both efficiency and speed. These poor performance and cost results invalidate its technical viability for deployment in production environments.

The modular RAG architecture shows unbalanced behavior. It has a near-perfect efficiency (9.9) and the highest speed in the comparison (8.1). However, this computational agility is achieved at the cost of a poor quality result (4.1) and reliability (3.0). In a technical sector where precision is fundamental, this architecture, configured with a single round and the capacity to retrieve ten fragments, is highly unviable for production environments.

The managed RAG shows the most balanced and solid structure in the comparison. This solution manages to match the maximum score in efficiency (9.9), leads in quality (6.0), and doubles the reliability with a 6.0 compared to the modular option. Its main technical disadvantage is concentrated on the speed axis with a 2.6. This penalty in latency is the direct result of the processing overhead applied by the provider's internal engine to ensure the retrieval of the correct fragments.

The global mapping of variables determines that the managed RAG architecture offers the most adequate technical compromise for this environment. It maximizes cost efficiency and guarantees the accuracy of technical responses, assuming a higher response latency as a structural tradeoff compared to modular developments.

8.2.5 Analysis conclusion

The comprehensive evaluation of the three architectures on the 300-page technical manual determines that the extensive processing of information is unviable for production environments. The long context approach, despite providing the model with total access to the document, introduces a computational load that severely penalizes latency and skyrockets costs. Furthermore, the manifestation of context fatigue in advanced queries demonstrates that injecting the complete documentation saturates the model's attention capacity, causing information loss and compromising its technical reliability.

For its part, the modular RAG design stands out for its computational agility, achieving the most competitive response times in the study. However, under a static single-cycle retrieval configuration, the vector database algorithm has deficiencies in capturing the correct information from a technical environment. The low hit rate and the drop in response quality limit its direct use, demonstrating that having speed without quality is equivalent to having nothing, since the quality of the responses is the most important factor in this field, even if the response time is slightly higher.

The analysis consolidates the managed RAG architecture as the most solid technological and financial solution. By delegating the indexing and search process to the provider's infrastructure, the system reaches the highest accuracy indices and guarantees a constant fail-safe mode in the face of erroneous queries. Although this design requires assuming a higher latency due to the internal processing of the retrieval algorithms, the system's maximum cost efficiency, reliability, and quality justify its selection as the reference architecture for operating with long technical manuals.

8.3 Analysis with different documents

To ensure that the conclusions of this project are reliable in a real-world environment, it was necessary to verify that the system's performance did not depend on a single type of file. The language model's attention capacity and the fragmentation algorithm's efficiency vary significantly according to the length and internal structure of the original document.

For this reason, an extended evaluation phase has been designed. The objective is to subject the three architectures (modular RAG, managed RAG, and long context) to a battery of tests using different technical manuals. In this process, not only is the total volume of pages modified, but also the density of the information, considering the concentration of text units, the layout of tables, diagrams, and numerical specifications.

To carry out this verification, two new documentary datasets have been incorporated that contrast with the original 300-page manual:

- The first is an electronic injection manual of 85 pages and approximately 36,000 text units. This file represents a scenario of moderate length but with high technical concentration, filled with diagnostic tables, electrical diagrams, and numerical values.
- The second is an automatic gearbox service manual of 142 pages and about 21,000 text units. This document represents a scenario with a higher number of pages but a lower actual text load, dominated by technical drawings of parts and specific text. It is very important to note that this latter file does not have an index, which may pose an additional difficulty for the long context architecture.

By evaluating these new documents under the same conditions and using the same automated judge, the goal is to measure, in isolation, the variables of economic cost, response time, and response quality. This approach will allow for the clear identification of the exact point at which artificial intelligence models that read long texts begin to suffer from attention fatigue, a common problem where the machine forgets or overlooks information located in the middle of the document. Likewise, it will be observed how external databases reduce the risk of the model fabricating responses based on its general memory when operating in technical environments.

8.3.1 Evaluation against structured data load

The first scenario of the expanded analysis is executed on an 85-page electronic injection manual. The particularity of this dataset lies in its technical density, as it is composed of diagnostic flows, pin connection schemes, and electrical value tables. This environment poses a challenge for the fragmentation algorithms used in retrieval-based architectures, since dividing the text carries the risk of sectioning continuous tables or diagrams. Due to its shorter length compared to the original manual, the file allows for assessing whether the long context architecture manages to analyze the information without suffering saturation.

Table 1. Consolidated results for the structured data load dataset.

Architecture	Average Latency (seconds)	Total Cost (euros)	Average Quality (LLM Judge)
Modular RAG	3.42	0.0269	8.30
Managed RAG	6.15	0.0178	7.60
Long Context	4.19	0.3698	9.30

The results obtained in Table 1 and subsequent visualizations expose a change in trend regarding the analysis of the long document. By reducing the information volume to 85 pages, the limitations of the architectures vary based on their capacity to manage dense and structured data.

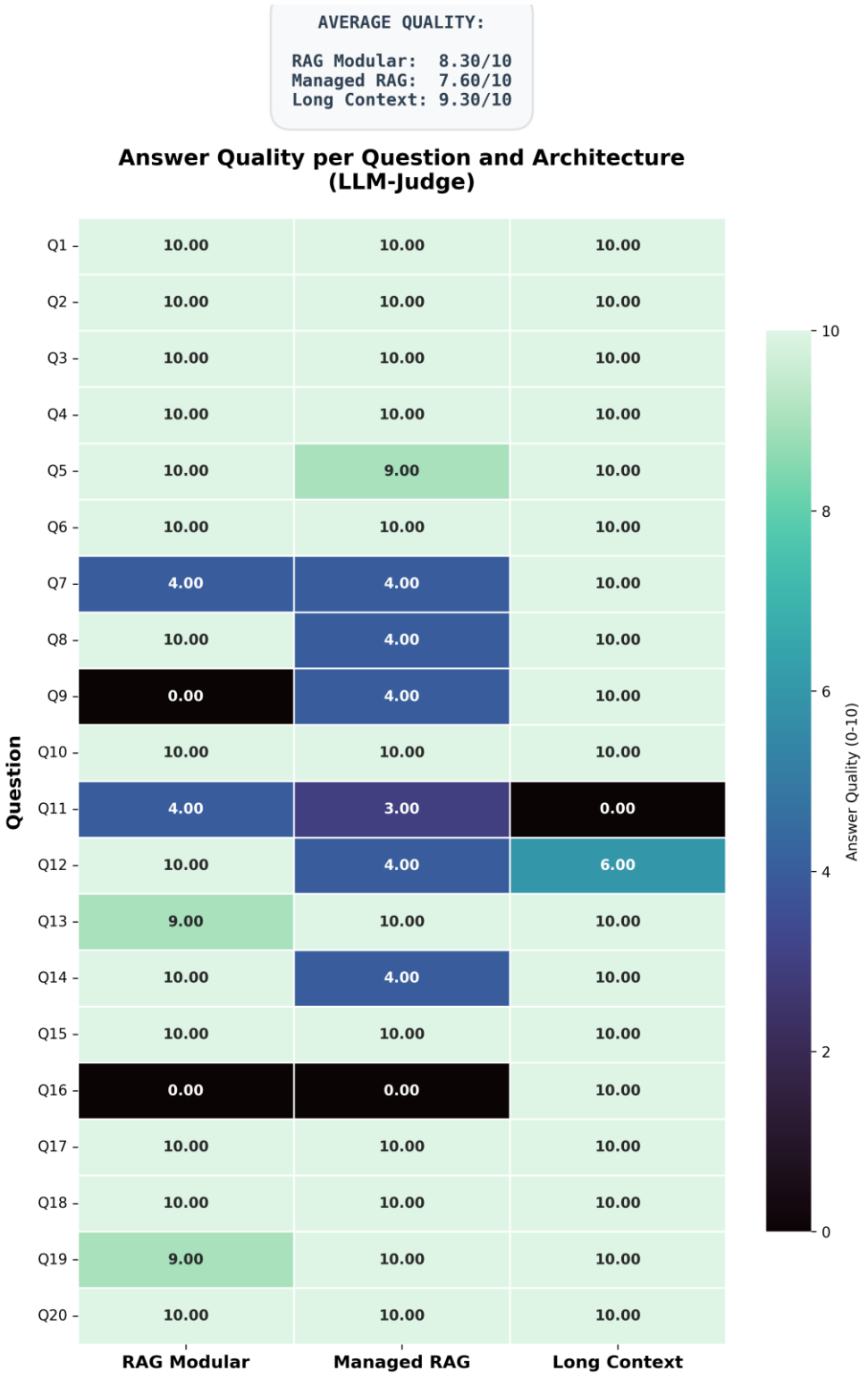

Figure 36. Heat map of response quality for the 85-page document.

As observed in Figure 36, the long context architecture is positioned as the most precise solution for this scenario, reaching an average quality of 9.30 out of 10. By processing the complete document without prior fragmentation, the model maintains the continuity of the electrical value tables and connection schemes. This eliminates errors derived from context loss, allowing the artificial intelligence to locate specific data with near-perfect accuracy in the majority of questions.

Conversely, retrieval-based approaches show technical degradation in the face of the manual's density. The modular RAG architecture obtains an 8.30, penalized by critical retrieval failures in questions Q9 and Q16, where text fragmentation prevented the model from accessing the complete information of the requested data. Managed RAG presents the lowest performance, 7.60, showing that its native indexing algorithms have difficulties correctly classifying data with many tables and complex diagnostic flows.

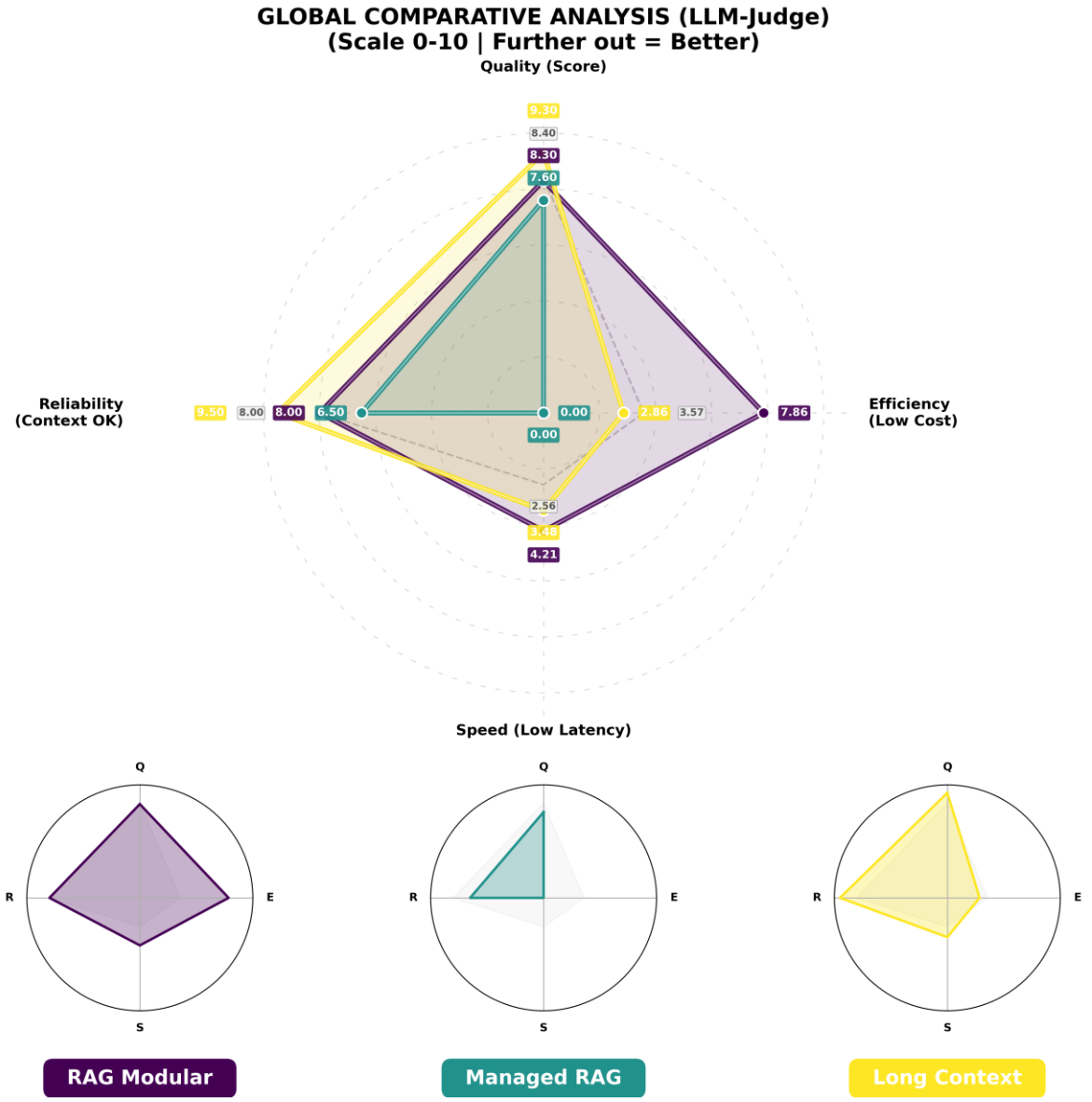

Figure 37. Global comparative analysis for the structured data load manual.

The multidimensional projection of Figure 37 allows for viewing the operational balance of each design. Despite its high quality, long context presents reduced cost efficiency (2.86) and higher latency than the managed option. With a total cost of 0.3698 euros for only 20 questions, this approach scales inefficiently even in moderate-sized documents.

The modular RAG architecture is consolidated as the most balanced option. It is the fastest architecture, with 3.42 seconds of average latency, and maintains a cost efficiency of 7.86. Although it sacrifices a margin of precision relative to long context due to data segmentation, its economic sustainability and speed position it as the most viable design for a production environment with high concurrency.

Finally, managed RAG has the most asymmetric and deficient profile in this scenario. Its representation in the spider web chart collapses on the efficiency and speed axes, obtaining null values of 0.00. The data confirm that, when facing manuals with high density of tables and schemes, the provider's internal processing engine adds an overhead that does not translate into an improvement in quality, invalidating its use for this type of document.

8.3.2 Evaluation with non-indexed document

The second scenario of the analysis employs a 142-page automatic transmission reconstruction manual. The particularity of this dataset lies in the disproportion between its physical length and the volume of actual information, added to the fact that the document lacks a structural index. Despite presenting 67% more pages than the structured data manual analyzed previously, the textual density is significantly lower, with approximately 21,000 text units.

This low density is because a large part of the file integrates graphic resources, exploded-view diagrams, and component lists with ample white space. This scenario allows for evaluating whether the degradation in the performance of the architectures is linked strictly to the number of pages or to the concentration of data. The consolidated results for this environment are detailed in Table 2.

Table 2. Consolidated results for the non-indexed document dataset.

Architecture	Average Latency (seconds)	Total Cost (euros)	Average Quality (LLM Judge)
Modular RAG	2.81	0.0309	9.75
Managed RAG	6.18	0.0287	8.40
Long Context	9.36	0.6358	9.60

The results in Figure 38 reflect that the success rate of each model depends as much on its technical design as on the way information is organized in the manual. In this 142-page scenario with low text density, the following is observed:

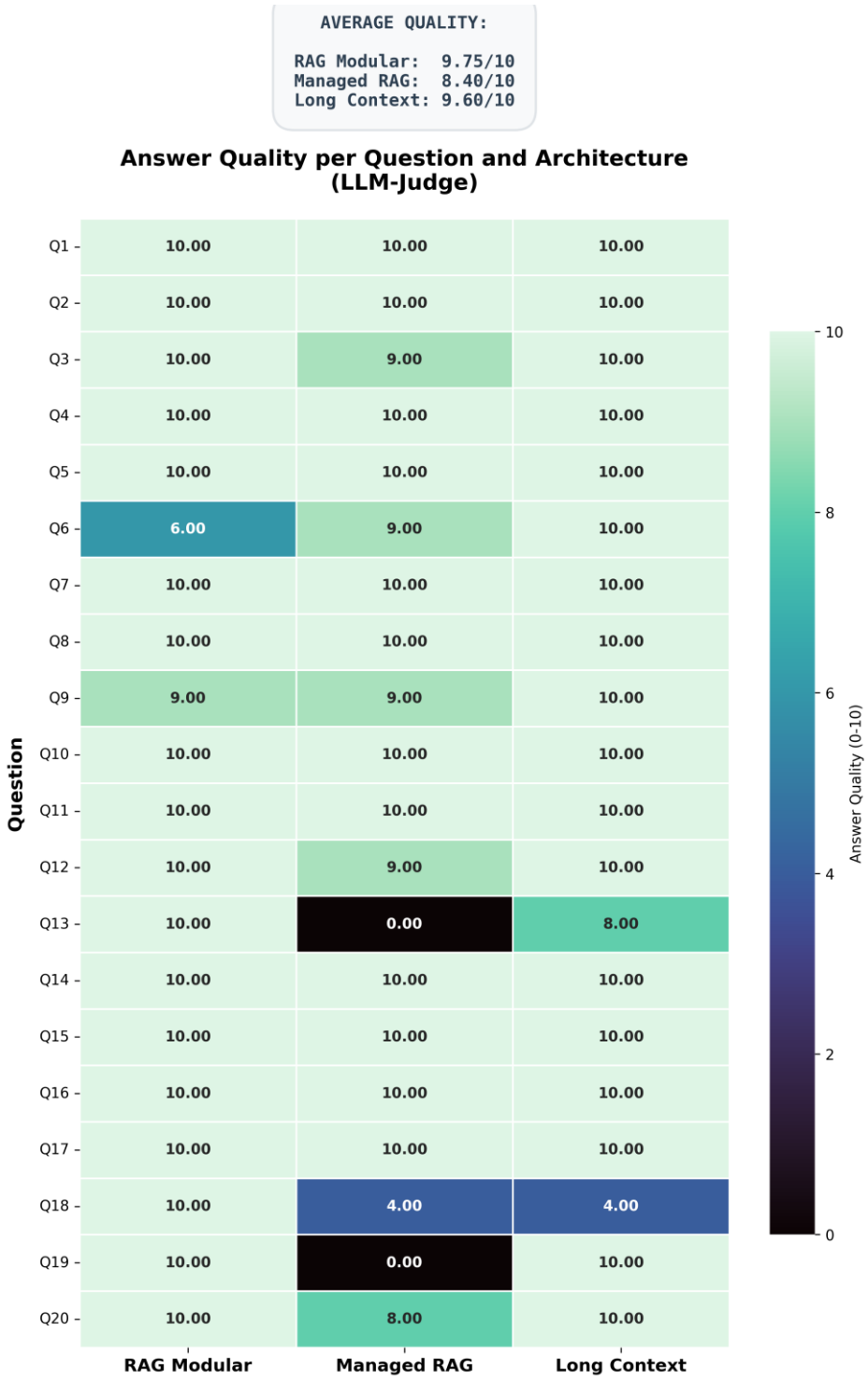

Figure 38. Heat map of response quality for the 142-page document.

The long context architecture maintains a high average quality of 9.60. Despite the physical length of 142 pages, the low textual density allows the model to process the information with ease in the majority of queries. A significant setback in reliability is identified in specific queries, such as Q18 obtaining a 4.00, and a slight degradation in Q13, obtaining an 8.00. These specific failures may draw attention and recall the problem that previous versions of Gemini 1.5 had, which suffered from the attention fatigue problem regarding information in the middle of the document. Although in this approach the intermediate information is not completely lost, we cannot say it suffers from this problem,

as Google stated that it was resolved in version 1.5 with 99% accuracy. I will detail this further later in point 8.3.3.

The modular RAG architecture is positioned as the most robust and precise design for this scenario, reaching an average score of 9.75. By employing a vector search engine that indexes only the relevant textual content, the system ignores the white spaces and graphic resources that disperse information in the original document. This capacity to "isolate" technical data allows for obtaining perfect scores in almost the entirety of the testbench, suffering only a punctual drop in query Q6.

Conversely, managed RAG presents the most irregular performance, with an average of 8.40. The absence of a structured index in the manual appears to severely penalize the provider's retrieval algorithms, causing critical retrieval failures, such as scores of 0.00 in queries Q13 and Q19. These results confirm that the managed solution is more sensitive to document disorder than the modular approach.

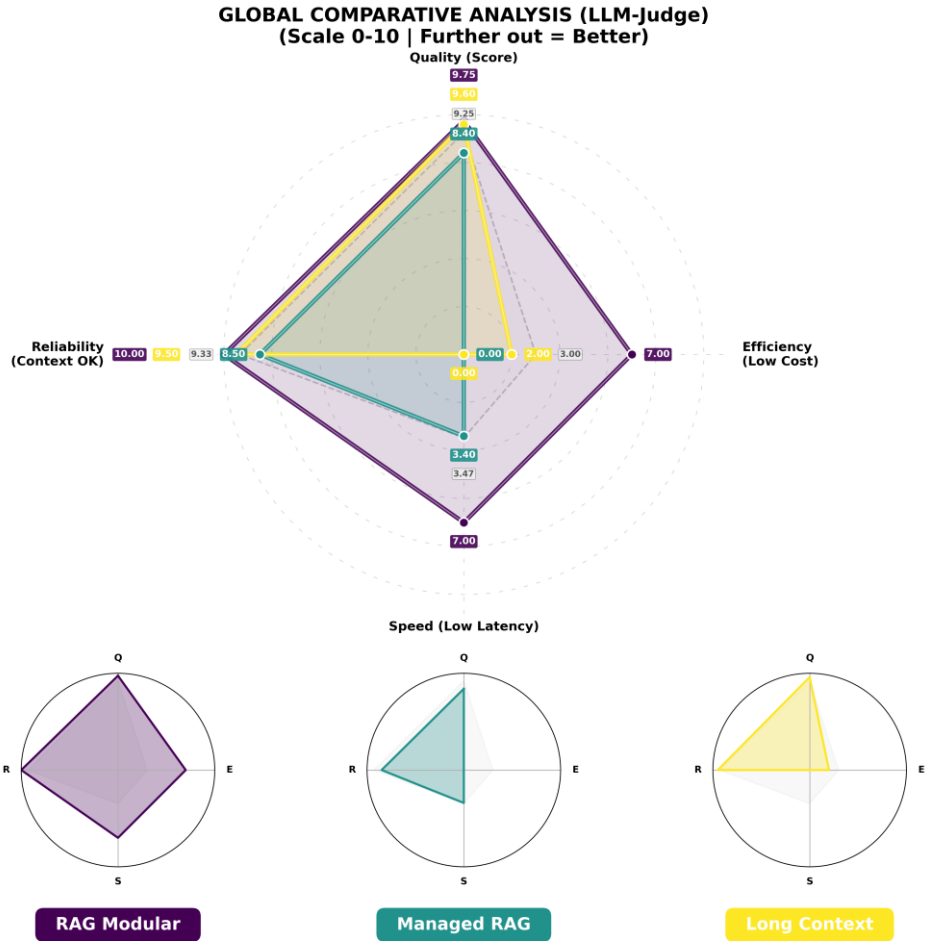

Figure 39. Comparative analysis using a spider web chart for the non-indexed document manual.

The multidimensional projection of Figure 39 confirms the operational superiority of the modular approach. This architecture draws the most balanced and efficient profile, leading the reliability and quality axes, while maintaining high competitiveness in speed and cost efficiency. The modular system guarantees a rapid response of 2.81 seconds and a cost of barely 0.03 euros by avoiding the redundant processing of pages without text.

The managed and long context architectures present asymmetric profiles with infrastructure shortcomings. The long context approach, despite its good technical quality, is operationally unviable by recording null speed and minimal economic efficiency, a direct

consequence of the overhead of processing the complete file in each interaction. The managed architecture shows similar degradation in performance, confirming that its internal engine adds a latency that does not translate into an improvement in documentary precision.

In conclusion, this scenario confirms that, to ensure maximum reliability in a production environment with documents of acceptable length but low density, modular RAG architecture is the most appropriate strategy. By searching only for relevant information, the system compensates for the poor format of the original manual. This improves the accuracy of the responses and makes the project more efficient.

8.3.3 Comparison of attention loss

The phenomenon of attention loss in long-context architectures has historically been one of the greatest challenges in natural language processing. This limitation is grounded in the test known as the "needle in a haystack" (Kamradt, 2023) [22] and in the study of advanced evaluation rules (Hsieh et al., 2024) [10]. Previous research confirmed that, given a high volume of data, the performance of the models dropped drastically when required to perform multidimensional reasoning that forced them to connect separate pieces of information, especially if these were located in the intermediate blocks of the document.

This behavior was formalized in the research titled "Lost in the middle" (Liu et al., 2023, Stanford University) [9]. This study demonstrated that language models suffer from a U-shaped positional bias. This means that the algorithm prioritizes and accurately recalls the information located at the beginning and at the end of the provided context, but systematically omits the central data. Under this theoretical paradigm, dividing information into small fragments using vector databases was considered a mandatory step to guarantee technical reliability.

However, the results obtained in this project through execution on AWS Lambda with the Gemini 3.0 Flash Preview model demonstrate a technical overcoming of this obstacle. Unlike previous generation models, the evaluation of the automotive manuals did not reflect the dreaded "U-shaped" curve. Although slight degradations were detected in very specific queries, the long-context model managed to maintain almost perfect scores when fully processing documents of up to 142 pages, locating diagnostic flows situated at the core of the text. These results confirm, in a practical environment, the claims of Google DeepMind (2024) [11], demonstrating that starting from the Gemini 1.5 version and later, this problem was resolved by 99%.

Despite this technical victory of the foundational models, analyses regarding the design of systems in production, led by experts such as Jerry Liu (2025) [23], warn that architecture design should not be based solely on raw reading capacity. The data extracted from this project corroborate this view and shift the decision-making paradigm: the primary problem of sending the complete document is no longer memory loss, but rather economic unviability. Tests demonstrate that the use of modular Retrieval-Augmented Generation architectures remains the definitive solution. By extracting only the relevant text, the RAG system reduces the cost and response time compared to full-reading approaches. In conclusion, the conversion of texts into mathematical data for indexing is no longer justified as a "patch" for the poor memory of artificial intelligence, but as an essential piece of engineering to optimize cost and latency in high-demand real-world environments.

8.3.4 Conclusion of the analysis

The evaluation with manuals of varying density and structure confirms that the operational viability of an architecture depends on the document's format and not solely on the number of pages.

Although the long context architecture processes the entire document without losing information, its high processing cost and sluggishness render it unviable in production. Likewise, it is demonstrated that disorganized or unindexed manuals penalize the internal systems of managed RAG. Conversely, the modular RAG architecture is consolidated as the best option in these scenarios: its external database isolates the relevant text and discards the graphic content, overcoming the poor format of the original file to guarantee maximum precision at a sustainable cost.

8.4 Final conclusion of the evaluation

The migration of the tests to AWS Lambda emerged as a technical decision after evaluating the initial results in n8n. Upon observing certain anomalies in response quality with long documents, it became necessary to verify whether these failures genuinely stemmed from the artificial intelligence or from the intermediary tool's format handling. Isolating the code and operating directly with the official APIs made it possible to clear up this uncertainty, guaranteeing the maximum reliability of the final data.

The exhaustive analysis of the metrics reveals the following strengths and weaknesses:

- Long context: It offers maximum analytical precision and proves to have mitigated information loss in intermediate blocks. However, its extremely high computational cost, elevated latency, and tendency toward saturation with large documents render it unviable for continuous use.
- Modular RAG: It is consolidated as the fastest, most efficient, and most secure solution for daily work with moderately sized manuals. It reduces costs, guarantees a fail-safe mode against hallucinations, and allows for total parametric control. Nevertheless, tests demonstrate that it suffers a severe degradation in reliability upon exceeding the 142-page threshold with a single-cycle configuration. Expanding the search iterations (multiple rounds) would resolve this deficit, but at the cost of sacrificing its main advantage: low cost.
- Managed RAG: It represents the most balanced alternative. Although it requires ceding control to the provider, it is the most agile design to implement and the only one capable of maintaining constant reliability and accuracy regardless of the document's length (85, 142, or 300 pages). Despite recording a higher latency, it guarantees very good results and a fail-safe mode, making it the best option for very large documents.

The primary conclusion of this evaluation is that there is no perfect system. From this structural limitation arises the need to design the intelligent router proposed in the implementation phase. This coordinator analyzes the intent of the query in real-time and routes it to the optimal architecture, thereby achieving a definitive system that ensures high technical reliability and a sustainable operational cost.

9 Conclusions

This research was proposed with the objective of designing, preparing, and evaluating in detail various generative artificial intelligence architectures oriented toward querying technical documentation, specifically within the automotive mechanics sector. Following the analysis of the variables of economic cost, response time, quality, and reliability, the following conclusions are detailed.

9.1 Technical conclusions

- Following the exhaustive evaluation in a simulated production environment, the main conclusion of the project determines that no single architecture optimally resolves all operational scenarios. The conclusions are:
- Data fragmentation remains essential: Despite the high precision of the long context, delegating the entire analysis to full reading is economically unviable and generates unacceptable latencies in production. Furthermore, volumes exceeding 300 pages still cause greater saturation. Therefore, the use of vector databases is mandatory to guarantee the viability and scalability of the system.
- The critical impact of the orchestration layer: It was demonstrated that low-code platforms such as n8n can alter text extraction in PDFs, introducing noise that simulates false memory failures in the AI (Lost in the middle). The migration to a serverless environment (AWS Lambda) was a vital step to eliminate intermediaries and reveal the true analytical capacity of the model.
- Hybridization as the definitive solution: The obtained results support recent research [12][13]: applied artificial intelligence requires composite systems. The development of the intelligent router materializes this conclusion. By analyzing user intent in milliseconds and routing the query to the ideal architecture, the definitive balance is achieved: the agility and economy of modular RAG for daily use, backed by the accuracy of the long context for complex diagnoses.

9.2 Contribution to personal development

Completing this Final Degree Project has been the definitive step in my training as an engineer. More than writing code, this project has forced me to confront the reality of the profession and to develop skills that go far beyond programming.

I have verified firsthand that in real engineering there is no perfect solution, but rather a constant balance. I have realized that the true work of a systems designer is knowing how to compromise. Accepting a slight loss of performance in exchange for reducing costs and response time.

I have also changed my perspective on integration work. I have proven to myself that I do not need to be an expert artificial intelligence researcher to build a solution for the industry. Handling the n8n tool, connecting the OpenAI and Google interfaces, and managing mathematical databases has given me a very broad technological perspective. The great challenge of these months has not been programming from scratch, but rather having clear concepts to know how to design the structure, connect the correct pieces, and set up three technological options ready to operate in the real world.

On an academic level, this project has served to give meaning to everything I have studied over these years. I have understood that the university does not seek to create code-writing machines, but professionals who know how to justify the reasoning behind every decision. During the development, I have had to merge my knowledge of distributed systems, architecture design, and automated quality control testing.

To conclude, I highlight my evolution when it comes to measuring results. I have left behind the habit of manually reading and verifying texts to move towards creating automated evaluation systems. Learning to use a language model as a judge has taught me to evaluate the precision of artificial intelligence mathematically, obtaining consistent data without relying on human error.

9.3 Future work

The great speed at which the artificial intelligence environment evolves opens various pathways to continue and expand the foundations of this research. As main lines of future work, the following development points are proposed.

Integration of knowledge graphs (GraphRAG): which are data structures that connect related concepts as if it were a conceptual map. This would give rise to advanced search systems based on information relationships. While the mathematical search system evaluated in this project has demonstrated high precision for very specific queries, such as the search for a specific numerical datum in the manual, graph-based systems would allow for general diagnostic queries. This would help the assistant understand complex links between multiple parts of the vehicle, for example, relating a failure in the fuel injection system with the readings from the exhaust pipe sensors.

Deployment of local infrastructure: It would be interesting to test data autonomy through the use of mathematical databases hosted on internal servers (ChromaDB or LanceDB), instead of using cloud services from external providers. To this end, testing processes could be implemented using specialized frameworks in artificial intelligence coordination. The objective of this test would be to analyze whether having total control over combined search algorithms, which use both traditional keywords and mathematical conversions, justifies the effort of maintaining the company's IT infrastructure versus the convenience of using third-party managed solutions, as has been done in this project.

Testbench and continuous auditing of new technologies: The rapid changes in this sector are demonstrated by the fact that, after finishing the testing phase of this project with a 3.0 Flash artificial intelligence model, the provider has launched a superior version on the market. As future work, it would be appropriate to repeat the automated testbench designed in this study using these new versions. This would allow evaluating whether software updates succeed in reducing the excessive response time of closed integrated systems and decreasing the high operating costs involved in sending complete documents to the artificial intelligence, as detailed in this document.

Fine-tuning of an LLM in a specific domain: As a vision oriented towards medium-term research and development, the creation of language models specialized in a specific field and their application in the industry is proposed. This would consist of the specific design and training of an artificial intelligence focused exclusively on the automotive sector. Just as the technological sector has developed medicine-oriented models capable of passing complex medical exams, there is room to create specialized support systems in applied mechanical engineering. The long-term objective would be to limit the machine's information to guarantee high precision within its technical specialty. Once control over this design in automobiles is consolidated, the system would have the capacity to adapt to other transport platforms, such as motorcycles, industrial vehicles, trains, or the aviation sector. This would allow deploying an intelligent system capable of reading and analyzing large quantities of technical pages to return precise mechanical indications, pointing exactly to which part to manipulate and under what adjustment measures, thus facilitating technical work in demanding professional environments.

10 Resources used

10.1 Bibliography

[1] **Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin**, I. *Attention Is All You Need*. Advances in Neural Information Processing Systems (NeurIPS), 2017, Vol. 30, pp. 5998-6008. Available at: https://arxiv.org/abs/1706.03762 [Accessed: 17-02-2026].

[2] **Reimers, N., & Gurevych, I. Sentence-BERT***: Sentence Embeddings using Siamese BERT-Networks*. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2019, pp. 3982-3992. DOI: 10.18653/v1/D19-1410 [Accessed: 17-02-2026].

[3] **Malkov, Y. A., & Yashunin, D. A**. *Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2018, Vol. 42, Number 4, pp. 824-836. DOI: 10.1109/TPAMI.2018.2889473 [Accessed: 17-02-2026].

[4] **Clark, E., August, T., Serrano, S., Haduong, N., Gururangan, S., & Smith, N. A.** *All That's 'Human' Is Not Gold: Evaluating Human Evaluation of Generated Text*. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL), 2021. Available at: https://aclanthology.org/2021.acl-long.565 [Accessed: 20-02-2026].

[5] **Zheng, L., Chiang, W. L., Ying, Y., Hao, S., Hwang, Y., ... & Xing, E. P.** *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena NeurIPS Conference* (Advances in Neural Information Processing Systems), 2023. Available at: https://arxiv.org/abs/2306.05685 [Accessed: 20-02-2026].

[6] **Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela**, D. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. ArXiv repository, 2020. Available at: https://arxiv.org/abs/2005.11401 [Accessed: 20-02-2026].

[7] **Liu, H., Zaharia, M., & Abbeel, P.** *Ring Attention with Blockwise Transformers for Near-Infinite Context.* ArXiv repository, 2023. Available at: https://arxiv.org/abs/2310.01889 [Accessed: 22-02-2026].

[8] **Rudin, C**. *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*. ArXiv repository, 2018. Available at: https://arxiv.org/abs/ 1811.10154 [Accessed: 22-02-2026].

[9] **Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2023).** *Lost in the Middle: How Language Models Use Long Contexts*. Stanford University. arXiv preprint arXiv:2307.03172. Available at: https://arxiv.org/abs/2307.03172 [Accessed: 05-03-2026].

[10] **Hsieh, C.-Y., Sun, S., Kim, S., et al. (2024).** *RULER: What's the Real Context Size of Your Long-Context Language Models?* arXiv preprint arXiv:2404.06654. Available at: https://arxiv.org/abs/2404.06654 [Accessed: 05-03-2026].

[11] **Google DeepMind. (2024).** *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context.* Technical Report. Available at: https://storage.googleapis.com/deepmind-media/gemini/gemini_v1_5_report.pdf [Accessed: 05-03-2026].

[12] **Kuan Li, Liwen Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Shuai Wang, Minhao Cheng**. (2025). *LaRA: Benchmarking Retrieval-Augmented Generation and Long-Context LLMs -- No Silver Bullet for LC or RAG Routing*. arXiv preprint arXiv:2502.09977. Available at: https://arxiv.org/abs/2502.09977 [Accessed: 07-04-2026].

[13] **Bowen Jin, Jinsung Yoon, Jiawei Han, Sercan Ö. Arık**. (2023/2024). *Long-Context LLMs Meet RAG: Overcoming Challenges for Long Inputs in RAG*. Available at: https://arxiv.org/html/2410.05983v1 [Accessed: 07-04-2026].

10.2 Web references

[14] **Bustamante, N.** *The RAG Obituary: Killed by Agents, Buried by Context Windows*.

Nicolas Bustamante Blog. Available at: https://www.nicolasbustamante.com/p/the-rag-obituary-killed-by-agents [Accessed: 10-02-2026].

[15] **Google. Gemini 1.5 Pro**: *Our next-generation model. Google The Keyword Blog (2024)*. Available at: https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024 [Accessed: 20-02-2026].

[16] **Pinecone Systems. Pinecone**: *The vector database to build knowledgeable AI.* Pinecone Official Documentation. Available at: https://docs.pinecone.io [Accessed: 20-02-2026].

[17] **OpenAI**. *What are tokens and how to count them?* OpenAI Help Center. Available at: https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them [Accessed: 20-02-2026]

[18] **AWS. What is RAG?** *Explanation of retrieval-augmented generation.*. Amazon Web Services. Available at: https://aws.amazon.com/es/what-is/retrieval-augmented-generation [Accessed: 20-02-2026].

[19] **Google for Developers**. *File Search: Retrieval Augmented Generation (RAG).* Google Gemini API Documentation. Available at: https://ai.google.dev/gemini-api/docs/file-search [Accessed: 22-02-2026].

[20] **Google DeepMind.** *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. Technical Report, 2024. Available at: https://storage.googleapis.com/deepmind-media/gemini/gemini_v1_5_report.pdf [Accessed: 20-02-2026].

[21] **NVIDIA Corporation.** *NVIDIA H100 Tensor Core GPU: Unprecedented performance for large-scale AI.* NVIDIA Data Center. Available at: https://www.nvidia.com/es-es/data-center/h100 [Accessed: 22-02-2026].

[22] **Kamradt, G. (2023).** *Needle In A Haystack - Pressure Testing LLMs*. Official GitHub repository with open-source tests. Available at: https://github.com/gkamradt/LLMTest_NeedleInAHaystack [Accessed: 05-03-2026].

[23] **Liu, J. (2025).** *LlamaParse One Year Later: The Complete Document Automation Platform*. LlamaIndex Official Blog. Available at: https://www.llamaindex.ai/blog/llamacloud-one-year-later-the-complete-document-automation-platform [Accessed: 05-03-2026].

11 Annexes

This section includes all the technical documentation necessary to understand, review, and replicate the project step by step.

When working with a visual automation tool, the source code is not composed of classic programming files. Instead, the system saves everything in structured text documents. These files contain the exact configuration, position, and connections of each block within the process.

11.1 User manual

To demonstrate that the results are real and to allow anyone to verify them, I explain below the exact steps to start the project on a personal computer. In this way, any member of the evaluation committee can set up the working environment on their local machine and import all the source code without complications.

Phase 1: Installation of n8n via command line

Native deployment through the Node.js package manager (npm) has been chosen for its lightness and agility.

Prerequisite: Ensure that Node.js is installed on the operating system.

1. Open the command console (Terminal).
2. Execute the following command to install the tool globally:
   "npm install -g n8n"
3. Once the installation is finished, start the local server by simply executing:
   "n8n"

Phase 2: Configuration of the local environment

Without closing the terminal window, open the web browser and type the address of the local server: http://localhost:5678.

Upon loading the page, the home screen requests the creation of an administrator account. This step is free, and the registration data is saved exclusively on the computer itself. As soon as this form is completed, one enters directly into the main panel of the n8n tool.

Phase 3: Download and importation of the source code

The source code of this project is located in a public GitHub repository. To execute the architectures, the following steps must be followed:

- Access the project's GitHub repository and download the files with the .json extension.
- In the n8n panel, click on "Add Workflow".
- Open the workflow options menu (top right corner) and select "Import from File...".
- Select the downloaded .json file.

Phase 4: Structure of the files and use cases

Several files are provided in the repository, among which the following two main files stand out, each designed for a specific auditing purpose:

File 1: Master Router - Sistema Multi-Arquitectura.json

This file contains the complete orchestrated workflow, designed to be executed and tested in a local environment. It allows interactively observing how the conditional router and the entire data lifecycle function. However, for its correct execution on the evaluator's machine, the following technical considerations must be taken into account:

- Strict requirement of the mathematical text conversion model (Embeddings): To guarantee compatibility and precision in the external database, this workflow strictly requires the use of the specific model from the artificial intelligence provider company OpenAI: the text-embedding-3-large model. It is not compatible with free versions from other providers, so the user must configure a private access key with an active balance for the transformation of text to numerical data to function correctly.
- Memory limit in file uploading: Due to the technical restrictions configured by default in local n8n installations regarding the handling of binary files, attempting to upload the complete 65 MB manual will cause a size excess error. The simplest operational solution for this testing environment is to divide the PDF document into two or more lighter fractions, processing them separately.
- Timeout periods in Architecture 1: Another limitation that could arise when testing the modular architecture is that the workflow fails due to timeout issues between the local n8n, OpenAI, and Pinecone. If the local internet network is unstable or the computer does not have sufficient memory for reading the document, the system may interrupt the task. In that case, it is recommended to process shorter texts to verify the logical operation without saturating the equipment.

File 2: TFG_Benchmark_chatmotor.json

This file contains the three architectures (Pinecone RAG, managed RAG, and long context) separated and prepared for validation tests. The evaluation committee can interact with them:

- In batch testbench mode, configuring the spreadsheet reading node to automatically introduce a list of questions, imitating the experiment documented in this project.

Finally, there is a very important point regarding credentials. For security reasons to avoid exposing secrets in public repositories, the exported files do not include private access keys to external services. For the processes to function after their importation, the evaluating user must access the component settings and provide their own private keys with an active balance. Specifically, it will be indispensable to configure the accesses for the mathematical conversion model, the search engine in the external database, and the text generation model that drafts the final response (having an active account in the Google and OpenAI nodes).

11.2 Source code repositories

The program code for the automated judge and the graphical analysis, as well as the automation tool files developed for this project, are properly stored and documented on the GitHub development platform. To guarantee the total transparency of the testbench and allow for technical review by the evaluation committee, the program design has been divided by modules into the following storage spaces:

n8n workflow repository - [Public Access]

It contains the configuration files that define the structure of the workflows described in this document, which are mentioned in the installation manual of the previous section. Furthermore, this space includes the programmed automatic backup mechanisms to guarantee the recovery of the main routing system in the event of potential computing failures.

AWS Lambda repository - [Public Access]

Below is all the Python code developed for the evaluation in the Amazon cloud. Here is the implementation that interacts directly with the Gemini and Pinecone APIs, without intermediaries. In addition, the prompts used in n8n were reused so that there are no differences, and everything has been executed with the LLM Judge evaluator. The code to generate all the comparative graphs among the three architectures is also uploaded.

User interface repository - [Private Access]

It contains the code of the interactive webpage developed with the Next.js framework and hosted independently on a proprietary virtual private server. Given that this code is integrated within the technological infrastructure of my personal portfolio and contains critical configuration variables and private server passwords, it is kept closed to the public for strict computer security reasons. Nevertheless, the resulting webpage is fully functional and is accessible for its evaluation through this web link, an internet address provided to the evaluation committee.

11.3 Data tables of the test benches

To maintain fluent reading in the results chapter, the obtained performance metrics were presented through visual graphs. In this section of the annex, the exact real data and the test set used during the automated evaluation are organized into tables, thereby guaranteeing the transparent tracking of the calculations for economic efficiency, quality, and wait times.

11.3.1 Complete manual test battery

The following details the set of twenty technical questions introduced into the system to force the reasoning of the different design options on the complete automotive manual.

Table 3. Breakdown of the technical test battery, classifying the queries by their level of complexity.

ID	Query sent in each architecture	Evaluation objective
P01	What is the gap setting for the spark plugs on all models?	Direct data extraction (Exact value)
P02	Where is the OBD diagnostic socket located inside the car?	Spatial localization

P03	What is the opening temperature of the thermostat for the M43TU engine versus the M54 engine?	Reasoning: Data comparison
P04	Describe the procedure to reset the Service Interval Indicator using the instrument cluster buttons.	Procedure extraction (Step by step)
P05	What is the minimum brake pad lining thickness allowed before replacement?	Direct data extraction (Limit value)
P06	How do you check the fluid level in the manual transmission?	Maintenance procedure extraction
P07	What is the tightening torque AND angle for the cylinder head bolts on the N42 engine? (Stage 1, 2, and 3)	Complex extraction (Multiple values/phases)
P08	Identify the correct firing order for the 6-cylinder engines.	Logical sequence extraction
P09	What specific Automatic Transmission Fluid (ATF) must be used for a transmission with a GREEN label/tag?	Conditional search
P10	On the rear suspension, what is the torque for the lower arm to the subframe, and what is the crucial condition for tightening?	Data extraction + Safety condition
P11	What is the valve clearance adjustment specification for the intake valves on the M54 engine?	Precision data extraction
P12	Compare the bleeding procedure of the cooling system for the 4-cylinder vs 6-cylinder engines. Does the 6-cylinder require extra steps?	Reasoning: Procedure comparison
P13	How many millimeters of "pre-load" must be applied to the driveshaft center bearing (propeller shaft) during installation?	Direct data extraction (Tolerance)
P14	Does this manual cover the repair of the High-Pressure Fuel Pump (HPFP) for the diesel 320d model?	Documentary limit (Scope verification)
P15	According to the wiring diagrams, what is the color of the wire connected to Pin 1 of the Ignition Coil for Cylinder 1?	Diagram reading and interpretation
P16	Locate the relay for the Secondary Air Pump. Where is it specifically located and what fuse number protects its circuit?	Multiple cross extraction
P17	Explain the removal of the starter motor on 6-cylinder models. Why is the top bolt considered difficult and what tool approach is suggested?	Complex procedure + Tool usage
P18	For the N42 engine Valvetronic system, what happens if you install the magnetic wheel on the eccentric shaft with a standard steel screw instead of the anti-magnetic one?	Reasoning: Technical cause-effect
P19	I have a rattling noise from the front of the engine on an M54. Based on the "Fault Finding" section and common	Reasoning: Diagnosis (Troubleshooting)

	issues described, what is the most likely Vanos-related cause?	
P20	Can the cylinder head be skimmed (machined) on the N42 engine? If so, what is the maximum material removal limit?	Trap / Parametric hallucination evaluation

11.3.2 Structured data load test battery

The set of twenty technical questions specifically used to evaluate the manual with structured data load and tables is detailed below.

Table 4. Breakdown of the structured data load test battery.

ID	Query sent in each architecture	Evaluation objective
P01	What is the tightening torque for the knock sensor according to the manual?	Easy-level direct numerical data extraction
P02	What is the required electrode gap for the spark plug?	Easy-level exact numerical data extraction
P03	According to the manual, what are the specific cut off pressures for the low and high A/C pressure switches, and the cut-in pressure for the medium switch?	Normal-level multiple numerical values extraction
P04	Which specific Diagnostic Trouble Code (DTC) corresponds to "Too low voltage in signal circuit of knock sensor"?	Normal-level diagnostic code identification
P05	When checking the resistance of the coolant temperature sensor at 20°C with a multimeter, what should the rated resistance be?	Normal-level direct data extraction under a specific condition
P06	At what temperature range does the conversion efficiency of the three-way catalytic converter reach its maximum?	Normal-level value range extraction
P07	In the failure diagnosis for the Electronic Throttle (DTC P0122), if you pull out the joint and the voltage between the 3 and 5 pins is around 12V and 5V between 6 and 2, and then you check the resistance between 1, 4 and 6 pins and find it to be 1.0kΩ, what is the exact follow-up step you must perform?	Hard-level deductive reasoning and technical procedure tracking
P08	During an inspection for a Front Oxygen Sensor failure, the engine warms up and runs at idle speed. You pull off the connector and measure the voltage between pin 3 (+) and 4 (-). It reads 0.5V. According to the troubleshooting table, what is the exact next step?	Hard-level step-by-step troubleshooting
P09	If an engine has periodic unsteadiness at idle speed, and doing a spark out cylinder test shows NO fall or	Hard-level decision making in a diagnostic tree

	fluctuation in engine revolution, what is the immediate next diagnostic step you should take?	
P10	For the electronic accelerator pedal position sensor, what is the relationship between signal 1 and the signal voltage, and what is the exact voltage of signal 1 at the idle speed position?	Hard-level technical relationship comprehension and data extraction
P11	I am diagnosing DTC P0134. I've put the ignition switch to ON, pulled the harness, and measured 12V between pin 1(+) and 2(-). I checked the resistance between 1 and 2 and got 35 ohms at 23°C. The heating circuit is normal. I checked for short circuits on pin 2 and pin 1 and found none. What is the next physical action I must do according to the table?	Very hard-level complex analysis of a diagnostic scenario
P12	A customer complains about warm starting difficulty. Following the steps, I shorted pins 30 and 87 of the fuel pump relay, but the fuel pressure only reached 300 kPa (which is not around 400kPa). What is the exact follow-up step I should jump to?	Very hard-level deductive reasoning with limit values
P13	You are replacing an EKP13 series electric fuel pump. The manual states that the electromotor revolutions are adjusted during production to change the flow for different engine requests. How exactly is this adjustment made, and what warning is given regarding replacing pumps between different models?	Very hard-level manufacturing process comprehension and safety warning
P14	The engine does not rotate when starting. The battery voltage is 12V. While maintaining the ignition switch at the START position, the voltage on the terminal connected to the pull in winding of the starting motor is 6V (below 8V). According to the manual's table, what must be done next?	Very hard-level electrical fault interpretation and action taking
P15	An engine shows failure codes P2122 and P2123. The mechanic wants to disassemble the accelerator pedal position sensor to repair the internal circuit. Based strictly on the manual's notes for this component, is this allowed, and how should it be maintained?	Expert-level handling restrictions and regulations comprehension
P16	According to the "Safety Precautions for Diagnosis and Maintenance", what are the exact procedures and conditions required if you need to drag the engine by the starter to inspect cylinder pressure without actually starting the engine?	Expert-level conditional safety protocol extraction
P17	A mechanic replaces a failed electronic throttle body. According to the "Special attention" note in the troubleshooting section, what must absolutely be done after replacing the electronic throttle body to prevent unsteady idle speed?	Expert-level critical procedure identification after a repair

P18	You are troubleshooting an engine that extinguishes with A/C load. After confirming the A/C switch input on ECU pin 75 is normal, you check the A/C system pressure and clutch and they are normal. You then check voltages on ECU pins 64, 65, 66 and 67 and find them abnormal (Result: No). What is the exact follow up step?	Expert-level complex and sequential electrical troubleshooting
P19	I need to repair a damaged knock sensor on a Chery Arauca. Can you provide the detailed procedure from the manual to open the knock sensor housing and replace its internal piezoelectric crystal?	Documentary limit evaluation and data fabrication detection
P20	What is the standard tire pressure (in psi or bar) for the Chery Arauca 2012 model as indicated in this electronic fuel injection manual?	Manual scope evaluation and general knowledge leakage detection

11.3.3 Non-indexed document test battery

The set of twenty technical questions used to evaluate the non-indexed information document manual, which is characterized by having a lower text load and a greater quantity of diagrams and images, is detailed below.

Table 5. Breakdown of the non-indexed document test battery.

ID	Query sent in each architecture	Evaluation objective
P01	What does the abbreviation "ATF" stand for in the manual?	Easy-level acronym meaning extraction
P02	How many speeds does the A442F automatic transmission have?	Easy-level direct numerical data extraction
P03	What is the specified torque for the Center support set bolt?	Normal-level exact numerical data extraction
P04	Before assembling new discs for the First and Reverse Brake, what must you do with them?	Normal-level prior mechanical procedure extraction
P05	What is the required piston stroke measurement for the Overdrive Direct Clutch?	Normal-level direct measurement data extraction
P06	What is the required piston stroke measurement for the Rear Clutch?	Normal-level direct measurement data extraction
P07	When checking the operation of the one-way clutch on the overdrive gear unit, how should the input shaft turn?	Hard-level mechanical operation comprehension
P08	What is the exact compressed air pressure required when checking the piston stroke of both the overdrive direct clutch and the rear clutch?	Hard-level mechanical operation comprehension

P09	During the Parking Lock Pawl Assembly, what is the standard distance required between the transfer adaptor surface and the top of the bracket tab, and what specific tool is used?	Hard-level tolerance measurement and specific tool extraction
P10	What is the specified torque for installing the Parking Lock Pawl to the Transfer adaptor?	Hard-level tolerance measurement and specific tool extraction
P11	When checking the correct installation of the overdrive planetary gear, overdrive direct clutch, and one-way clutch assembly, what specific SST is used on the installation surface of the oil pump and what tool is used to measure the distance?	Very hard-level multiple special tools identification
P12	If the lining of a disc is peeling off or discolored in the First and Reverse Brake, or if a part of the printed numbers are defaced, what is the exact action required according to the manual?	Very hard-level visual troubleshooting and action taking
P13	To remove the throttle cable during component parts removal, what specific tool is required and how exactly is the cable removed?	Very hard-level disassembly procedure extraction and tool usage
P14	For the A442F automatic transmission, what specific engine is mentioned where shift response is greatly improved by communication between the ECM and TCM to momentarily reduce engine output when shifting?	Very hard-level complex electronic interactions comprehension
P15	When comparing the specified torque for tightening the "Valve body × Transmission case" versus the "Oil pan × Transmission case", what are the respective exact N-m values for each?	Expert-level multiple numerical values extraction and comparison
P16	Exactly how many apply gaskets must be removed when removing the O/D case and center support?	Expert-level specific quantitative data extraction
P17	According to the manual's abbreviations section, what do B0, B2, and C0 stand for respectively?	Expert-level specific quantitative data extraction
P18	What is the specific SST number required to place a dial indicator onto the overdrive brake piston?	Expert-level exact tool technical reference search
P19	What is the recommended tire rotation interval and tire pressure for a vehicle equipped with the A442F transmission?	Manual scope evaluation and general knowledge leakage detection
P20	Can you provide the specific paint color codes for the transmission casing as listed in the A442F manual?	Documentary limit evaluation and data fabrication detection