



University of
Zurich^{UZH}

Design and Implementation of an AI-based Agent to Inform Best Practices on Test Case Execution Routines

Zihan Liu
Zurich, Switzerland
Student ID: 22-736-938

Supervisor: Chao Feng, Jan von der Assen, Siddhant Gupta,
Gongpei Cui

Date of Submission: June 29, 2025

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, 29/06/2025

A handwritten signature in black ink, appearing to read 'Fikary', is written above a horizontal line.

Signature of student

Abstract

The rapid advancement of Large Language Models (LLMs) has introduced new opportunities in software and hardware development, particularly in assisting engineers with test case creation and automation. This thesis explores the application of LLMs in the automotive testing domain, focusing on the development and deployment of an AI-based assistant for test case generation. By leveraging fine-tuning and Retrieval-Augmented Generation (RAG), the proposed solution aims to enhance both the efficiency and semantic accuracy of test development workflows.

The study begins with a comprehensive analysis of existing LLM architectures and adaptation methods to identify the most effective strategies for domain-specific tasks. Based on these insights, an AI-driven module is designed and implemented to support the requirements of automotive testing at Volvo. The system features a structured data processing pipeline and employs vector databases to encode semantic representations of test artifacts, enabling context-aware retrieval and generation.

To evaluate the proposed approach, this work investigates several fine-tuning techniques for embedding models using a curated dataset derived from real-world requirements and test sequences. A comparative analysis of state-of-the-art open-source models demonstrates that embedding performance is highly sensitive to model selection, fine-tuning strategy, and data quality. These findings contribute to the emerging field of AI-assisted engineering by offering practical guidance on the deployment of LLMs in complex industrial environments.

Abstrakt

Die rasante Entwicklung großer Sprachmodelle (Large Language Models, LLMs) eröffnet neue Möglichkeiten in der Software- und Hardwareentwicklung, insbesondere zur Unterstützung von Ingenieur:innen bei der Erstellung und Automatisierung von Testfällen. Diese Masterarbeit untersucht den Einsatz von LLMs im automobilen Testumfeld mit Fokus auf die Entwicklung und Implementierung eines KI-basierten Assistenten zur Generierung von Testfällen. Durch die Kombination von Fine-Tuning-Methoden und Retrieval-Augmented Generation (RAG) zielt die vorgeschlagene Lösung darauf ab, sowohl die Effizienz als auch die semantische Genauigkeit innerhalb von Testentwicklungsprozessen zu verbessern.

Die Arbeit beginnt mit einer umfassenden Analyse bestehender LLM-Architekturen und Anpassungsstrategien, um geeignete Methoden für domänenspezifische Aufgaben zu identifizieren. Aufbauend auf diesen Erkenntnissen wird ein KI-gestütztes System entworfen und implementiert, das auf die Anforderungen von automobilen Testprozessen bei Volvo zugeschnitten ist. Das System integriert eine strukturierte Datenakquisitionspipeline und verwendet Vektordatenbanken zur semantischen Repräsentation von Testartefakten, wodurch eine kontextbezogene Generierung und Wiederverwendung ermöglicht wird.

Zur Bewertung des Ansatzes werden verschiedene Fine-Tuning-Strategien für Einbettungsmodelle anhand eines kuratierten Datensatzes aus realen Anforderungen und Testsequenzen untersucht. Ein Vergleich aktueller Open-Source-Modelle zeigt, dass die Leistung semantischer Einbettungen stark von der Modellwahl, der gewählten Anpassungsmethode und der Qualität der Trainingsdaten abhängt. Die Ergebnisse leisten einen Beitrag zum wachsenden Forschungsfeld der KI-gestützten Ingenieurwissenschaften, indem sie praxisrelevante Erkenntnisse zur Nutzung von LLMs in komplexen industriellen Umgebungen liefern.

Acknowledgments

I would like to express my sincere thanks to Chao Feng for his continuous guidance through regular bi-weekly meetings, which provided essential technical insights. I am also grateful to Siddhant Gupta and Gongpei Cui at Volvo Cars for offering the opportunity to work on this project and for providing valuable resources and support. My appreciation extends to all colleagues at Volvo for their helpful feedback and suggestions throughout the process. Finally, I would like to thank my supervising professor, Prof. Dr. Burkhard Stiller, for enabling this research and overseeing its academic direction.

Note: I have used ChatGPT (OpenAI) solely as a writing assistant for language refinement and clarity improvement. All scientific content, analysis, and conclusions are my own work.

Contents

Declaration of Independence	i
Abstract	iii
Abstrakt	v
Acknowledgments	vii
1 Introduction	1
1.1 Motivation	2
1.2 Description of Work	2
1.3 Thesis Outline	3
2 Background	5
2.1 Domain Context: Automotive and Software Testing	6
2.1.1 Software Testing	6
2.1.2 HIL Testing	6
2.1.3 CAN, CAPL, and Test Infrastructure	8
2.2 LLM Technologies and Adaptation Strategies	8
2.2.1 LLMs	8
2.2.2 RAG	9
2.2.3 Fine-tuning	11
2.2.4 Fine-tuning vs RAG	12
2.2.5 Humiliation and Failure Modes	13

2.2.6	MCP and Tool Integration	14
2.2.7	Private Data Integration	15
3	Related Work	17
3.1	Literature Review Methodology	17
3.2	Fine-Tuning for Domain Adaptation	18
3.2.1	Full-Parameter Fine-Tuning	19
3.2.2	Training Data Construction in Industrial Contexts	20
3.3	RAG in Engineering Contexts	21
3.3.1	Database Construction for Retrieval	21
3.3.2	Embedding Model Utilization	22
3.3.3	Fine-Tuning Data Preparation for Embeddings	23
3.3.4	Combining Documents with Large Language Models	24
3.4	Comparison Between Fine-Tuning and RAG Strategies	25
3.4.1	Adaptation Mechanism	25
3.4.2	Data Requirements and Flexibility	25
3.4.3	Latency and Resource Implications	26
3.4.4	Maintenance and Interpretability	27
3.4.5	Summary of Trade-Offs	27
4	Architecture	29
4.1	Overall Design	29
4.2	Data Acquisition and Preprocessing	30
4.2.1	Data Collection and Curation for LLM-Driven Systems	30
4.2.2	Industrial Data Annotation and Preprocessing	31
4.2.3	Challenges in Structuring Requirements and Test Assets	31
4.2.4	Data Source Diversity and Stakeholder-Guided Selection	31
4.2.5	From Raw Extraction to Structured Knowledge Units	32
4.3	Embedding and Vector Indexing Layer	33

<i>CONTENTS</i>	xi
4.3.1 Model Selection and Fine-Tuning Strategy	33
4.3.2 Vector Store and Indexing Scheme	34
4.3.3 Periodic Re-Embedding and Pipeline Integration	37
4.4 RAG Agent	37
4.4.1 Context Construction from Retrieved Chunks	37
4.4.2 Prompt Template and Generation Interface	38
4.4.3 Tool Integration and External Invocation	38
4.4.4 Response Composition and Traceability	38
5 Implementation	41
5.1 Data Collation	41
5.2 Data Collation Pipeline	42
5.3 From Text to Vector Representations	42
5.4 Embedding Model Fine-Tuning	43
5.4.1 Candidate Model Evaluation	44
5.4.2 Triplet Construction and Contrastive Learning	44
5.4.3 Deployment and Embedding Inference	46
5.5 Agent Tool Integration.	47
6 Evaluation	49
6.1 Evaluation Setup	49
6.2 Triplet Construction and Subset-Based Model Screening	51
6.3 Embedding Model Comparison	52
6.3.1 Initial Evaluation on Benchmark Subset	52
6.3.2 Full Evaluation with Parameter Scaling Analysis	53
6.4 Effect of Domain-Specific Fine-Tuning	53
6.5 Impact of Negative Samples in Fine-Tuning	56
6.6 Interaction Between Embedding Quality and LLM Behavior	58
6.6.1 Motivation and Setup	58

6.6.2	Results and Observations	58
6.6.3	Interpretation and Design Implications	58
6.7	User Feedback and A/B Evaluation of RAG-Enhanced Agent	59
6.7.1	Sentiment Analysis Methodology	61
6.8	Discussion	63
6.8.1	Performance Interpretation Across Embedding Models	63
6.8.2	Retrieval Quality vs. Generative Attribution	64
6.8.3	Label Construction and Negative Sampling Effects	65
6.8.4	Performance, Scale, and Deployment Trade-offs	66
6.8.5	Cross-Component Generalization and Use Case Reflection	67
7	Summary and Conclusions	69
7.1	Summary	69
7.2	Limitations and Future Work	70
7.3	Conclusion	71
	Bibliography	73
	List of Figures	83
	List of Tables	83
A	User Feedback Summary	87
	User Feedback Summary	87

Chapter 1

Introduction

Large Language Models (LLMs) have rapidly advanced in recent years and are beginning to shape how software testing is performed across industries. In particular, the automotive domain which is characterized by safety-critical systems and increasingly complex validation requirements which has unique challenges that traditional testing methodologies struggle to address. While manual testing provides domain expertise and flexibility, it is inherently time-consuming and error-prone. Automation offers improvements in speed and repeatability, but it still demands significant human oversight and suffers from limitations in adaptability and knowledge management.

This thesis explores the application of LLMs to support automotive test development by designing a specialized assistant capable of both test case generation and interactive dialogue. Although general-purpose LLMs such as GPT-4 have demonstrated strong performance across a wide range of natural language processing tasks, their effectiveness within the automotive domain remains limited. Initial experiments conducted during this study indicate that these models often produce inaccurate or irrelevant outputs when applied to tasks that require domain-specific precision.

The primary challenge arises from the technical complexity of automotive testing. Many procedures rely on the use of Controller Area Network (CAN) signals, which are structured messages exchanged among electronic control units within a vehicle. These signals contain diagnostic and control information and require detailed understanding of message identifiers, signal encoding, and timing specifications. In addition, test cases are often implemented in a specialized scripting language known as Communication Access Programming Language (CAPL), which is commonly used for simulating, monitoring, and controlling communication in vehicle networks. Automotive testing is also frequently conducted in HIL environments. These setups allow engineers to validate embedded control software by connecting it to real-time simulations of physical vehicle components, without the need for an actual car.

Publicly available models and datasets typically do not cover such domain-specific elements. Relevant artifacts, including CAN signal definitions, mappings between requirements and test cases, and diagnostic scripts, are usually stored in internal company tools or

customized development environments. As a result, the direct use of pre-trained general-purpose models is insufficient for real-world deployment in this context. This motivates the development of a tailored approach that integrates proprietary engineering knowledge while addressing concerns related to scalability, interpretability, and deployment feasibility.

Given this gap, the present work proposes the development of a domain-adapted LLM-based agent for testing generation, enhanced through fine-tuning and Retrieval Augmented Generation(RAG). The research also includes a structured approach to data acquisition and curation, tailored to the needs of internal system integration at industry. This effort establishes the foundation for a sustainable data pipeline that supports both ongoing fine-tuning and future research initiatives within the organization.

1.1 Motivation

In the course of development and validation at the company, testing workflows initially relied on manual execution, which, although flexible, consumed significant resources and remained susceptible to human error. The adoption of automated testing improved repeatability but introduced new challenges, such as the need for extensive scripting, manual dataset preparation, and maintenance of testing infrastructure.

These limitations motivated the integration of AI, particularly LLMs, into the automotive testing workflow. The objective was not only to automate routine processes using AI but also to enable intelligent interaction with test engineers, supporting both repetitive tasks and exploratory testing activities.

However, experimentation with general-purpose LLMs like GPT-4 revealed insufficient domain-specific knowledge for specialized testing tasks. Queries involving CAN signals, diagnostic routines, or CAPL-based execution sequences often returned generic or inaccurate responses. An extensive review of resources such as Hugging Face, GitHub, and academic publications yielded no suitable pretrained models.

In addition, the scarcity of structured and annotated automotive test datasets—especially those related to CAN communication, diagnostics, and in-vehicle architectures—presents a major obstacle to fine-tuning efforts. These challenges underscore the need for a dedicated automotive-specific LLM and a reusable open dataset to facilitate progress in this field.

1.2 Description of Work

To address the aforementioned challenges, this thesis presents the design, implementation, and evaluation of an AI-based assistant tailored for interactive support in automotive testing environments. Rather than focusing solely on automatic test case generation, the assistant incorporates RAG techniques to provide engineers with contextualized access to requirement and test sequence information. The contributions of this work are summarized as follows:

- An in-depth literature review was conducted on LLM adaptation techniques, including supervised fine-tuning, unsupervised pretraining, and RAG, with a focus on their applicability to the automotive testing domain.
- A modular, AI-powered architecture was designed, integrating fine-tuned LLMs with a vector store backend to enable semantic test case retrieval and CAN signal metadata lookup via structured tool interfaces.
- A hybrid dataset was constructed by combining publicly available programming examples—sourced from platforms such as GitHub and Stack Overflow—with internally extracted testing specifications, CAPL code snippets, and CAN signal definitions. All data sources were thoroughly filtered, cleaned, and reformatted to ensure compatibility with LLM input formats. The dataset creation and definition methodology was systematically validated to support downstream tasks.
- A RAG mechanism was introduced into the automotive testing workflow from the ground up. This includes strategies for automotive-domain data collection, a long-term architecture for sustainable dataset evolution, and a reusable agent-tool interaction framework supporting future extensions.
- Common causes of hallucination in LLM-based agents within this domain were identified and countermeasures investigated, including embedding model enhancement and query reformulation strategies.
- An empirical evaluation was performed using multiple state-of-the-art embedding models on real test queries, comparing scoring metrics such as cosine similarity, token overlap, and embedding distance quality in the context of test case retrieval.
- The performance impact of scaling the LLM decoder component was investigated while keeping the embedding model fixed. This analysis assessed whether increased model capacity yields measurable gains in response quality within a RAG pipeline under domain-specific constraints.
- A method was developed to synthetically augment missing training data in scenarios where ground-truth annotations were unavailable. This approach enabled the generation of valid training triples (**anchor**, **positive**, **negative**) by prompting the LLM to infer plausible pairs based on domain knowledge and semantic similarity, thereby supporting robust embedding training despite incomplete datasets.

1.3 Thesis Outline

The remainder of this thesis is organized as follows:

- **Chapter 2 – Background** introduces the foundational concepts behind LLMs, fine-tuning, RAG, and their relevance to automotive testing.

- **Chapter 3 – Related Work** reviews academic and industrial efforts in AI-assisted test generation, domain-specific LLMs, and open-source tooling in the automotive context.
- **Chapter 4 – Architecture** presents the high-level system design of the proposed solution, outlining key components and their interactions in a technology-agnostic manner.
- **Chapter 5 – Design and Implementation** details the practical aspects of the system, including data collection, preprocessing, model training, and vector search pipeline construction.
- **Chapter 6 – Evaluation** reports on experiments conducted using real HIL configurations, including metrics such as retrieval accuracy and user satisfaction.
- **Chapter 7 – Summary and Conclusions** summarizes the research contributions, reflects on limitations, and discusses opportunities for future work such as dataset expansion, multilingual support, and cross-OEM adaptation.

Chapter 2

Background

The development of intelligent assistants for automotive testing requires a deep understanding of both domain-specific engineering practices and the capabilities of modern AI systems. This chapter outlines essential background concepts by reviewing key topics in software and hardware testing, presenting core technologies such as LLMs, and discussing their integration within industrial workflows.

It begins with an overview of software testing principles, with particular focus on automation and model-based approaches. The chapter then introduces HIL testing, a widely adopted methodology in automotive systems that enables real-time simulation of dynamic behavior with physical ECUs. Supporting technologies such as CAN and CAPL are also discussed, as they form the foundation of many vehicle-level test environments.

Building on this, the chapter surveys recent advances in LLMs and their application to engineering tasks. Two main strategies for domain adaptation are introduced: RAG, which enables dynamic access to external knowledge sources, and fine-tuning, which modifies model weights using proprietary data. Their roles are analyzed in the context of evolving, safety-critical testing environments.

Beyond adaptation techniques, the chapter also addresses practical system-level considerations, including hallucination risks, tool orchestration via MCP, and secure integration of private data. These aspects are critical for deploying LLM-based systems that maintain traceability, explainability, and robustness under real-world constraints.

By synthesizing insights from testing theory, ML infrastructure, and modern software architecture, this chapter provides a comprehensive foundation for understanding how LLMs can be effectively applied in automated test development scenarios.

2.1 Domain Context: Automotive and Software Testing

2.1.1 Software Testing

Software testing is a critical component of the software development lifecycle, aimed at evaluating the functionality, correctness, and robustness of a software system under development. It ensures that the software behaves as intended under various conditions and meets specified requirements before release. Testing helps identify bugs, performance bottlenecks, security vulnerabilities, and usability issues, thereby improving the quality and reliability of software systems [1], [2].

In modern software engineering, testing is often categorized into several levels, including unit testing, integration testing, system testing, and acceptance testing [3]. Each level addresses distinct concerns, from verifying small components in isolation to validating system-level behavior against business requirements.

In addition to these testing levels, testing methodologies are commonly divided into manual and automated approaches. While manual testing provides flexibility and context-awareness, it is labor-intensive and prone to human error. Automated testing, on the other hand, leverages scripts and frameworks to increase speed, consistency, and coverage [4].

However, designing effective tests remains a non-trivial challenge. As Whittaker notes, testing is difficult because it often lacks a formal oracle and must reason about an infinite number of program behaviors under varying inputs and system states [5].

In recent years, model-based testing (MBT) has gained traction, enabling test case generation from formal specifications or system models. MBT can reduce manual effort and increase traceability, especially in large systems [6].

Furthermore, artificial intelligence (AI) and machine learning (ML) techniques are increasingly being incorporated into testing processes. AI can assist in automated test case generation, bug detection, and test suite optimization [7]. This trend is especially relevant in the context of rapidly evolving software systems and limited test resources.

Search-based software testing (SBST), a subfield of search-based software engineering, applies evolutionary algorithms and other heuristics to optimize test cases against various objectives, such as coverage, fault detection, or performance thresholds [8].

Overall, software testing continues to evolve from manual and heuristic practices toward more automated, formalized, and AI-supported workflows.

2.1.2 HIL Testing

HIL testing refers to a test methodology in which actual hardware is part of the testing process, primarily utilized in the automotive industry. HIL test benches are indispensable for

evaluating various controllers in modern vehicles, ensuring they function correctly alongside sensors before being deployed in real vehicles. On an HIL test bench, it is possible to simulate a wide range of conditions—such as steering wheel position, on-screen operations, wheel speeds (active, passive, or intelligent), and different sensor states—thereby enabling comprehensive validation of both controllers and sensors in a controlled environment [9].

Typically, conducting on-site inspections of physical vehicle behavior is both costly and time-intensive, and only a limited set of scenarios can be authorized for testing. In contrast, simulated testing of vehicle control using installed sensors and a comprehensive environment provides a more practical and cost-effective alternative [10]. HIL systems allow developers to conduct fault injection, regression testing, and boundary testing under repeatable conditions, improving test coverage and system reliability [11].

A standard HIL setup consists of a real-time simulation platform (often powered by tools such as dSPACE or NI VeriStand), the device under test (DUT), signal conditioning hardware, and a software interface for test orchestration [12]. These platforms ensure deterministic execution and closed-loop feedback between the virtual environment and physical components, which is especially crucial in embedded automotive systems with strict real-time requirements

The CAN bus was developed by Germany’s Bosch in the 1980s as a serial data communication protocol designed to handle data exchange among numerous control and testing devices in modern automobiles. Widely adopted in electric vehicle control systems, CAN is supported by CANoe which is a practical and powerful system design and analysis tool developed by Germany’s Vector. Through Vector’s CAN bus interface hardware, CANoe provides a link between the virtual and physical bus environments [13].

By using CANoe, it is possible to realize completely virtual, all-digital simulations of bus applications, as well as semi-physical simulations that combine physical and virtual nodes. Additionally, it enables real-time monitoring of communication on a physical bus. CANoe supports the entire lifecycle of bus development—from initial design and simulation to final testing and analysis—offering a seamless integration of network design, simulation, and testing [14].

In recent years, HIL testing has become especially critical for automotive developing, where the complexity of environment perception, decision-making, and actuation layers demands scalable, high-fidelity validation frameworks [15]. HIL can be combined with sensor simulation (LiDAR, radar, camera emulation etc.) and traffic scenario engines to provide realistic, interactive environments for safety-critical components.

Moreover, standardized test frameworks such as AUTOSAR are often validated through HIL systems to ensure compliance and integration across vendors [16]. Overall, HIL is not only a bridge between virtual and physical validation but also an enabler of early-stage system integration and continuous delivery in model-based development.

2.1.3 CAN, CAPL, and Test Infrastructure

Modern automotive validation workflows are tightly coupled with network communication protocols, scripting environments, and test orchestration platforms. Among these, CAN [17], CAPL [18], and specialized test infrastructure play foundational roles in enabling robust, scalable, and reproducible test procedures.

CAN serves as the primary communication backbone in embedded automotive systems, facilitating message exchange between ECUs. Each signal transmitted over CAN is identified by a unique message ID and is associated with timing, encoding, and value interpretation rules that vary by OEM and platform. Effective validation requires not only decoding these messages, but also understanding their semantic relationship to physical phenomena and functional requirements.

CAPL is widely used within testing environments as the de facto scripting language for simulating and monitoring CAN behavior. Its event-driven syntax allows engineers to specify precise signal interactions, define complex temporal dependencies, and inject faults in a controlled manner. CAPL scripts are often executed within tooling ecosystems such as CANoe or CANalyzer, which support both purely virtual and mixed virtual-physical test configurations [19], [20].

Beyond messaging and scripting, test execution is typically orchestrated using integrated platforms that combine hardware signal conditioning, simulation control, and result capture. These systems interface with DUTs, manage HIL configurations, and enable regression testing across multiple variants and vehicle configurations. Metadata such as test purpose, feature tags, and signal mappings are often embedded in non-standard formats, making downstream automation and retrieval challenging.

The tight coupling between CAN protocols, CAPL scripts, and heterogeneous test infrastructure presents significant barriers to AI integration. Data extraction, normalization, and alignment are non-trivial, especially when artifacts are stored across multiple disconnected tools. This motivates the development of LLM-based systems that can bridge structured signal-level knowledge with unstructured documentation and test context [21], [22].

2.2 LLM Technologies and Adaptation Strategies

2.2.1 LLMs

LLMs are deep learning models trained on massive corpora of text to learn the statistical structure of natural language and generate human-like responses [23]. With the rise of transformer-based architectures such as BERT [24] and GPT [25], LLMs have become the cornerstone of modern natural language processing (NLP), excelling in tasks like text generation, summarization, classification, and question answering [26].

At every stage of modern software and hardware development, engineers encounter documentation complexity and knowledge barriers, particularly when working in proprietary environments where web-based resources are scarce. Internal company data is often fragmented, domain-specific, and poorly structured, which severely hampers efficient knowledge discovery and onboarding for new developers. In such cases, LLMs can function as knowledge aggregators and intelligent assistants, capable of consolidating domain knowledge and providing natural language interfaces to complex datasets [27].

Trained on diverse datasets including books, websites, and code repositories, LLMs develop a general-purpose capability to understand and generate language. This allows end users to interact with systems using natural language rather than formal queries or scripts [28]. In enterprise settings, fine-tuned LLMs can ingest proprietary documentation, codebases, and test specifications to support engineers in locating information, generating test scripts, or suggesting improvements [29].

Unlike traditional rule-based retrieval systems, LLMs offer reasoning ability, flexible language comprehension, and the capacity to generalize across unseen queries. This makes them especially powerful in dynamic domains like automotive testing, where information may be implicit, fragmented, or partially defined.

However, LLMs also come with limitations. One major challenge is the risk of hallucination—generating plausible but incorrect information—which poses risks in safety-critical environments like automotive engineering [30]. Furthermore, large context sizes, memory bottlenecks, and lack of domain-specific knowledge in off-the-shelf models limit their out-of-the-box usability [31]. Addressing these limitations often requires fine-tuning on curated internal data or combining LLMs with structured retrieval systems (e.g., via RAG) [32].

Open-source models such as LLaMA [33] and Falcon have lowered the barrier to customization, making it feasible for companies to deploy and specialize LLMs within secure, on-premises environments. Combined with frameworks for tool use and memory management, LLMs are evolving into AI agents capable of executing workflows, interacting with APIs, and adapting to user feedback.

In conclusion, LLMs represent a transformative technology for intelligent information access in industrial domains. When carefully aligned with organizational knowledge and guardrails, they can serve as efficient, conversational front-ends to deeply technical environments.

2.2.2 RAG

In this thesis, RAG is not merely studied as a theoretical concept, but is implemented as a central mechanism in the proposed HIL-GPT agent. Specifically, the retriever is optimized to handle structured test documents, while the generator is integrated into the CI pipeline for interactive scenario construction. This hybrid design addresses the limitations of standalone fine-tuning, particularly in dynamic and knowledge-intensive domains like vehicle testing.

RAG is a hybrid framework that combines the strengths of information retrieval and natural language generation, offering a powerful solution to augment LLMs with external, domain-specific knowledge [32]. This approach is particularly beneficial in specialized environments like automotive testing, where the pre-trained knowledge of LLMs may be insufficient or outdated.

Unlike traditional closed-book LLMs, which rely solely on internal parameters to generate responses, RAG-based systems incorporate a retrieval component to fetch relevant documents or text chunks from a knowledge corpus (e.g., vector databases), and then use these to guide the answer generation process [34]. This dramatically enhances the model’s factual accuracy, grounding ability, and domain adaptability.

The process of RAG typically includes two steps:

1. **Retrieval:** Given a query x , retrieve the top- k most relevant documents $\{z_1, z_2, \dots, z_k\}$ from a corpus \mathcal{D} using a similarity function (typically cosine similarity on dense embeddings).
2. **Generation:** Condition the response y on both x and the retrieved context z using an autoregressive model.

The overall conditional probability can be formalized as:

$$P(y | x) = \sum_{z \in \mathcal{D}} P(y | x, z)P(z | x) \quad (2.1)$$

This reflects the marginalization over all retrieved documents, combining the generation probability with the retrieval relevance [32].

RAG offers several advantages for industrial applications, particularly in domains like automotive testing. By decoupling retrieval and generation, RAG enables large language models to incorporate up-to-date or proprietary information without retraining, thereby facilitating domain adaptation and lowering overall deployment costs [32]. It enhances interpretability by allowing inspection of retrieved documents [35], and empirical studies show it effectively reduces hallucination by grounding responses in factual content [36]. However, applying RAG in automotive contexts presents unique challenges. Data granularity is a key issue, as artifacts like CAPL scripts and test specifications are often semi-structured and require customized chunking and schema alignment. Additionally, domain-specific elements such as CAN signal identifiers and bit-level fields may be poorly handled by standard tokenizers [37]. Embedding models also risk underperformance due to terminological drift, making domain-specific fine-tuning essential [38]. Finally, the retrieval stage introduces inference latency, which can be problematic in time-sensitive testing environments.

To address the aforementioned challenges, this work customizes the RAG pipeline for automotive testing scenarios. Key adaptations include a domain-specific chunking and embedding strategy tailored to test case documents, benchmarking of multiple embedding

models—such as BGE, Instructor, and MiniLM—based on their performance in signal recall tasks, and the extension of the retriever to incorporate metadata attributes like test purpose, configuration, and associated CAN signals. These enhancements collectively improve semantic matching accuracy and the factual consistency of generated responses. As such, RAG emerges as a compelling alternative to static fine-tuning and brittle prompt engineering, offering dynamic access to evolving technical knowledge while maintaining adaptability and interpretability.

2.2.3 Fine-tuning

With the rapid advancement of deep learning techniques, pre-trained language models (PLMs) have become the dominant paradigm in NLP. Fine-tuning plays a central role in this paradigm, enabling general-purpose language models to adapt to specific downstream tasks [26], [39]. This approach not only preserves the strong generalization capabilities acquired during pre-training but also significantly boosts task-specific performance [40].

The traditional NLP pipeline heavily relied on task-specific architectures and handcrafted features. The emergence of models such as BERT [39] and T5 [26] marked a shift toward a pre-training–fine-tuning framework. This framework typically involves two stages: (1) a large-scale pre-training phase on a general corpus to capture linguistic patterns, and (2) a task-specific fine-tuning phase on smaller, supervised datasets for tasks such as classification, question answering, or named entity recognition.

Fine-tuning methods can be broadly categorized as follows:

- **Full fine-tuning:** All model parameters are updated during the task-specific training phase. This method often yields high performance but may lead to overfitting on small datasets and requires substantial computational resources.
- **Feature-based tuning:** The pre-trained model is frozen and only the top layers like classification head are trained. This is efficient but might underperform if task-domain shift is significant.
- **Parameter-efficient fine-tuning (PEFT):** Recent approaches such as Adapter layers [41], LoRA [42], Prefix-Tuning [43], and BitFit [44] modify or train only a small subset of parameters. These methods are particularly effective for large models and low-resource scenarios [45].

Fine-tuning has proven effective in various domains, including multilingual NLP [46], code understanding and generation [47]–[49], and software engineering tasks like test case generation. These models demonstrate that transfer learning not only improves accuracy but also enhances robustness and uncertainty calibration [50].

As PLMs continue to grow, fine-tuning strategies are evolving toward more efficient, modular, and robust methods. Current research focuses on improving generalization, reducing overfitting, enabling zero-shot performance, and adapting models to new domains with minimal data [51].

2.2.4 Fine-tuning vs RAG

Fine-tuning and RAG are two complementary strategies that enhance the effectiveness of LLMs in downstream applications. In the context of software and automotive testing, both approaches offer unique advantages for addressing domain-specific challenges. This section compares the two methods across key dimensions such as knowledge integration, adaptability, efficiency, and use in the proposed HIL-GPT system.

Fine-tuning refers to the process of updating a pre-trained model's internal weights using task-specific labeled data [26], [39]. This method is especially effective when the task distribution is narrow and well-represented in the training data. In the context, fine-tuning was primarily used to optimize embedding models, such as BGE and InstructorXL, for accurately representing automotive test case data, including signal names, configurations, and expected outcomes. The strength of fine-tuning lies in its ability to internalize complex domain semantics and compress knowledge directly into model parameters. Once trained, these models are fast and robust at inference time [42], [43].

However, fine-tuning also comes with several limitations. It requires computational resources for training, significant labeled datasets, and frequent retraining when domain knowledge evolves [44]. Moreover, the model becomes a black box in which factual sources are hard to trace. These drawbacks are particularly pronounced in domains like vehicle testing, where specifications, signal definitions, and test cases are constantly being updated.

RAG offers a more flexible alternative. It augments a frozen language model with a retrieval component that fetches relevant documents or knowledge chunks at inference time [32]. This design allows the model to access up-to-date or proprietary information dynamically, without retraining. In the system, RAG is central to the HIL-GPT agent's ability to interact with a structured test case corpus. The retriever identifies relevant content based on semantic embeddings, and the generator conditions its output on both the user query and the retrieved context [34], [37]. This leads to grounded, interpretable responses with traceable evidence [35].

The primary advantage of RAG is its adaptability: knowledge updates require only corpus modification, not model re-training. This significantly lowers maintenance costs and facilitates real-time response in CI environments [32]. Furthermore, RAG improves interpretability by showing retrieved documents and helps reduce hallucination risks by grounding outputs on factual text [30], [36].

Nevertheless, RAG also introduces latency due to the retrieval step and depends heavily on the quality of the embedding model and chunking strategy. In the implementation, the project mitigates these challenges by designing a fine-tuned embedding pipeline, applying automotive-specific token normalization, and enriching metadata within the retrieval index.

In summary, fine-tuning and RAG serve different yet synergistic purposes. Fine-tuning enables deep internal adaptation to domain semantics, while RAG empowers dynamic knowledge access and transparency. By combining both in the system, fine-tuning for

precise embeddings, and RAG for flexible response generation, the project achieves a balance between performance, maintainability, and interpretability. This hybrid architecture is particularly suited for domains like HIL testing, where accuracy, explainability, and updatability are equally critical.

2.2.5 Humiliation and Failure Modes

Despite their remarkable fluency and task generalization, LLMs can generate outputs that are misleading, factually incorrect, inappropriate, or even offensive. In sensitive domains such as software testing, automotive systems, and HIL engineering, these failure cases pose serious risks to safety, usability, and trust. Although the term "humiliation" is not formally defined in machine learning literature, it can loosely describe situations where LLM behavior results in reputational damage, social discomfort, or decision errors due to incorrect or inappropriate content.

One major cause of humiliation-like behavior is hallucination, the phenomenon where the model generates content that is fluent but ungrounded in truth [30]. Hallucination can occur in both closed-book models and retrieval-augmented systems when context is sparse or ambiguous [52], [53]. For instance, an LLM may assert that a signal is valid in a test configuration when it is not defined in any actual documentation.

Another critical failure mode is confident misstatement [54], where the model expresses incorrect information with high certainty. This is particularly harmful in engineering workflows, where users may defer to the perceived authority of the system [55]. Such false confidence is tightly linked to poor uncertainty calibration [50].

Inappropriate tone, style, or bias may also result in humiliation effects. LLMs have been observed to generate disrespectful or offensive content [56], [57], or reinforce harmful stereotypes when prompted carelessly [58]. In collaborative industrial environments, such a tone can damage human trust and undermine the professional tone expected in tooling.

Further risks arise in multi-turn dialogue, where models may inconsistently contradict themselves [59], [60] or exhibit unsafe behavior under adversarial prompting [61]. These behaviors are amplified in long contexts or poorly scoped use cases.

Several strategies have emerged to mitigate such risks like RAG provides external factual anchors to constrain generations.

In the HIL-GPT deployment, the implementation addresses humiliation risks through a layered defense: fine-tuned embedding models, automotive domain-specific retrieval, output ranking, and structured test case validation. Moreover, responses are filtered using rule-based verifiers to avoid potential hallucination or ambiguous signal mappings.

As LLMs move closer to high-stakes decision domains, avoiding humiliation-type failure modes becomes not only a technical challenge, but also a broader alignment and ethical concern [62], [63]. Responsible deployment requires not only better models, but also better human oversight and tool integration.

2.2.6 MCP and Tool Integration

As LLMs become increasingly integrated into real-world applications, a major limitation arises from their isolated nature: they lack the ability to dynamically interact with external tools, real-time data, or structured databases. To overcome this constraint, Anthropic proposed the MCP in early 2024, an open standard that defines how LLMs can communicate with external systems in a secure, interpretable, and modular fashion [64].

MCP provides a unified interface and message format that allows LLMs to make structured calls to external servers, tools, and APIs during inference. This effectively enables the model to “augment” its context with information retrieved or computed externally, such as querying a signal database, executing code, or interacting with local file systems. Unlike traditional prompt-only interactions, MCP supports structured, multi-turn interactions and tool delegation, bringing LLMs closer to autonomous agent behavior [65], [66].

The core design of the MCP incorporates several key components to ensure reliable, secure, and extensible agent–tool interactions. It adopts a standardized message protocol, encoding requests and responses in formats such as JSON or MsgPack [67], which supports explicit function signatures, argument typing, and structured outputs. Tool registration and discovery are managed through a registry system, where individual tools—referred to as MCP servers—declare their capabilities and input/output schemas, enabling dynamic routing of model queries [68]. To ensure safety and robustness, each tool executes within an isolated environment governed by permission constraints, thereby preventing unintended side effects [61]. Furthermore, MCP enables streaming and stateful interactions, including asynchronous calls, callbacks, and memory-aware tool chaining, making it well suited for long-context workflows and iterative task execution [69], [70].

In practical terms, an MCP ecosystem consists of multiple **MCP servers**, these are lightweight services (often implemented in `Node.js` or `Python`) that expose tool interfaces like `getTestMetadata()`, `querySignalDefinition()`, or `runSimulation()`. The model communicates with these tools through the MCP runtime, which manages request scheduling, tool lookup, and response integration into the LLM’s generation process [71].

In the developed HIL-GPT system, the MCP serves as an orchestration layer that defines a suite of modular tools accessible to the agent for accomplishing domain-specific subtasks. These tools enable the agent to retrieve and process engineering data from local systems, dynamically execute test-related operations such as analysis, transformation, or validation, and compose multi-step workflows tailored to complex objectives. For example, the agent can perform a sequence such as “extract requirement → generate test → summarize result,” with the MCP ensuring consistency and coordination across tool invocations. This modular design enhances system extensibility and facilitates robust integration with existing engineering pipelines.

This architecture allows the system to overcome the inherent limitations of static LLM contexts. For example, rather than embedding all test case knowledge into model weights (which is costly and inflexible), the design allow the model to delegate structured subtasks to specialized external tools, and re-integrate the outputs into coherent, human-readable answers [72].

MCP thus represents a critical step toward operationalizing LLMs in high-stakes, tool-rich environments. It provides the glue layer between general-purpose language models and the dynamic, heterogeneous systems they need to operate within [73]. By doing so, it helps dissolve information silos and enables scalable, secure, and interpretable AI toolchains.

2.2.7 Private Data Integration

As LLMs are increasingly adopted in enterprise and safety-critical domains, integrating private and proprietary data into these systems has become a fundamental challenge. Unlike public benchmarks or internet-scale corpora, internal datasets, such as product specifications, test plans, source code, and sensor logs which are often sensitive, dynamic, and non-standardized. Leveraging this data effectively, while preserving security, compliance, and performance, requires carefully designed architectures.

One of the primary challenges is that foundational LLMs are pre-trained on public corpora and lack context about proprietary environments. This makes them ill-suited for in-domain reasoning without adaptation [29], [74]. Fine-tuning models with private data is effective but poses risks, including model leakage [75], reproducibility limitations, and cost. Moreover, retraining large models every time internal knowledge updates is impractical.

RAG provides a more flexible mechanism by decoupling model parameters from internal data [32], [34]. However, integrating private corpora into a RAG pipeline is non-trivial. Internal documents may be semi-structured, heterogeneous, and versioned. Signal identifiers, test step logic, and tool-specific metadata often require normalization before embedding [36], [38].

Security and privacy are equally important. Embedding private documents for retrieval may expose vector representations that can leak information under adversarial settings [76], [77]. Enterprises must consider encryption, access controls, and secure deployment strategies when building internal retrieval systems.

To mitigate the practical limitations of applying RAG in sensitive, domain-specific settings such as automotive testing, several strategies are commonly employed. Embedding-level adaptation—through domain-specific training or adapter-based tuning—enhances the relevance of vector search results over proprietary corpora [78]. Hybrid knowledge bases that combine structured databases with chunked document retrieval have been shown to improve both recall and precision [79]. To ensure data privacy, many deployments opt for fully on-premise execution of LLMs and vector indices, thereby keeping sensitive information within controlled environments [74]. Query annotation and context expansion, such as injecting metadata, signal identifiers, or hierarchical context, further improves retrieval accuracy in test-related tasks [37]. Finally, incorporating access logging and auditability mechanisms supports post-hoc inspection, enhancing transparency and reducing the risk of misuse.

In the implementation, the HIL-GPT agent accesses private test artifacts, signal catalogs, and documentation through a vectorized retrieval layer. These documents are pre-processed using automotive-specific chunking strategies and embedded using fine-tuned

models on structured signal/token datasets. The embedding index is deployed on a secured server inside the enterprise firewall. Queries are normalized to align with naming conventions in test databases, improving both recall and contextual alignment.

MCP-based tool calls are also used to fetch live data from internal services, combining retrieval with real-time validation. By integrating private data through both RAG and tool APIs, HIL-GPT achieves high relevance while maintaining strong data governance.

As organizations continue to scale their use of LLMs, the secure and efficient integration of private data will remain central. Whether for knowledge access, test automation, or human-AI collaboration, future systems must balance openness with confidentiality.

Chapter 3

Related Work

The aim of this chapter is to provide a structured overview of recent advances in LLM adaptation methods, specifically fine-tuning and RAG which focuss on their applicability to engineering domains such as automotive testing and HIL validation. Section 3.1 outlines the methodology used to identify and categorize relevant research, including a systematic literature search across three academic databases and the construction of keyword queries tailored to domain-specific adaptation scenarios.

Section 3.2 reviews prominent studies on full-parameter fine-tuning, highlighting both general trends and unique challenges encountered in embedded or structured data settings. Section 3.3 examines the growing body of work on RAG systems in technical and industrial contexts, focusing on document preparation pipelines, embedding models, and prompt construction strategies. A direct comparison between fine-tuning and RAG is presented in Section 3.4, addressing architectural, operational, and cost related trade-offs.

To ensure consistency, original terminology used in the reviewed works is preserved wherever applicable (“SimCSE,” “CPT,” “RAGO”). In the absence of explicit framework names, abbreviated references to study titles are employed. Consistent with prior literature [80], [81], the discussion emphasizes not only methodological aspects but also practical considerations such as latency, maintenance overhead, and interpretability—critical factors in real-world HIL deployments.

3.1 Literature Review Methodology

The literature review commenced with an exploratory search on Google Scholar to broadly assess the practical applications of fine-tuning and RAG techniques. As an initial step, publications containing the terms “*fine-tuning*” and “*retrieval-augmented generation*” in their titles were identified. This query produced a large number of results, although many were only tangentially related to the automotive or embedded systems domain. Relevance was therefore determined based on titles and abstracts, with works focused exclusively on NLP benchmarks or unrelated commercial use cases excluded.

To obtain a more domain-specific perspective, two additional databases—IEEE Xplore and arXiv—were consulted. Refined queries were applied to the abstract fields using terms such as *"automotive"*, *"domain-specific"*, *"signal"*, and *"HIL testing"* in conjunction with *"fine-tuning"* or *"RAG"*. The specific search queries are summarized in Table 3.1. These queries were designed to surface studies situated in engineering contexts, particularly those addressing challenges such as data heterogeneity, latency, and domain adaptation.

Each retrieved publication was assigned to one of two thematic categories: (1) studies that apply or investigate fine-tuning strategies in structured, signal-rich, or embedded environments; and (2) studies exploring RAG systems in engineering, automotive, or system-testing domains. Classification was occasionally non-trivial, as some papers referenced both retrieval and tuning strategies without clearly delineating their respective contributions. In such cases, categorization was based on the primary methodological focus.

Backward and forward citation tracking were also employed to identify additional relevant studies. While several publications mentioned automotive use cases, relatively few directly addressed the joint application of fine-tuning, RAG, and embedded HIL testing, highlighting a notable research gap. This gap forms the basis for further exploration in the present work. The final set of reviewed studies is presented in Table 3.2. Foundational studies on fine-tuning are discussed in Section 3.2, followed by RAG-related research in Section 3.3, and a comparative analysis in Section 3.4.

Table 3.1: Database and Search Queries for Literature Review

Database	Search Query
Google Scholar	allintitle: "fine-tuning" OR "retrieval-augmented generation" AND "automotive"
IEEE Xplore	("Document Title": fine-tuning OR RAG) AND (Abstract: "automotive" OR "HIL testing" OR "domain-specific")
arXiv	order: -announced_date_first; size: 50; include_cross_list: True; terms: fine-tuning AND (automotive OR HIL OR retrieval-augmented generation)

Table 3.2: Reviewed Research by Category

Category	Representative References
Fine-tuning	[82], [83], [84], [85], [86], [74], [87], [88], [89], [40], [44], [90], [91], [92], [93], [81], [94], [95], [96]
RAG + Fine-tuning	[88], [89], [97], [98], [31], [34], [99], [100]

3.2 Fine-Tuning for Domain Adaptation

Fine-tuning is a widely adopted technique to adapt LLMs to specific domains by continuing the training process using in-domain data. This approach helps models internalize

domain-specific terminologies, syntactic structures, and task requirements, thereby improving their performance in specialized settings.

Recent studies have explored fine-tuning strategies in various technical domains. For example, [101] provides a comprehensive survey on the generation of test sequences in embedded systems, highlighting the growing role of machine learning techniques — although applications of LLM fine-tuning remain scarce in this context.

In engineering design, Elhambakhsh et al. [102] proposes a fine-tuning pipeline for GPT-3.5 to classify mechanical assembly components based on functional descriptions. Their experiments in the Oregon State Design Repository dataset demonstrated clear benefits of domain adaptation in improving annotation accuracy.

Lu et al. [87] further compares various fine-tuning strategies: including Continued Pre-training (CPT), Supervised Fine-Tuning (SFT) and Direct Preference Optimization (DPO). They found that multi-strategy approaches yield synergistic benefits, particularly when tuning LLMs on task- and domain-specific data.

Despite these advancements, the adoption of fine-tuned LLMs in embedded or safety-critical automotive applications remains limited. Englhardt et al. [103] evaluates multiple LLMs in the context of embedded system development and observed that while general reasoning capabilities are strong, generating reliable and fully functional code for hardware interfaces (I²C) remains a challenge.

These findings suggest an underexplored opportunity for future work: developing tailored fine-tuning pipelines for embedded automotive test systems - particularly in HIL testing scenarios - where traditional data scarcity and heterogeneity could be mitigated through pretraining/fine-tuning strategies on curated domain corpora.

3.2.1 Full-Parameter Fine-Tuning

Full-parameter fine-tuning refers to updating all the weights of a pre-trained language model during domain or task adaptation. That is, the entire parameter set θ is optimized using gradients computed from the downstream objective. This approach directly adjusts the internal representations of the model to better reflect the statistical patterns of the new data.

Although complete fine-tuning is often considered computationally prohibitive for very large models (7B+ parameters), it remains highly effective and tractable for moderate-sized models, such as those in the 100M–1B parameter range. In such models, full-parameter updates can be performed on standard GPU clusters without the need for sophisticated parallelism frameworks, making it an attractive choice in industrial settings where model size is constrained by latency or hardware requirements.

Compared to parameter-efficient tuning (PET) techniques such as adapters, LoRA, or prefix tuning, full-parameter fine-tuning offers the following advantages in small- to medium-scale settings:

- **No Architectural Modifications:** The base model remains structurally unchanged, simplifying implementation and deployment.
- **Maximum Expressiveness:** All parameters are free to adapt, allowing deeper alignment with domain-specific distribution shifts.
- **Better for Out-of-Distribution Generalization:** In smaller models, freezing most weights—as in PET methods—can overly constrain the model, leading to underfitting on specialized domains [104].

However, full-parameter fine-tuning is not without limitations. It requires more training steps, careful regularization to avoid overfitting (especially on small domain datasets), and thoughtful version control when used in continuous integration (CI) pipelines.

Nonetheless, for lightweight models deployed in real-time automotive applications or embedded systems, full fine-tuning remains a strong and practical baseline. It offers a balanced trade-off between adaptability and simplicity—particularly when large-scale retrieval infrastructures, such as those required by RAG, are unavailable or infeasible.

3.2.2 Training Data Construction in Industrial Contexts

A recurring challenge across many industrial applications of large language models is the scarcity of labeled training data. In domains such as embedded system testing or automotive validation, domain knowledge is often stored in unstructured formats—ranging from legacy spreadsheets and GUIs to in-house documentation—making supervised learning difficult to apply directly [27].

To overcome this, recent works have explored weak supervision and contrastive learning as alternatives to manual labeling. In particular, triplet-based contrastive learning has gained traction for tasks involving semantic matching or retrieval. This approach involves training on triplets of the form (a, p, n) , where a is an anchor sample, p is a semantically similar positive, and n is a dissimilar negative. Studies such as SimCSE [105] and others have demonstrated that triplet-based objectives can effectively learn high-quality sentence embeddings without requiring explicit task-specific labels.

In the context of system engineering, heuristics based on shared identifiers, component co-occurrence, or textual similarity have been used to generate weakly labeled triplets for contrastive learning [37]. Moreover, engineer-in-the-loop feedback mechanisms are occasionally employed to improve triplet quality in safety-critical settings.

Although not widely adopted yet in HIL testing pipelines, these strategies provide a promising direction for training domain-adapted LLMs under low-resource conditions. In the following chapters, the project build on this insight to construct a triplet-based corpus for embedding fine-tuning using unstructured requirement and test sequence data.

3.3 RAG in Engineering Contexts

RAG is an increasingly prominent paradigm for enabling LLMs to generate more accurate and context aware responses by grounding their outputs in external knowledge sources. Unlike fine-tuning, which encodes domain knowledge directly into the model weights, RAG architectures combine a frozen or lightly tuned LLM with a retriever that surfaces relevant information at inference time. This design is particularly well-suited to domains characterized by dynamic, fragmented, or large-scale information repositories—such as engineering systems, test documentation, or embedded development workflows.

Several works have explored RAG-based systems in technical and industrial contexts. For example, Zhang et al. [88] proposed a dual-encoder RAG framework for intelligent industrial question answering, leveraging both code and specification retrieval to support developers in safety-critical environments. Similarly, [31] demonstrated the effectiveness of retrieval-augmented transformers in system diagnostics, where real-time access to domain-specific fault descriptions was shown to enhance interpretability and correctness.

In the embedded systems space, Yu et al. [89] integrated RAG with domain-pretrained encoders for microcontroller diagnostics, showing that retrieval latency and relevance quality were key performance bottlenecks. Their ablation studies suggest that while fine-tuning improves the model’s linguistic alignment with domain terminology, RAG remains indispensable when knowledge is too sparse or volatile to encode in static parameters.

A notable benefit of RAG is its modularity: the retriever and generator can be independently improved or adapted. This aligns well with industry workflows where knowledge bases evolve rapidly, and tools must remain interpretable, updatable, and compliant with version control policies. Moreover, RAG naturally supports hybrid reasoning workflows like combining code snippets, tabular logs, and textual specifications which are common in automotive testing and HIL pipelines.

Despite its promise, RAG also presents unique challenges in engineering contexts:

- **Retrieval Quality:** Engineering documents often lack natural language consistency, making dense retrieval harder to calibrate.
- **Latency Constraints:** Real-time use cases such as HIL testing may impose strict latency requirements that conventional RAG pipelines cannot yet satisfy.
- **Tool Integration:** Unlike static document answering, engineering use cases often involve multi-modal reasoning, external tool calls, or dynamic document composition.

3.3.1 Database Construction for Retrieval

The first step in building a RAG system is constructing a retrieval-friendly document database. In industrial contexts such as automotive system validation, source materials

are often heterogeneous—ranging from PDFs, requirement spreadsheets, and legacy documentation to markdown files and GUI-exported logs. These raw inputs must be converted into a structured, retrievable corpus.

Recent work in automotive-specific RAG pipelines has demonstrated the centrality of PDF-aware preprocessing—including layout-sensitive chunking and metadata alignment with Liu et al. [106] reporting significant gains in retrieval precision through tailored chunking strategies. Other approaches, such as CFIC [107], explore chunking-free methods to maintain document coherence by operating on hidden states rather than explicit text segments.

Furthermore, Bhat et al. [108] empirically analyze fixed-size chunking strategies over long documents, revealing the trade-off between fine- and coarse-grained chunk granularity. In domains like financial reporting, Yepes et al. [109] show that element-type based chunking can outperform naive paragraph splits in retrieval effectiveness.

This process typically involves several steps aimed at transforming raw industrial documents into a retrieval-friendly format. First, chunking is applied to split long documents into semantically coherent segments—such as paragraph-level or instruction-level units—to improve retrieval granularity. Next, normalization procedures are carried out to remove formatting noise, standardize encodings, and extract clean textual content from structured elements like tables, figures, or GUI exports. Finally, metadata tagging is performed by associating each chunk with relevant attributes such as system version, component ID, or requirement category, thereby enabling filtered and context-aware retrieval. The outcome is a cleaned and structured corpus suitable for embedding and indexing in downstream RAG pipelines.

3.3.2 Embedding Model Utilization

After preparing a structured document corpus, each chunk is encoded into a dense vector—or embedding—that semantically represents its content. These embeddings enable identifying relevant passages through efficient nearest-neighbor search in high-dimensional space, forming the core mechanism of the RAG retriever.

Choosing the right embedding model is critical. General-purpose options like Sentence-BERT [110] are widely used, as they produce accurate sentence-level encodings optimized via Siamese or triplet architectures. Alternatively, E5—a contrastively pretrained model—delivers state-of-the-art performance across retrieval benchmarks and supports both zero-shot and fine-tuned retrieval tasks [111].

In domain-specific scenarios (automotive logs or test procedures), embedding quality often degrades due to specialized terminology. To address this, embedding models can be fine-tuned using in-domain data, such as triplet sets where each anchor and positive belong to the same subsystem or test case, and negatives are semantically unrelated. This fine-tuning sharpens the embedding space, improving retrieval relevance under domain-shift conditions [105].

High-quality embeddings are vital because they directly determine what context the LLM sees. If retrieved passages are off-topic or semantically distant from the query intent, the RAG output will suffer in accuracy and coherence. Thus, the embedding model not only enables efficient retrieval—it also lays the foundation for grounded, factually relevant generation by the LLM.

Choosing the right embedding model is critical. General-purpose, open-source options such as Sentence-BERT [110], E5 [111], and its multilingual variant [112], as well as arctic-embed [113], offer strong baselines with favorable performance on benchmarks like MTEB and long-context retrieval. Proprietary alternatives (for example OpenAI Ada-002, Cohere embed-v3) may perform competitively, but open-source models like E5-base-v2 have shown equal or superior accuracy at lower latency and cost [114].

Table 3.3: Comparison of Open-Source and Closed-Source Embedding Models

Model Type	Approx. Count	Representative Examples
Open-Source	~120	[110], [111], [113], [112], [105]
Closed-Source	~10	[114] (OpenAI Ada-002, OpenAI text-embedding-3-small/large Cohere embed-v3, etc.)

To guide model selection for domain-specific retrieval, this project will benchmark several top-ranked models from public leaderboards—including MTEB and the Sentence-Transformers model zoo—under industrial test case and requirement retrieval settings.

3.3.3 Fine-Tuning Data Preparation for Embeddings

Open-source embedding models are especially valuable because they are transparent, customizable, and avoid vendor lock-in, while enabling fine-tuning to accommodate domain specific grammar and terminology.

In industrial scenarios—such as automotive logs, control signals, or test procedure datasets generic embedding models often underperform due to the presence of highly specialized vocabulary and structural patterns. To mitigate this, embedding models can be fine-tuned using in-domain data through contrastive learning, typically with triplet sets. In this setup, each triplet consists of an anchor and a positive sample that share the same subsystem, test case, or signal lineage, while the negative sample is drawn from unrelated modules or functionally distant requirements. This training strategy encourages the model to learn domain-specific semantic structures by bringing related instances closer together in the embedding space while pushing irrelevant ones further apart.

These triplets (a, p, n) guide the model to cluster semantically related chunks together while pushing irrelevant ones apart. This approach has been demonstrated to substantially improve retrieval performance under domain shift [91], [105]. Further, industrial benchmark studies—such as Databricks’ evaluation on enterprise datasets—confirm that embedding fine-tuning yields marked improvements in Recall@10 and overall RAG accuracy, even without manual labeling [92], [114].

Embedding quality plays a pivotal role: it directly affects the relevance of retrieved chunks, which in turn determines the factuality and contextual integrity of the LLM’s output. In fact, domain-specific fine-tuning of embedding models has been identified as the primary driver of RAG improvements—often more impactful than fine-tuning the LLM itself [93]. Recent methods like HEAL [115] and MAFIN [116] push this further by introducing hierarchical or model-augmented fine-tuning strategies that adapt embeddings efficiently even with limited data.

Overall, embedding fine-tuning is now recognized as one of the most effective techniques for enhancing RAG systems in specialized domains—especially in settings like HIL testing, where data heterogeneity and terminology specificity demand tailored semantic alignment.

3.3.4 Combining Documents with Large Language Models

Once the retriever and embedding store are in place, a user query is converted into an embedding and used to fetch top- k relevant document chunks. These retrieved passages are then concatenated with the original query to construct an augmented prompt for the LLM.

This prompt engineering step is critical: it must balance relevance, context diversity, and the token budget. Common techniques include:

- Wrapping retrieved text in `<BEGIN_CTX> . . . </END_CTX>` delimiters or prefixing with metadata such as source ID, timestamp, or version—notably improving reasoning and attribution in the generated output [117].
- Truncation or compression of overly long passages, or employing prompt-level query refinement to reduce noise [117].
- Applying response-level control instructions (for example “If unsure, respond with ‘I don’t know’”) to improve factuality and reliability [117].

When prompted in this manner, the LLM generates responses that are explicitly grounded in retrieved content, which substantially reduces hallucinations and improves factual alignment compared to vanilla prompt-based LLM usage [32], [117], [118].

Furthermore, the architecture supports dynamic reasoning over evolving corpora: as documentation, logs, or test cases update, new chunks are indexed and become immediately available during retrieval. This real-time adaptability and grounding are crucial in industrial environments like HIL testing, where documentation and system specifications frequently change.

3.4 Comparison Between Fine-Tuning and RAG Strategies

Fine-tuning and RAG represent two prominent strategies for adapting LLMs to domain-specific tasks as mentioned in previous sessions. While both approaches aim to enhance performance on specialized queries, they differ fundamentally in their architecture, data requirements, latency characteristics, and deployment complexity. This section provides a comparative analysis to support design decisions in later chapters.

3.4.1 Adaptation Mechanism

Fine-tuning directly modifies the internal weights of an LLM using labeled or pseudo-labeled data, enabling the model to internalize domain-specific syntax and logic patterns [104]. However, this process is compute-intensive and time-consuming, often requiring multiple GPU-days or even GPU-weeks depending on model size and dataset volume [81], [95]. In contrast, RAG systems maintain a frozen or lightly-tuned LLM and rely on an external retriever module. This enables lower setup costs, as expensive offline training is avoided, and faster iteration cycles, since new data can be integrated simply by updating the document index—not the model itself [32], [81], [94].

For enterprise adoption, these differences are critical. Fine-tuned models demand specialized engineering expertise, large memory footprints, and a maintained retraining pipeline raising total cost of ownership [94], [95]. By contrast, RAG architectures are more accessible: teams can integrate internal documentation sources such as requirement spreadsheets or test logs into a vector store and connects them to a frozen LLM—providing up-to-date, traceable responses without retraining the model [80], [94]. Indeed, surveys indicate that only about 20% of enterprises employ full fine-tuning, while approximately 80% adopt RAG due to its scalability, security, and lower infrastructure burden [80], [95]. Moreover, enterprises often achieve 5–15× savings in compute cost by choosing RAG over full fine-tuning, with faster time-to-value and improved auditability [81].

Thus, while fine-tuning offers deep domain specialization and lower inference latency, its high upfront costs and maintenance overhead limit its practical enterprise use. In contrast, RAG strikes a pragmatic balance—enabling organizations to leverage rich internal knowledge at scale, with lower expertise requirements and operational complexity.

3.4.2 Data Requirements and Flexibility

Fine-tuning typically requires a substantial amount of in-domain training data, which can be challenging in settings like HIL testing due to data scarcity. RAG, however, can leverage unstructured or unlabeled corpora—such as requirement documents or logs—and incorporates new content immediately without retraining [32], [119].

From an enterprise perspective, this flexibility translates directly into reduced engineering overhead and faster deployment cycles. A typical full fine-tuning workflow can consume several GPU-days and requires expertise in data labeling, training pipeline maintenance,

and model versioning [95]. In contrast, RAG systems allow organizations to “drag and drop” internal documents into vector indexes and instantly update the system’s knowledge base, significantly lowering the barrier to entry for enterprise LLM use [120], [121]. This is particularly valuable in regulated domains (automotive, aerospace), where documentation evolves frequently and auditability is critical.

Furthermore, up to 90% of enterprise data is unstructured—logs, PDFs, emails—making RAG the preferred approach for knowledge retrieval in practical business environments [121], [122]. Industry reports suggest that around 36% of enterprise LLM deployments now utilize RAG frameworks due to their scalability and minimal maintenance requirements [123]. Additionally, enterprise-grade studies highlight that RAG systems can reduce development and infrastructure costs by up to 5–10× compared to fine-tuning efforts, with faster time-to-value and improved compliance tracking due to explicit source referencing [121].

3.4.3 Latency and Resource Implications

Inference with fully fine-tuned models typically achieves the lowest latency, as retrieval is not required—completion takes just a single forward pass through the network. However, training these models can incur significant offline compute costs, often requiring multiple GPU-days depending on the model size and dataset scope [124].

RAG pipelines introduce additional steps—vector retrieval and prompt assembly—that increase inference time. Recent systems such as RAGO optimize RAG serving and demonstrate reductions in time-to-first-token of up to 55%, while sustaining or improving throughput (QPS) [125]. Despite this, pipelines still exhibit roughly 30–50% higher end-to-end latency compared to directly using fine-tuned LLMs [126].

These stages can be independently optimized. Vector lookups via FAISS or HNSW offer millisecond-level retrieval, while LLM inference can leverage token-caching and batching to boost throughput [126]. For example, Cache-Craft achieved up to a 2× reduction in response latency by caching chunk embeddings across queries [126].

From an enterprise implementation perspective, these trade-offs are important:

- Fine-tuning: provides ultra-low latency suitable for time-critical tasks but requires expensive infrastructure and retraining pipelines, which complicates versioning and deployment [127].
- RAG: introduces moderate latency due to retrieval, but offers flexibility—indexes can be updated without retraining, leading to agile maintenance and deployment in enterprise environments where documentation evolves rapidly [125], [126].

Moreover, adaptive caching and selective retrieval strategies allow RAG systems to approach fine-tuned model latencies while preserving key advantages like modularity and auditability—making them viable for real-world HIL system deployments.

3.4.4 Maintenance and Interpretability

In industrial settings where traceability is critical, RAG provides enhanced transparency by including explicit references to source documents in generated outputs. This facilitates auditing, error diagnosis, and compliance verification—key requirements in regulated domains such as automotive and aerospace [32], [128]. In contrast, fine-tuned LLMs often act as “black boxes” after deployment, making it difficult to trace how decisions are derived from internal weights [129], [130]. Interpretability of LLMs has long been a concern. Even industry leaders admit that current models often lack explainable reasoning pathways except through output behavior analysis [131]. Efforts in mechanistic interpretability—such as neuron-level analysis and activation probing—have demonstrated progress. For example, Anthropic researchers identified neuron clusters (“features”) corresponding to concept representations, enabling targeted modification of model behavior [131], [132].

Within RAG systems, interpretability extends beyond architectural transparency. The retrieval layer itself can be audited: each output includes context from specific document chunks, reducing hallucination risk and aiding in forensic reconstruction when outputs are incorrect—vital for root cause analysis in HIL environments [128]. Additionally, combining RAG with fine-tuned LLMs can create hybrid architectures that maintain modular audit trails, helpful for maintaining system integrity in long-term deployments.

Overall, RAG’s combination of context-grounded generation and retrieval traceability provides a pragmatic compromise: it improves interpretability and compliance without sacrificing performance—an ideal configuration for safety-sensitive and regulation-bound industrial applications.

3.4.5 Summary of Trade-Offs

Overall, RAG offers enterprises a robust and practical strategy for domain adaptation, as it eliminates the need for extensive labeled datasets, supports continuous knowledge updating without retraining, ensures traceable outputs linked to source documents, and reduces both compute and engineering costs. These advantages make RAG particularly well-suited for HIL testing environments, where data is generated continuously and adaptability and auditability are essential. In summary, fine-tuning and RAG offer complementary benefits: fine-tuning excels in settings with abundant high-quality domain data and strict latency requirements, whereas RAG proves advantageous in dynamic, data-scarce environments that prioritize interpretability and modularity. Consequently, this thesis adopts a hybrid strategy—employing fine-tuned embedding models to enhance retrieval within a RAG framework tailored for HIL testing scenarios.

Table 3.4: Comparison of Fine-Tuning and RAG for Domain Adaptation

Aspect	Fine-Tuning	RAG
Adaptation Method	Updates model weights via domain data	Uses frozen LLM + external retriever
Data Requirement	Requires substantial in-domain labeled data	Can operate on unstructured or unlabeled corpora
Inference Latency	Low (single-pass inference)	Moderate (retrieval + generation)
Update Cost	High (needs retraining)	Low (documents can be updated independently)
Interpretability	Low transparency post-fine-tuning	High (responses linked to explicit sources)
Suitability for HIL	Effective with abundant data	Flexible and modular in sparse-data settings

Chapter 4

Architecture

This chapter presents the architectural design of the implemented AI-based assistant system, developed to support requirement understanding and test sequence retrieval in the context of HIL testing at the company. Building on the insights and preparatory steps outlined in the previous chapters, the system integrates both RAG and fine-tuned language models to enable context-aware, traceable, and efficient interaction with domain-specific engineering data. The architecture is modular, comprising a document processing pipeline, embedding and vector indexing modules, a RAG enhanced agent, and a frontend query interface. This design was shaped by the constraints of the industrial setting, where latency, interpretability, and CI pipeline integration are crucial. The following sections detail each component and their interaction within the broader workflow.

4.1 Overall Design

The architecture of the proposed system is tailored to support intelligent integration and recommendation of test requirements and test sequences within the company's HIL testing environment. It is structured around two core capabilities: (1) semantic understanding and retrieval of relevant engineering content using fine-tuned embedding models, and (2) contextualized response generation via a RAG pipeline.

A modular architecture has been adopted to enable flexibility across components such as embedding, retrieval, generation, and user interaction. Figure 4.1 illustrates the system's high-level architecture, showing the flow of information from raw data sources to the final user-facing responses.

The pipeline begins with structured data ingestion from selected internal systems that host requirement specifications and test sequences. These inputs are preprocessed through segmentation, normalization, and metadata enrichment before being converted into vector representations using a fine-tuned sentence embedding model trained on triplet examples specific to the testing domain.

At inference time, user queries are managed by a hybrid agent that can retrieve semantically relevant documents from the vector store or, when necessary, generate contextualized

answers using a LLM. The generated responses are then returned to a user-facing interface, which is integrated into commonly used engineering environments such as Visual Studio Code or Slack, ensuring minimal disruption to existing workflows.

This modular and hybrid design enables the system to meet key industrial requirements, including traceability, low-latency response, and compatibility with continuous integration pipelines, while also maintaining the flexibility to evolve alongside changing data sources and engineering practices.

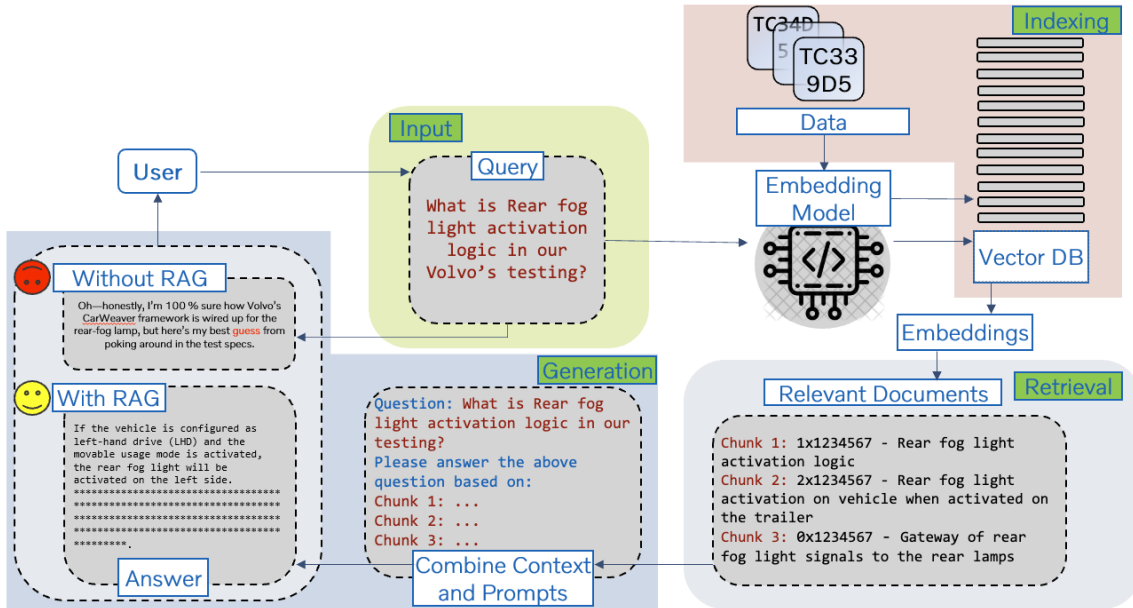


Figure 4.1: High-level system architecture for hybrid retrieval and generation

4.2 Data Acquisition and Preprocessing

4.2.1 Data Collection and Curation for LLM-Driven Systems

The effectiveness of LLM-based applications in system testing and embedded environments hinges on the availability of structured, domain-specific data. In this context, data were obtained directly from internal tools (Software1, Software2, and Software 3) that collectively store system requirements, component hierarchies, and test sequences. These tools expose the data either through internal APIs or structured exports, enabling direct extraction of raw textual and tabular content.

Although the data sources were accessible, their heterogeneity posed challenges. Requirements were often written in unstandardized language, and test sequences were dispersed across various formats with inconsistent schema definitions. Consolidating this data into a coherent corpus suitable for LLM training and retrieval required significant preprocessing and normalization.

4.2.2 Industrial Data Annotation and Preprocessing

Industrial datasets are rarely curated for machine learning purposes. Requirements may be presented as free-form paragraphs, semi-structured tables, or use domain-specific notations, while test sequences embed control logic or signal dependencies in procedural formats [49], [133]. Preprocessing entailed token normalization, segmentation into coherent chunks, metadata tagging, and relationship extraction among signals, components, and behavioral logic [134], [135].

The absence of labeled training data necessitated the use of weak supervision [136], distant supervision, or unsupervised similarity heuristics [105]. These methods supported contrastive learning setups with minimal manual labeling effort, enabling the generation of embedding models for downstream semantic search [38].

4.2.3 Challenges in Structuring Requirements and Test Assets

Semantic structuring of requirement and test artifacts introduced considerable complexity. Requirements were often duplicated across product variants, authored using inconsistent formats, and lacked shared taxonomies [137]. Test sequences encoded conditional logic, preconditions, and signal dependencies across versions and components [27]. To transform raw content into meaningful retrieval units, domain-specific chunking strategies and standardized metadata schemas were applied.

Since formal traceability between requirements and test sequences was limited, heuristics were employed to infer associations based on overlapping signal names, module identifiers, or functional tags [37], [90]. Embedding models were trained with auxiliary inputs such as signal IDs and version metadata to improve retrieval accuracy.

Versioning further complicated data integration, as both requirements and test sequences evolved frequently. To preserve traceability while accommodating updates, the preprocessing pipeline was designed to support incremental alignment across linked artifacts.

4.2.4 Data Source Diversity and Stakeholder-Guided Selection

The internal data environment spanned several specialized tools, each designed for specific lifecycle stages and teams. Despite accessibility, the lack of format standardization and semantic consistency across these tools made unified indexing and retrieval challenging. Rather than performing exhaustive ingestion, a stakeholder-driven strategy was employed.

Through interviews with test engineers and system architects, high-priority tools and commonly referenced datasets were identified—most notably Software1 for subsystem level test sequences, Software2 for component metadata, and Software 3 for requirement specifications. This approach ensured that the curated dataset remained aligned with practical usage and traceable across relevant engineering processes.

4.2.5 From Raw Extraction to Structured Knowledge Units

Raw textual and tabular data were often noisy, redundant, or incomplete. Preprocessing involved deduplication, sentence segmentation, field normalization, and metadata enrichment using both domain heuristics and stakeholder feedback. This process aimed to preserve semantic integrity while ensuring compatibility with embedding based retrieval systems.

The refinement process was iterative and guided by feedback from domain experts, who helped validate chunking boundaries, signal naming conventions, and tagging schema. This ensured that the final dataset retained operational relevance and improved the downstream performance of the RAG retrieval module.

Despite being time intensive, this phase laid the foundation for robust semantic search and ensured the model’s alignment with real-world testing and requirement engineering practices.

Dataset Structuring for Retrieval and Training. To support the dual use case of RAG-based inference and embedding fine-tuning, two data formats are defined:

- **Knowledge Base Format (RAG Corpus):** Each entry represents a semantically coherent retrieval unit. The required fields include:
 - **id:** A unique identifier;
 - **title:** A short label or functional descriptor;
 - **requirements:** Core textual content intended for retrieval;
 - **description (optional):** Supplementary information or metadata;
 - **sequences (optional):** Associated procedural or test related steps;
 - **category:** A tag indicating origin or classification source.

This format is serialized as a JSON array and used to populate the embedding vector store during inference.

- **Triplet Training Format:** To fine-tune embedding models via contrastive learning, a dedicated dataset is constructed consisting of anchor–positive–negative triples. Each sample includes:
 - **anchor:** A query like input or prompt representative of downstream usage;
 - **positive:** A semantically relevant corpus item;
 - **negative:** A semantically irrelevant or misleading item drawn from the corpus.

This format can be prepared using heuristic mining, lexical retrieval methods, or prior system interaction logs, and is typically stored as a list of JSON records.

These two formats are designed to ensure separation between inference-ready content and supervision-ready signals, while maintaining consistency in document structure and semantic granularity. Their adoption facilitates reproducibility, model interoperability, and future extension of the RAG pipeline within industrial settings.

4.3 Embedding and Vector Indexing Layer

Once the requirement and test sequence data were preprocessed into clean, structured textual units, the next step was to convert them into semantically meaningful vector representations suitable for retrieval. This section describes the design and implementation of the embedding model, the training process for domain adaptation, and the integration of a vector indexing layer optimized for industrial usage.

4.3.1 Model Selection and Fine-Tuning Strategy

To identify a suitable base embedding model for semantic retrieval in the automotive testing domain, the project first conducted a comparative analysis of both open-source and proprietary models using the MTEB (Massive Text Embedding Benchmark) leaderboard as a reference. The objective was to select models that offered a strong balance between retrieval quality and computational efficiency.

The initial candidate pool included a range of open-source models, including:

- `all-distilroberta-v1` [138];
- `all-mpnet-base-v2` [139];
- `multi-qa-mpnet-base-dot-v1` [139];
- `multi-qa-mpnet-base-cos-v1` [139];
- `all-roberta-large-v1` [140];
- `bge-base-en-v1.5` [141];
- `gtr-t5-large` [38];
- `gtr-t5-xl` [38].

Additionally, two closed source models from OpenAI were considered:

- `text-embedding-ada-002` [142];
- `text-embedding-3-small` [143].

A small-scale evaluation was performed on an internal validation set representing company-specific requirement and test sequence queries. This step helped identify three open models as strong candidates for domain adaptation: `bge-base-en`, `gtr-t5-large`, and `gtr-t5-xl`, covering both moderate and high parameter ranges.

To adapt the selected models to the automotive test engineering context, the project performed fine-tuning using contrastive learning with domain-specific triplets. Each triplet consisted of an anchor (a requirement statement), a semantically aligned positive (a corresponding test sequence), and a negative sample from an unrelated subsystem. Two fine-tuning formats were explored:

- **Triplet format:** Direct use of anchor–positive–negative samples with triplet loss. For example, given an anchor input describing a specific test requirement (like “Brake pressure must be stable under 80 bar”), a semantically similar test case is used as the positive (a test sequence verifying brake pressure stability), while an unrelated or misleading case serves as the negative (a test sequence for seatbelt warning).
- **Pairwise format:** Reformulated as anchor–positive pairs using binary contrastive loss, without an explicit negative. For instance, the anchor might be the same requirement statement, and the paired positive is its corresponding test case, while training relies on distinguishing from randomly sampled non-matching pairs at the batch level rather than using designated negatives.

These approaches produced different embedding behaviors, with triplet loss yielding tighter semantic clustering, while pairwise training offered improved generalization in sparse retrieval scenarios. Final model selection was based on downstream retrieval performance within the RAG pipeline.

4.3.2 Vector Store and Indexing Scheme

To support scalable and low latency semantic retrieval in the cloud, the system adopts **Azure Cosmos DB for MongoDB vCore** as its vector store. This managed database supports native dense vector indexing and approximate nearest neighbor search using the `$vectorSearch` operator under the `cosmosSearch` extension.

Index Definition. Two types of indexes were created for each collection: a dense vector index and a traditional metadata index. The vector index was configured with the following MongoDB command:

```
db.command({
  'createIndexes': COLLECTION_NAME,
  'indexes': [
    {
      'name': 'VectorSearchIndex',
      'key': {
        "contentVector": "cosmosSearch"
      },
      'cosmosSearchOptions': {
        'kind': 'vector-ivf',
        'numLists': 1,
        'similarity': 'COS',
        'dimensions': 768
      }
    }
  ]
})
```

Listing 4.1: Creating a vector index in Cosmos DB using MongoDB API

Cosine Similarity. To measure semantic closeness between vectors, cosine similarity is used as the scoring function:

$$\text{sim}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \cdot \|\mathbf{d}\|} \quad (4.1)$$

where \mathbf{q} is the query vector and \mathbf{d} is a document embedding. All vectors are normalized prior to indexing, allowing inner product computation to approximate cosine similarity.

In parallel, traditional filter indexes were added to support keyword filtering during retrieval, such as:

```
db.command({
  "createIndexes": COLLECTION_NAME,
  "indexes": [
    {
      "key": { "title": 1 },
      "name": "title_filter"
    }
  ]
})
```

Listing 4.2: Creating a keyword filter index on the 'title' field

Retrieval Pipeline. At query time, the user input is converted into an embedding using the selected model and passed into a MongoDB aggregation pipeline. The retrieval is conducted via the `$search` stage as shown below:

```

pipeline = [
  {
    "$search": {
      "cosmosSearch": {
        "vector": query_embedding,
        "path": "contentVector",
        "k": 5,
        "filter": {
          "title": { "$regex": keyword, "$options": "i↔
↔ " }
        }
      },
      "returnStoredSource": True
    }
  },
  {
    "$project": {
      "similarityScore": { "$meta": "searchScore" },
      "document": "$$ROOT"
    }
  }
]

```

Listing 4.3: Cosmos DB vector search pipeline with keyword filter

Top- k Vector Retrieval. Given a query vector \mathbf{q} and a corpus of N embedded documents $\{\mathbf{d}_1, \dots, \mathbf{d}_N\}$, the retrieval operation returns the k nearest vectors according to similarity:

$$\text{Top-}k(\mathbf{q}) = \arg \max_{\mathbf{d}_i \in \mathcal{D}, i=1, \dots, N}^k \text{sim}(\mathbf{q}, \mathbf{d}_i) \quad (4.2)$$

where $\text{sim}(\cdot, \cdot)$ is typically cosine similarity.

Industrial Justification. The decision to adopt Cosmos DB was strongly motivated by industrial constraints. Company’s internal data governance requires all production systems to run on Azure hosted infrastructure. Furthermore, locally hosting a vector database such as FAISS [144] or Qdrant [145] was evaluated but ultimately discarded due to concerns over limited throughput, operational resource demands, and integration complexity in CI/CD pipelines.

FAISS, although highly optimized for similarity search on local hardware, lacks out-of-the-box support for high availability, real-time updates, and cloud-native scaling [144]. Qdrant offers a more modern feature set including RESTful APIs and metadata filtering, but requires additional deployment overhead and infrastructure management [145], [146]. By contrast, Cosmos DB provides fully managed vector search capabilities, serverless

maintenance, and seamless integration with Azure-native services including authentication, logging, and monitoring [147]. This made it particularly well-suited for continuous integration within an enterprise-grade environment.

4.3.3 Periodic Re-Embedding and Pipeline Integration

Given the evolving nature of requirement specifications and test sequences in the automotive testing domain, the system supports *periodic re-embedding* and dynamic updates to the vector store. As new artifacts are introduced or existing ones are revised, their embeddings are recomputed using the current fine-tuned model and upserted into the Azure Cosmos DB vector index. This ensures that the retrieval pipeline remains aligned with the latest knowledge base, without requiring a complete re-ingestion.

Formally, let $\mathcal{D}_{\text{new}} = \{d_1, d_2, \dots, d_m\}$ represent newly added or modified documents. Each d_i is passed through the embedding function $f_\theta(\cdot)$:

$$\mathbf{v}_i = f_\theta(d_i), \quad \forall d_i \in \mathcal{D}_{\text{new}} \quad (4.3)$$

where f_θ denotes the current fine-tuned embedding model. The resulting vectors \mathbf{v}_i are then inserted into the existing collection \mathcal{V} using an `upsert` operation, preserving metadata and index consistency.

The embedding and vector indexing service is exposed via a dedicated backend endpoint. At query time, the user input q is embedded as $\mathbf{q} = f_\theta(q)$ and compared against stored vectors using cosine similarity mentioned before.

The top- k retrieved chunks are then passed downstream to the generation agent, which constructs a response using either a prompted LLM or a fine-tuned decoder.

4.4 RAG Agent

The generation agent constitutes the reasoning and response layer of the system. It leverages RAG to synthesize context-aware outputs for engineering queries based on semantically matched requirement and test sequence fragments. This approach combines the strengths of dense retrieval with the generative capabilities of LLMs, while preserving interpretability through reference tracking and tool-level invocation traces.

4.4.1 Context Construction from Retrieved Chunks

Given a user query q , the system retrieves the top- k most semantically relevant documents $\{\mathbf{d}_1, \dots, \mathbf{d}_k\}$ from the vector store, as described in previous sections. Each document consists of a content block c_i and associated metadata (for example test case ID, version tag, signal list). These chunks are concatenated into a single input context C_q as follows:

$$C_q = \text{concat}(c_1, c_2, \dots, c_k) \quad (4.4)$$

To manage the token limit imposed by the LLM, the system applies a sliding window or top priority truncation strategy based on relevance scores or document hierarchy.

4.4.2 Prompt Template and Generation Interface

The retrieved context C_q is then inserted into a domain-specific prompt template, which guides the model to produce grounded and verifiable answers. A representative prompt follows this structure:

```
You are an assistant specialized in HIL testing. Based on the follow-
ing reference information, answer the query:
--
[Retrieved Context]
--
Query: "[User Question]"
Answer:
```

This prompt is passed to the backend LLM endpoint, which returns the generated response \hat{y} based on both the context and query.

4.4.3 Tool Integration and External Invocation

In addition to passive retrieval, the agent supports dynamic tool invocation to fulfill more complex queries—such as retrieving test cases directly from the Volvo CarWeaver system. Tools are defined using the function calling schema compatible with Azure OpenAI’s ‘functions’ interface, allowing structured registration and invocation at runtime.

At inference time, if the model determines that external tool execution is required (for real-time database lookup), it emits a structured call conforming to this schema. The backend system then routes this request to the registered function and feeds the result back into the agent’s response pipeline. This mechanism enables hybrid reasoning workflows that combine static document grounding with live system interrogation.

4.4.4 Response Composition and Traceability

The final answer \hat{y} is computed as:

$$\hat{y} = \text{LLM}(C_q, q) \quad (4.5)$$

In addition to the generated text, the system logs which documents or tools contributed to the final answer. For retrieval-based responses, metadata such as document ID, test version, and signal name are embedded in the response trace. For tool-based results, the tool name, parameters, and returned payload are logged. This dual traceability spanning both retrieval and function execution—is critical in regulated industrial environments, where generated answers must be auditable and reproducible.

Chapter 5

Implementation

This chapter details the practical realization of the system architecture introduced in Chapter 4. While the previous chapter focused on high level design principles and modular interactions, the current discussion addresses how each component was concretely implemented, integrated, and deployed within the constraints of Volvo’s infrastructure.

The implementation encompasses several interdependent layers, including automated data extraction from GUI-bound engineering tools, embedding model fine tuning using contrastive learning, cloud-based vector indexing with hybrid metadata filtering, and a RAG pipeline powered by large language models. These components are orchestrated through a modular backend service, which communicates with a lightweight frontend for real-time user interaction.

Particular attention is paid to industrial constraints such as tool compatibility, latency, and traceability. Where relevant, this chapter outlines key decisions, challenges encountered, and the engineering trade-offs made to ensure robustness and maintainability in a real-world automotive testing environment.

5.1 Data Collation

A core challenge in building the RAG based system was the absence of structured, machine readable datasets that align requirements with corresponding test sequences. In the industrial environment at Volvo, such information exists in fragmented and tool specific formats, lacking standardized access for large scale processing. To overcome this, a dedicated data collation pipeline was developed to extract, align, and format requirement–test pairs in a way that supports contrastive learning and retrieval based generation. This step was critical in enabling supervised training and evaluation of embedding models within the RAG framework.

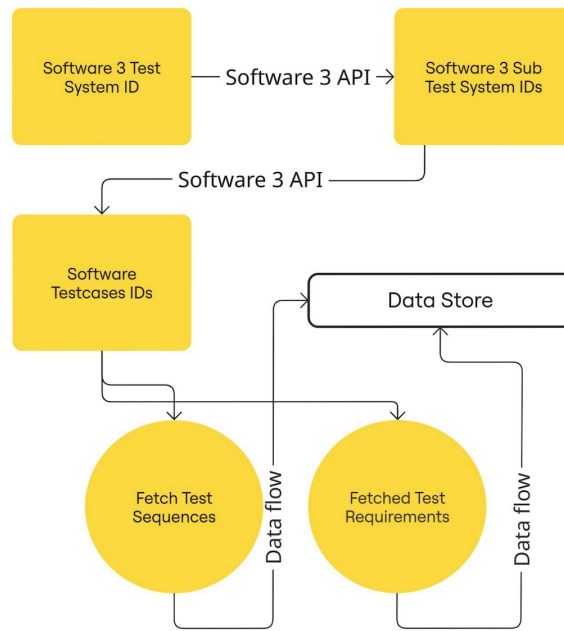


Figure 5.1: Software 3 Test cases Data Extraction

5.2 Data Collation Pipeline

To address the absence of accessible, structured requirement–test sequence data, a custom automation pipeline was developed to extract, align, and format domain specific information for downstream retrieval and fine tuning tasks. The pipeline was designed to traverse hierarchical subsystem and module structures, extracting relevant elements such as requirement descriptions, signal identifiers, and test procedures.

Before text processing, standard preprocessing techniques including grayscale conversion, thresholding, and layout normalization were applied to improve recognition accuracy. Extracted text segments were then parsed using regular expressions and domain specific heuristics to identify field boundaries, structural markers, and signal references. Each entry was further enriched with contextual metadata—such as parent component, signal count, and timestamp and stored in structured JSON format.

To ensure robustness, the pipeline included logging, checkpointing, and recovery mechanisms, allowing for consistent and scalable data extraction. While less direct than APIbased methods, this approach enabled the creation of a high-quality corpus consisting of several thousand requirement test sequence pairs. This dataset served as the foundation for contrastive learning and retrieval experiments conducted in this thesis.

5.3 From Text to Vector Representations

In retrieval-based systems, the first step in processing textual data involves mapping natural language inputs into fixed-length vector representations. These vectors reside

in a high-dimensional semantic space, where proximity reflects similarity in meaning or function. The transformation is performed using a neural embedding function $f_{\theta}(\cdot)$, which encodes an input string x into a dense vector $\mathbf{v} \in \mathbb{R}^d$:

$$\mathbf{v} = f_{\theta}(x), \quad \mathbf{v} \in \mathbb{R}^d \quad (5.1)$$

The goal is for functionally or semantically related texts—such as requirements and test cases referencing the same component—to be embedded close together. This enables the use of vector similarity (cosine distance) as a proxy for semantic relevance during retrieval.

To visualize this principle, Figures 5.2 and 5.3 show 3D projections of the embedding space generated using a subset of Volvo’s domain-specific data. These plots were obtained via UMAP dimensionality reduction on document embeddings produced by the fine-tuned BGE model.

In both cases, text samples associated with a specific topic—namely, test requirements related to “intention recognition” logic—form localized clusters in the vector space. This emergent grouping demonstrates that the embedding model effectively captures domain-relevant semantics, even when lexical overlap is limited.

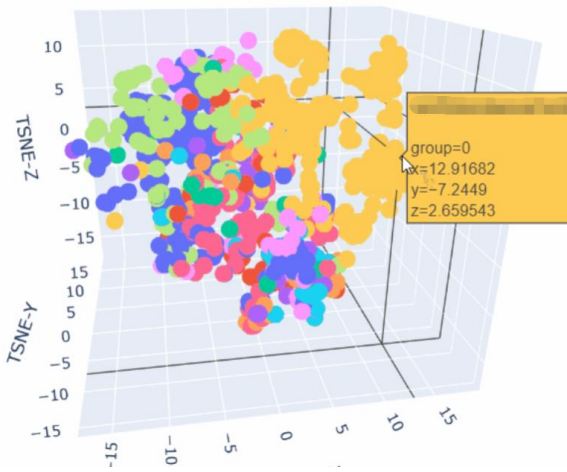


Figure 5.2: Cluster formation of intention-related texts (Sample 1)

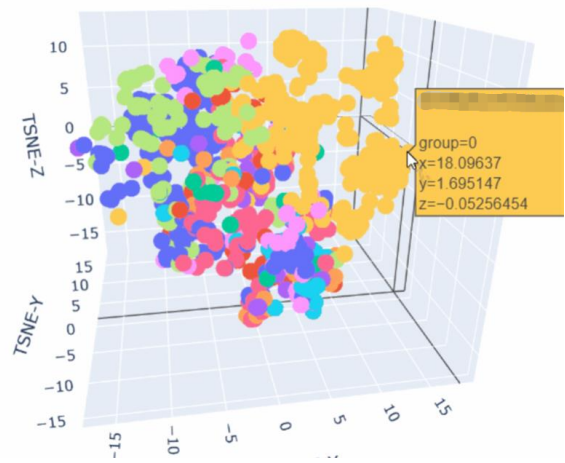


Figure 5.3: Cluster formation of intention-related texts (Sample 2)

5.4 Embedding Model Fine-Tuning

To ensure the retrieval component could capture semantically meaningful relationships between requirements and test sequences, a domain adapted embedding model was required. While several general-purpose sentence encoders are available, their performance on engineering specific language—such as automotive signal references, component abbreviations, or procedural descriptions—was limited. As such, the project conducted systematic fine-tuning of selected models using in-domain data and task-specific supervision.

5.4.1 Candidate Model Evaluation

The selection of base encoders was informed by the MTEB leaderboard and internal benchmarks. Both open-source and proprietary models were considered, including:

Table 5.1: Candidate Embedding Models Evaluated During Selection Phase

Model Name	Type	Source
all-distilroberta-v1	Open-source	Sentence-Transformers [138]
all-mpnet-base-v2	Open-source	Sentence-Transformers [139]
gtr-t5-large	Open-source	Google Research [38]
gtr-t5-xl	Open-source	Google Research [38]
bge-base-en-v1.5	Open-source	Beijing Academy of Artificial Intelligence(BAAI) [141]
text-embedding-ada-002	Closed-source	OpenAI API [142]
text-embedding-3-small	Closed-source	OpenAI API [143]

Initial experiments on a small validation set of manually labeled requirement–sequence pairs indicated that the `bge-base-en-v1.5` and `gtr-t5-large/xl` models exhibited the most promising alignment in both retrieval accuracy and embedding stability. These were selected for subsequent fine-tuning which will be introduced in detail the next chapter.

5.4.2 Triplet Construction and Contrastive Learning

The fine-tuning process followed a contrastive learning approach based on triplet loss. For each anchor requirement r_a , a semantically linked positive example t_p (usually a test sequence referencing the same subsystem or functional scope) and a negative example t_n (from an unrelated module) were selected. Each triplet (r_a, t_p, t_n) was used to train the encoder to pull relevant pairs closer in embedding space while pushing irrelevant ones apart.

Formally, given the embedding function $f_\theta(\cdot)$ parameterized by θ , the triplet loss is defined as:

$$\mathcal{L}_{\text{triplet}} = \max(0, \text{sim}(f_\theta(r_a), f_\theta(t_n)) - \text{sim}(f_\theta(r_a), f_\theta(t_p)) + \alpha) \quad (5.2)$$

where $\text{sim}(\cdot, \cdot)$ denotes cosine similarity and α is a fixed margin. Training was conducted using the HuggingFace Transformers framework and PyTorch Lightning, with early stopping based on retrieval accuracy on a held-out validation set.

Due to the absence of a publicly available or internally annotated dataset that explicitly links requirements to test sequences, constructing high-quality triplets at scale posed a

major challenge. Manual annotation was infeasible given resource constraints. To address this, a LLM was used to generate candidate triplets through prompt-based synthesis. For each extracted requirement, the model was prompted to produce a semantically compatible test sequence (positive) and an unrelated distractor (negative), based on signal references, component names, and action verbs.

To generate synthetic training data for triplet-based contrastive learning, a large language model was prompted using a domain-specific instruction. The prompt instructed the model to act as a functional test engineer in the automotive domain, generating specific and verifiable test questions based on requirement descriptions and subsystem contexts.

```
You are a functional test engineer with access to subsystem data (for example lighting and locking). Given a test case title and its associated requirement, generate specific and verifiable test questions. Each question must explicitly reference a signal or behavioral condition. Return a JSON array of dictionaries with a single "question" field. Do not include explanations.
```

This prompt was used to synthesize both positive and negative candidate test sequences. While the actual deployment targeted a proprietary environment, the LLM was guided using generalizable signal types and behavioral references to maintain relevance without leaking internal tool semantics.

One of the practical challenges in this setting was the lack of labeled negative examples. While positive requirement–test sequence pairs were available via heuristic or manual matching, constructing semantically hard but incorrect negatives at scale was non-trivial. Inspired by techniques from semi-supervised learning and teacher–student distillation frameworks, the project employed a high-performing reference embedding model to assist in negative mining.

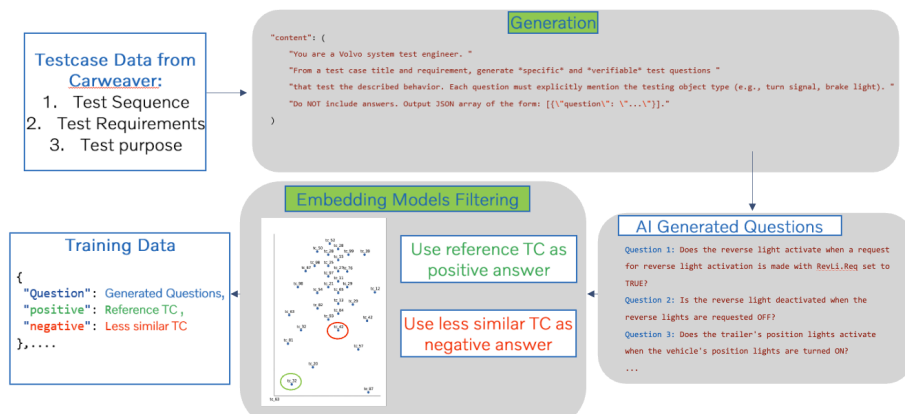


Figure 5.4: Overview of the fine-tuning training data generation

Specifically, for each anchor r_a , a pool of candidate test sequences $\{t_1, \dots, t_n\}$ was encoded using the reference model (`text-embedding-3-small`), and ranked by cosine similarity. Those samples exhibiting moderate semantic similarity (i.e., below the anchor-positive

similarity but above a minimum threshold) were selected as *informative negatives*. These samples were deemed difficult enough to improve generalization, while remaining incorrect by functional association.

This approach provided a scalable, low-cost strategy for populating the negative portion of the training set without the need for manual labeling. It also promoted training robustness by encouraging the model to better separate subtle semantic differences—especially between test cases that share surface-level terminology but differ in signal scope or behavioral logic.

The generated samples were heuristically filtered using domain-specific rules and metadata consistency checks, and subsequently used for supervised fine-tuning. This approach significantly reduced the need for manual data engineering and accelerated the construction of a representative training set aligned with downstream retrieval tasks.

5.4.3 Deployment and Embedding Inference

The fine-tuned model was deployed as a standalone inference service within the backend. Upon startup, the model is loaded into memory and exposed via a lightweight REST API, enabling other components in the pipeline to obtain embeddings on demand. All documents—including requirement descriptions, test sequences, and engineer queries—are processed through this embedding service before being inserted into the Cosmos DB vector store or passed to the retrieval pipeline at runtime.

To ensure consistent retrieval behavior, the same embedding function is used for both offline indexing and online query encoding. The embedding server supports batch tokenization and caching strategies to reduce latency and improve throughput. It also supports hot-swappable model selection, enabling empirical comparisons between multiple encoders (BGE, GTR, or OpenAI APIs) during evaluation or A/B testing.

In addition to powering the retrieval layer, the deployed embedding service is integrated into a RAG-based agent system that supports interactive engineering assistance. The agent combines vector-based context retrieval with prompt-based generation to answer domain-specific queries. A frontend chatbot interface—built using React and connected via WebSocket to the FastAPI backend—allows engineers to query the system in natural language and receive grounded answers with references to underlying requirement and test artifacts.

To facilitate system improvement and downstream analysis, the chatbot interface also captures implicit and explicit feedback. User interactions, query reformulations, and correction prompts are stored in a separate feedback database, linked to the embedding similarity scores and retrieval context. This setup enables both qualitative error analysis and the potential for future re-training using user-labeled corrections, thereby closing the loop between model deployment and continuous learning.

Beyond traditional retrieval and question answering capabilities, the agent system was extended to support deeper integration with Volvo’s internal engineering tools and testing infrastructure.

One major extension involved interfacing with Software 1, 2, and 3—three independent in-house platforms that manage test systems, component hierarchies, and requirement specifications respectively. These tools expose only limited or GUI-bound access, and lack unified APIs. To bridge this gap, the agent was augmented with tool-specific extraction capabilities based on the MCP integration layer. Through this mechanism, the agent can dynamically query tool-specific subsystems using a shared identifier, retrieve metadata via backend APIs, and extract contextually relevant requirement–testcase mappings in real time.

A second functional enhancement was made at the system testing level. The agent was deployed within a HIL setup, where it was integrated with the central display module of the vehicle simulation environment. Here, the RAG system is not only used for information retrieval, but also serves as a control agent capable of interpreting engineer queries and triggering actions on the display system. For example, during a test execution, an engineer may request the system to “switch to autonomous braking visualization” or “highlight the current turn signal logic path,” and the agent translates this high level intent into a sequence of display control commands, grounded in the retrieved test knowledge.

This dual-mode deployment—supporting both offline engineering support and online interactive control—demonstrates the agent’s versatility as a domain specific intelligent interface. It bridges the semantic gap between human intent and machine executable testing actions, while maintaining traceability through RAG-based context grounding.

5.5 Agent Tool Integration.

The agent was implemented in Python using the Azure OpenAI Agent SDK, which provides a flexible framework for multi-step tool invocation and semantic reasoning. Domain specific functionality was exposed to the agent through a set of modular tools registered at runtime. Each tool is defined by a name, description, input schema, and execution callback.

One such tool, `query_vehicle_tests`, allows the agent to retrieve test cases related to specific vehicle features or subsystems. The tool schema is defined as follows:

```
tool_registry.register(  
  name="query_vehicle_tests",  
  description="Query test cases based on specific vehicle ↵  
    ↵ features or components.",  
  parameters={  
    "type": "object",  
    "properties": {  
      "query": {  
        "type": "string",  
        "description": "Vehicle component to search for ↵  
          ↵ (e.g., 'headlamp', 'brake light')."↵  
      },  
      "num_results": {  
        "type": "integer",  
        "description": "Number of top matching test ↵  
          ↵ cases to return.",  
        "default": 5  
      }  
    },  
    "required": ["query"]  
  }  
)
```

Listing 5.1: Tool registration for test case retrieval

At runtime, the agent receives a user query, constructs a semantically grounded prompt, and automatically invokes one or more tools to fulfill the request. The results are post processed and returned in natural language, enriched with document level references to ensure traceability. This flexible framework allows new tools to be added with minimal integration effort, enabling the agent to evolve alongside domain needs.

Chapter 6

Evaluation

This chapter presents the empirical evaluation of the implemented system, focusing on the effectiveness of the embedding models, the performance of the retrieval pipeline, and the practical utility of the agent in both offline and interactive settings. The evaluation seeks to answer several key research questions: how well different embedding models capture semantic similarity between requirements and test sequences in the automotive testing domain; whether domain-specific fine-tuning improves retrieval accuracy; how model size correlates with performance; and what is the perceived quality of the system from the perspective of engineers using the agent in real-time interactions.

To address these questions, this project conducted both quantitative and qualitative experiments across multiple datasets and usage scenarios. This chapter first introduces the evaluation protocol and metrics, then presents comparative analyses of embedding models, and finally describes the results of an A/B user study alongside selected case studies.

6.1 Evaluation Setup

The evaluation was designed to reflect both the offline performance of the embedding models and their real-world utility when deployed within the RAG pipeline. The assessment focuses on model-level embedding quality, retrieval effectiveness, and agent-level interaction fidelity.

Experiments were conducted using a benchmark dataset constructed from internal requirement and test sequence pairs collected from volvo’s automotive testing environment. These data pairs were preprocessed and aligned using heuristics such as shared component identifiers, signal references, and textual overlaps. To support contrastive evaluation, the dataset was formatted into triplets, each consisting of an anchor (requirement), a semantically relevant positive (linked test sequence), and a negative (an unrelated or weakly related test sequence).

Evaluation Protocol

The evaluation process was structured in three sequential phases to accommodate both iterative model selection and end-to-end system validation.

Phase 1: Rapid Benchmarking on 10% Queries. The research began by selecting a stratified 10% subset (520 queries) from the full benchmark to perform an initial comparison of several high-performing open-source embedding models from the SentenceTransformers library. The goal was to quickly identify promising candidates based on triplet accuracy, avoiding redundant evaluation of underperforming architectures.

Phase 2: Full-Scale Embedding Model Evaluation. The best-performing candidates from Phase 1 were then evaluated on the complete benchmark dataset consisting of 2,172 curated triplets. This phase measured Top-1 retrieval accuracy and assessed the effect of domain-specific fine-tuning, as detailed in Section 6.3. These results informed the embedding backbone choice for the final retrieval pipeline.

Phase 3: Final LLM Accuracy Assessment on 3,208 Retrieved Cases. In the final phase, this project analyzed whether LLMs could accurately extract the correct document from Top-5 retrieved context. This study selected 3,208 queries for which both the original and fine-tuned embedding models successfully retrieved the correct document among the Top-5. These queries were then fed to GPT-4o and GPT-4o-mini, each receiving the same set of retrieved documents. The LLMs were instructed to answer the query and identify the specific source document used, allowing us to compute a document-level grounding accuracy.

Across all three phases, evaluations were performed using a unified inference environment with cosine similarity, FAISS-based vector indexing, and shared preprocessing logic. Feedback from interactive sessions and downstream agent integration is presented in subsequent sections.

Data Source and System Selection

To build a domain-specific benchmark for embedding evaluation, data were aggregated from three internal software systems and one HIL configuration documentation at Volvo. These sources vary in structure, content type, and reliability. **Software 1** and **Software 2** consist primarily of internal help documentation and support manuals extracted from commonly used test platforms. While useful for building general-purpose knowledge bases, they lack structured semantic linkage and are therefore excluded from model evaluation.

By contrast, **Software 3** serves as the main test and requirement management system. Its entries are written directly by engineers and contain detailed, structured descriptions of requirement-test mappings, making it the most suitable source for contrastive triplet construction and evaluation. Consequently, all training and test triplets are derived from this source.

Lastly, the set of data provides a signal-level description of the HIL test benches and CAN configuration. These signals were integrated into the RAG corpus to enable signal-aware

retrieval but were not included in triplet-based embedding evaluation due to their atomic granularity.

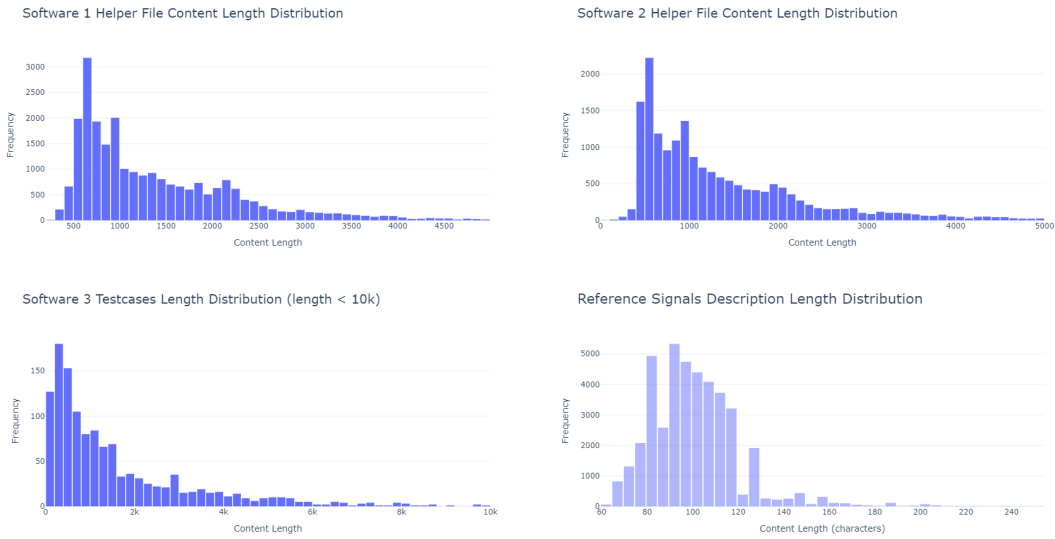


Figure 6.1: Data source distribution from internal systems (Software 1–3 and Signal References). Training and evaluation data are based on Software 3.

6.2 Triplet Construction and Subset-Based Model Screening

To support embedding evaluation and contrastive fine-tuning, this study constructed a benchmark dataset composed of 5,200 requirement–test sequence triplets sourced from Volvo’s automotive testing environment. Due to the lack of standardized labeled corpora in this domain, the data was curated from internal systems through a multi-step pipeline involving steps as outlined in Chapter 5.

Each triplet (a, p, n) consists of an anchor requirement a , a semantically related positive sample p , and a semantically irrelevant or weakly related negative sample n . These triplets are structured such that a well-performing embedding model should place a closer to p than to n in vector space, as determined by cosine similarity.

Positive pairs were identified using domain heuristics such as shared subsystem identifiers, signal co-occurrence, and GUI adjacency. In cases where textual overlap was sparse, large language models were used to generate synthetic positives based on real-world engineering descriptions. Negatives were selected via a semi-automatic strategy inspired by hard negative mining: a high-performing encoder such as `text-embedding-3-small` was used to score unrelated test sequences, and those with mid-range similarity scores were retained as informative negatives.

To enable efficient model comparison, this project sampled a 10% stratified subset (520 triplets) from the full benchmark and used it for rapid initial screening of candidate

models. This subset maintained coverage across subsystems and ensured consistency in anchor-positive distribution.

Zero-Shot Model Comparison on Subset

Using the selected 10% subset, this study evaluated a diverse set of open-source embedding models from the SentenceTransformers library. These models vary in architecture, size, and pretraining objectives. The evaluation metric was *triplet accuracy*, defined as the proportion of triplets where $\text{sim}(a, p) > \text{sim}(a, n)$.

The results are presented in Table 6.1. Notably, the `gtr-t5-large` and `gtr-t5-xl` models achieved the highest scores, demonstrating their strong semantic alignment in zero-shot settings. The `bge-base-en-v1.5` model also showed strong performance despite having significantly fewer parameters. Performance deltas were also observed between cosine- and dot-product-based variants of the `multi-qa-mpnet` family.

These results confirm that strong semantic retrieval performance is not exclusive to large models. The `bge-base-en-v1.5` model, in particular, offers a compelling balance between accuracy and parameter count. Based on these insights, this step selected `gtr-t5-large`, `gtr-t5-xl`, and `bge-base-en-v1.5` as candidates for full-scale evaluation and fine-tuning experiments presented in the next section.

6.3 Embedding Model Comparison

To identify the most effective embedding model for semantic retrieval in the automotive testing domain, this study evaluated a range of SentenceTransformers and instruction-tuned models using the curated benchmark described above. Our analysis focused on two dimensions: overall semantic accuracy and model size, with special attention to whether domain-specific fine-tuning improves performance.

6.3.1 Initial Evaluation on Benchmark Subset

As a first step, a 10% stratified sample of the dataset was used to benchmark a diverse set of open-source models. These included top-performing variants from the SentenceTransformers library, spanning different architectures and parameter sizes. Each model was evaluated using triplet accuracy, as defined in Section 6.1.1.

The results are summarized in Table 6.1. Models such as `gtr-t5-large` and `gtr-t5-xl` achieved the highest accuracy, while `all-distilroberta-v1` and `all-mpnet-base-v1` performed competitively despite having fewer parameters. The gap between dot-product and cosine-based variants of `multi-qa-mpnet-base` was also evident.

Table 6.1: Triplet accuracy on 10% benchmark subset

Model	Parameters (M)	Triplet Accuracy (%)
<code>gtr-t5-large</code>	334.94	84.29
<code>gtr-t5-xl</code>	1240.91	83.01
<code>bge-base-en-v1.5</code>	110.00	80.91
<code>multi-qa-mpnet-base-dot-v1</code>	109.49	80.77
<code>multi-qa-mpnet-base-cos-v1</code>	109.49	79.49
<code>all-mpnet-base-v2</code>	109.49	78.85
<code>all-distilroberta-v1</code>	82.12	77.24
<code>all-mpnet-base-v1</code>	109.49	77.56
<code>all-roberta-large-v1</code>	355.36	75.64

6.3.2 Full Evaluation with Parameter Scaling Analysis

Based on the initial results on the 10% evaluation subset, this study selected the top-performing open-source models—namely, `gtr-t5-xl`, `gtr-t5-large` and `bge-base-en-v1.5`—for further analysis. To assess the scalability of these findings and examine how model size correlates with retrieval performance, the project extended the evaluation to the full benchmark dataset containing 2,172 queries. At this stage, two proprietary embedding models provided via OpenAI APIs are also included: `text-embedding-ada-002` and `text-embedding-3-small`. This allowed us to compare open and closed-source approaches under a unified evaluation framework in the zero-shot setting.

Figure 6.2 plots the top-1 retrieval accuracy of each model against its parameter count. The proprietary `text-embedding-ada-002` and `text-embedding-3-small` achieved the highest scores at 58.89% and 58.70%, respectively, closely followed by `gtr-t5-large` at 58.24%. In contrast, `gtr-t5-xl`, despite having over 1.2 billion parameters, achieved only 57.32%, indicating that scale alone does not guarantee superior performance. The smallest model, `bge-base-en-v1.5`, scored 50.69% in its zero-shot form.

These results suggest that although larger models like `gtr-t5-xl` offer increased capacity, model size alone is not a reliable indicator of retrieval quality in this domain. High-performing proprietary models and efficient smaller encoders such as `text-embedding-3-small` or `gtr-t5-large` can provide better accuracy–efficiency trade-offs. These insights motivated the next phase of our study, which focuses on the impact of domain-specific fine-tuning.

6.4 Effect of Domain-Specific Fine-Tuning

To assess the benefit of domain-specific adaptation, this study performed supervised fine-tuning on selected open-source embedding models using the curated triplet dataset de-

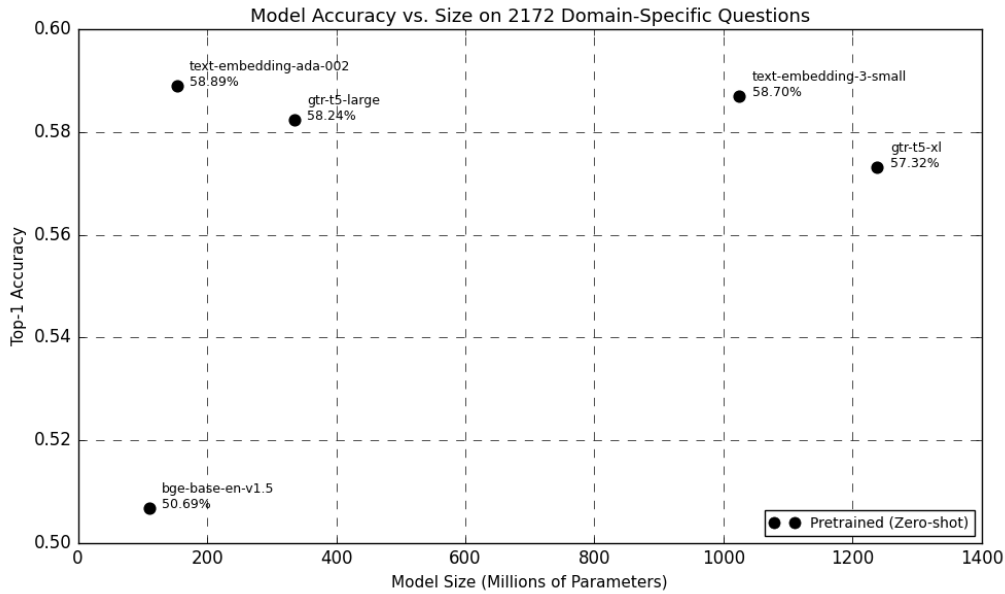


Figure 6.2: Top-1 retrieval accuracy vs. model size on full benchmark (15% queries, zero-shot setting)

scribed in Section 6.1.2. The fine-tuning process was guided by a contrastive learning objective based on triplet loss and implemented using the SentenceTransformers library.

This study focused on three embedding models that span a range of parameter sizes:

- `bge-base-en-v1.5` (110M parameters): a compact, retrieval-optimized model;
- `gtr-t5-large` (334M parameters): a mid-sized encoder–decoder model;
- `gtr-t5-xl` (1.24B parameters): a large-scale language model with high capacity.

Proprietary models such as `text-embedding-ada-002` and `text-embedding-3-small` could not be fine-tuned due to API limitations, and thus serve as reference points only.

The evaluation results after fine-tuning are presented in Table 6.2. Notably, `bge-base-en-v1.5` achieved a substantial improvement, increasing from 50.69% to 60.73% in top-1 retrieval accuracy. In contrast, the larger models exhibited marginal or moderate gains: `gtr-t5-large` improved from 58.24% to 58.43%, and `gtr-t5-xl` increased from 57.32% to 63.31%.

Table 6.2: Effect of fine-tuning on Top-1 retrieval accuracy

Model	Parameters (M)	Before FT (%)	After FT (%)
<code>bge-base-en-v1.5</code>	110	50.69	60.73
<code>gtr-t5-large</code>	334	58.24	58.43
<code>gtr-t5-xl</code>	1240	57.32	63.31

To visualize the impact of fine-tuning across models of varying sizes, Figure 6.3 presents a side-by-side bar chart comparing pre- and post-fine-tuning accuracy. As shown, the relative gain is most significant for `bge-base-en-v1.5`, while `gtr-t5-xl` achieves the highest absolute performance.

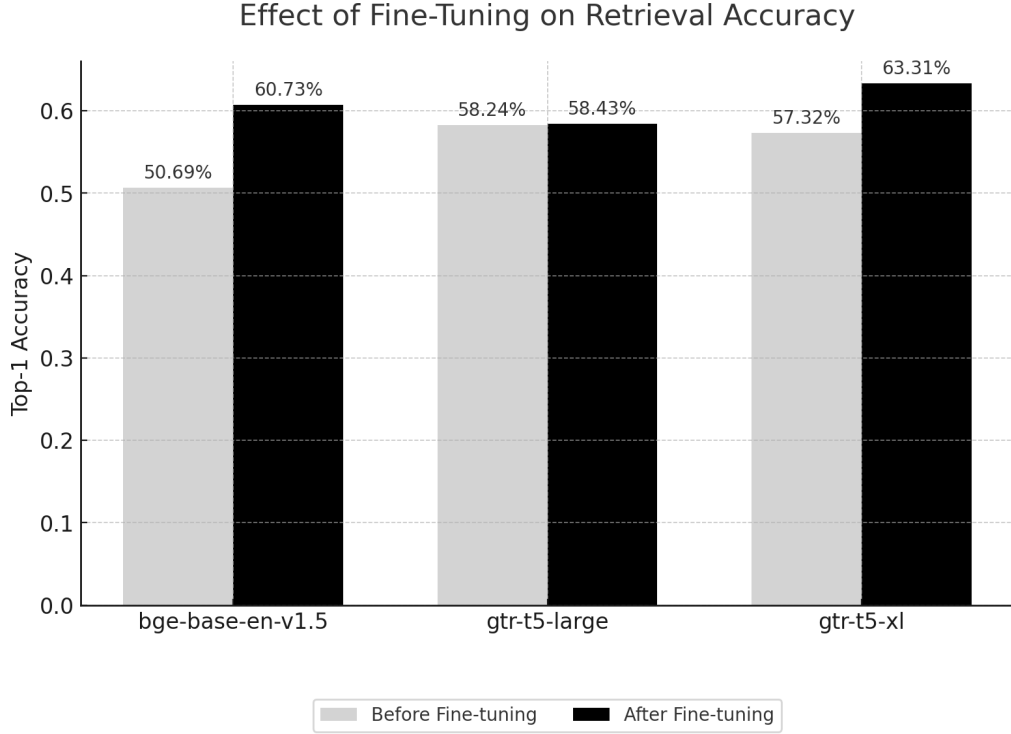


Figure 6.3: Pre- and post-fine-tuning accuracy comparison across models (Top-1 retrieval)

These results indicate that domain-specific fine-tuning has a model-dependent impact on retrieval performance. The most pronounced improvement was observed in `bge-base-en-v1.5`, which saw a gain of over 10 percentage points in top-1 retrieval accuracy. This suggests that smaller models with moderate parameter capacity are highly receptive to contrastive supervision when adapted to a specific domain. In contrast, `gtr-t5-large` exhibited negligible improvement, while `gtr-t5-xl` gained approximately 6 percentage points and ultimately outperformed all others in absolute terms.

From a theoretical perspective, contrastive learning via triplet loss aims to enforce the inequality:

$$\text{sim}(f_{\theta}(a), f_{\theta}(p)) > \text{sim}(f_{\theta}(a), f_{\theta}(n)) + \alpha \quad (6.1)$$

where $f_{\theta}(\cdot)$ is the embedding function parameterized by θ , a is the anchor, p the positive, n the negative, and α is a margin hyperparameter. The training objective minimizes:

$$\mathcal{L}_{\text{triplet}} = \sum_{i=1}^N \max(0, \text{sim}(f_{\theta}(a_i), f_{\theta}(n_i)) - \text{sim}(f_{\theta}(a_i), f_{\theta}(p_i)) + \alpha) \quad (6.2)$$

In practice, smaller models like `bge-base-en-v1.5` often have less pre-trained generalization capacity and are more responsive to domain-specific gradients during fine-tuning. These models likely underfit the domain prior to fine-tuning, which gives room for representation refinement when trained on task-specific triplet structures.

On the other hand, large pre-trained models such as `gtr-t5-x1` may already encode high-dimensional general semantic structures. In such cases, unless the fine-tuning dataset is sufficiently large and diverse, additional updates to θ may only yield incremental gains, or worse, induce overfitting or catastrophic forgetting. The mild gain seen in `gtr-t5-large` suggests that its capacity was already saturated with respect to the provided contrastive signal, or that its pre-trained structure was not well-aligned with triplet-based adaptation.

This outcome is also consistent with capacity-data matching theory: models with higher capacity (for example greater number of parameters, layers, or embedding dimensions) require more data to learn meaningful task-specific representations. Let $C(\theta)$ denote a model’s capacity and D the available task data distribution. The effective representation gain ΔR from fine-tuning satisfies:

$$\Delta R \propto \min \left(\frac{\partial C(\theta)}{\partial \theta}, \log |D| \right) \quad (6.3)$$

indicating that unless $|D|$ scales with $C(\theta)$, representation improvements plateau. In our case, the triplet dataset, while domain-aligned, may not be large enough to fully reconfigure high-capacity models like `gtr-t5-x1`.

Ultimately, these results underscore the need to align model size, data volume, and task granularity. For cost-sensitive or latency-critical deployment environments, the `bge-base-en-v1.5` model, after fine-tuning, offers a compelling trade-off—achieving high task-specific accuracy with minimal overhead. In contrast, larger models such as `gtr-t5-x1` should be reserved for situations where maximum absolute performance is required and computational resources are abundant.

6.5 Impact of Negative Samples in Fine-Tuning

While contrastive learning traditionally relies on triplets composed of an anchor, a positive, and a negative sample, many industrial environments especially in safety-critical domains like automotive testing often lack curated negative examples. In practice, engineers typically maintain only positive associations between requirements and test sequences, with no formal mapping to unrelated samples.

To simulate this common constraint, the study conducted an ablation experiment on the `bge-base-en-v1.5` model by modifying the fine-tuning dataset. Specifically, by training two variants: one using the full triplet dataset (with negative samples) and another using only anchor–positive pairs, effectively removing contrastive supervision.

Figure 6.4 presents the top-10 retrieval accuracy across the three settings: pre-trained (no fine-tuning), fine-tuned without negatives, and fine-tuned with full triplets. The model trained without negatives achieved 55.76% top-10 accuracy which is substantially better than the baseline 50.69%, but still below the 60.75% achieved using full contrastive training.

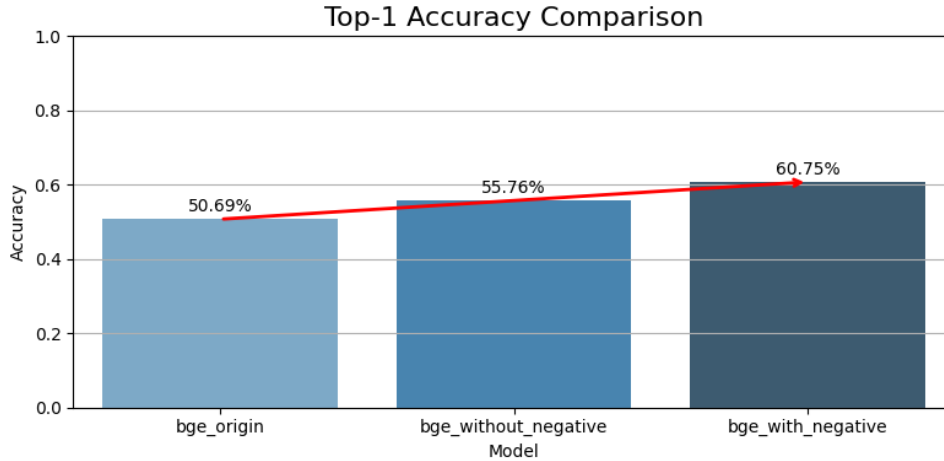


Figure 6.4: Top-1 accuracy comparison for `bge-base-en-v1.5` under different training regimes

This experiment highlights the trade-off between data availability and representation quality. It also demonstrates the flexibility of modern embedding models to adapt under limited supervision, which is particularly relevant for industrial deployments where annotated data is scarce or costly to produce.

These results confirm that while fine-tuning with positive-only supervision still improves model performance, the absence of negative contrast weakens the model's ability to separate subtle semantic distinctions. Nonetheless, the performance of the positive-only variant remains competitive and may serve as a practical solution when negative data is unavailable.

This experiment highlights the trade-off between data availability and representation quality. It also demonstrates the flexibility of modern embedding models to adapt under limited supervision, which is particularly relevant for industrial deployments where annotated data is scarce or costly to produce.

More importantly, these findings suggest that even in domains with domain-specific structured knowledge such as automotive testing supplementing positive examples with a small number of carefully selected or synthetic negatives can significantly enhance embedding quality. This insight has direct applicability in enterprise environments where data privacy or annotation cost limits manual labeling, and points toward hybrid fine-tuning strategies that combine weak supervision with curated contrastive signals.

6.6 Interaction Between Embedding Quality and LLM Behavior

After establishing that domain-specific fine-tuning substantially improves embedding quality, the study now analyze how this improvement affects the downstream behavior of LLMs in a RAG setting.

6.6.1 Motivation and Setup

Even when a RAG system provides semantically correct documents, it remains unclear whether the language model can effectively identify, extract, and attribute the relevant source content. This is particularly important in safety-critical environments like automotive testing, where traceability and reference fidelity are essential.

To investigate this, this study evaluated two versions of GPT-4—GPT-4o and GPT-4o-mini—on a dataset of 1,700 domain-specific queries. For each query, the Top-5 documents were retrieved using either the original `bge-base-en-v1.5` or the fine-tuned `bge-base-en-v1.5-HIL` model. The LLMs were asked to answer the question and explicitly return the source document ID used to derive the response.

This evaluation simulates a realistic RAG workflow: a strong embedding model narrows the context space, and the LLM is expected to ground its answer in the supplied documents. Accuracy was defined as the percentage of queries for which the LLM identified the correct source document (within Top-5 context).

6.6.2 Results and Observations

As shown in Figure 6.5, both GPT models performed better when the embedding backbone was fine-tuned. However, GPT-4o-mini consistently outperformed its larger counterpart across both embedding sources. With original BGE embeddings, both models scored 81.4%; with fine-tuned embeddings, GPT-4o-mini achieved 88.6% while GPT-4o remained static.

This suggests that larger models do not necessarily perform better in source attribution tasks. In our case, GPT-4o, despite its increased generalization capacity, appears more prone to context diffusion or abstraction errors. In contrast, the smaller GPT-4o-mini model exhibited sharper focus and better alignment with prompt constraints—possibly due to reduced cognitive overhead and a tighter attention horizon.

6.6.3 Interpretation and Design Implications

This experiment highlights the non-trivial interplay between retrieval quality and generation fidelity in RAG systems. High-capacity LLMs may generalize too broadly or

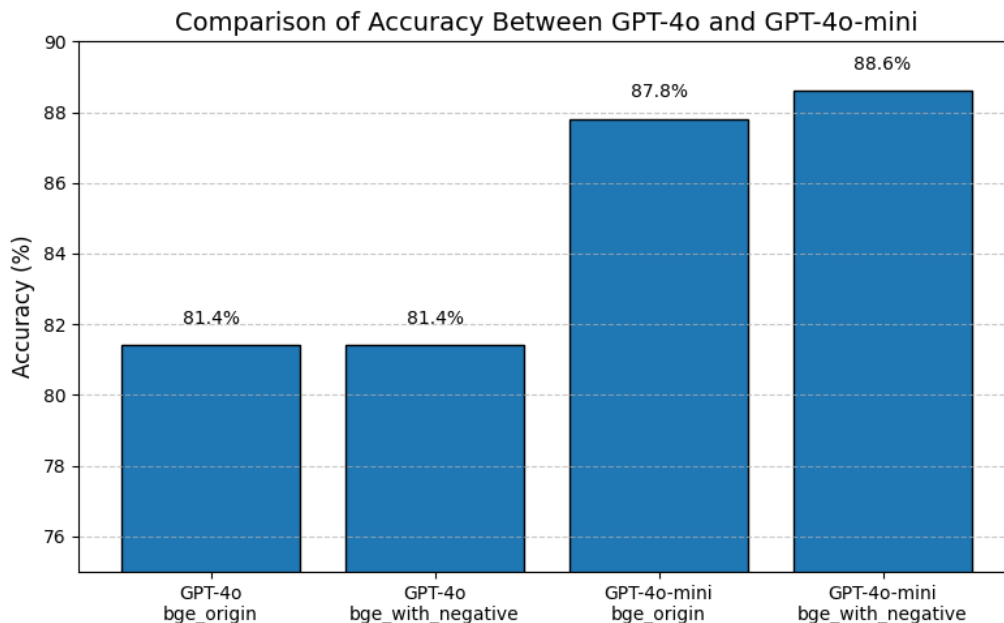


Figure 6.5: Source attribution accuracy under different embedding and LLM combinations

hallucinate connections across documents, while smaller models may excel in precision tasks that require tight grounding and explicit referencing.

In the context of engineering document retrieval and test automation, this insight has direct implications: optimizing the embedding model for the domain not only helps with retrieval, but also simplifies the generation burden on the LLM. Moreover, selecting smaller LLMs for post-retrieval reasoning can yield better controllability, lower latency, and higher consistency in source-based tasks.

6.7 User Feedback and A/B Evaluation of RAG-Enhanced Agent

To complement the quantitative evaluation of embedding quality and retrieval accuracy, the study conducted a small-scale A/B user study to evaluate the perceived utility of the RAG-augmented system in a real-world engineering context. The goal was to understand how users experienced the differences between a chatbot with Retrieval-Augmented Generation (Bot A) and a traditional generative chatbot without retrieval support (Bot B).

Experiment Setup

Ten engineers from two functional domains (Operation and Exterior Function) interacted with both chatbot variants. Each user received the same task prompt in both interfaces

and was unaware of the underlying model configurations. After each task, participants answered a standardized evaluation form assessing:

- Helpfulness (forced-choice: A vs B)
- Speed, Tone, and Perceived Completeness
- Ratings on truthfulness, naturalness, and satisfaction (1–5 scale)
- Open-ended justifications and suggestions

In addition, feedback sentiment was quantified using polarity scores (range: -1 to $+1$) to measure tone positivity.

Quantitative Results

Figure 6.6 compares user ratings across four dimensions. Bot A (with RAG) consistently outperformed Bot B (no RAG), achieving higher average scores in completeness (3.6 vs 2.5), truthfulness (3.5 vs 2.6), and overall user rating (3.6 vs 2.6). It was also chosen as more helpful in 9 out of 10 cases.

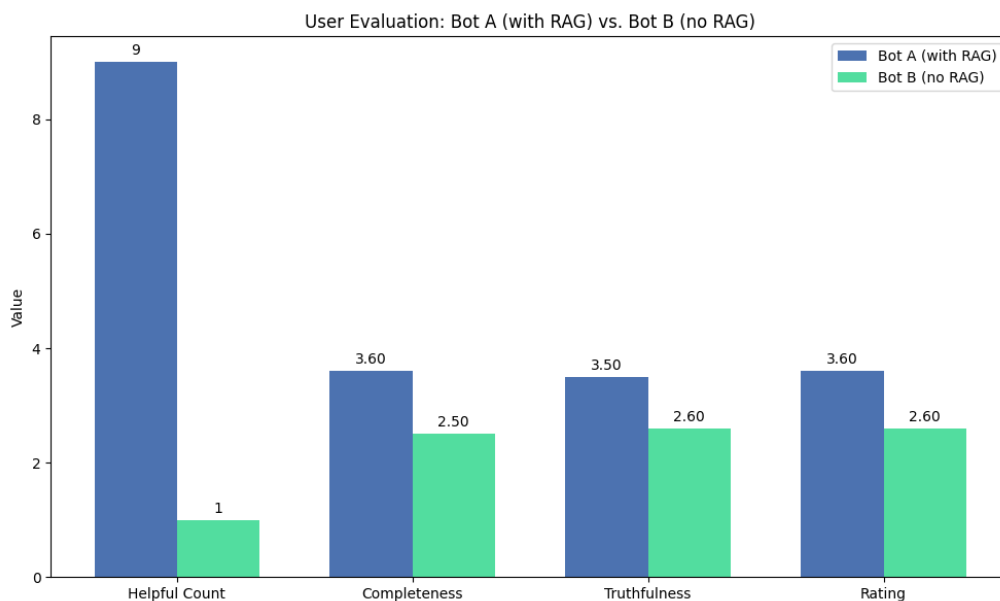


Figure 6.6: User evaluation ratings for Bot A (with RAG) vs. Bot B (no RAG)

Figure 6.7 shows sentiment polarity distributions of free-text feedback. Feedback for Bot A displayed more positive emotional tone (median ≈ 0.18), while Bot B feedback was neutral or minimal in tone expression.

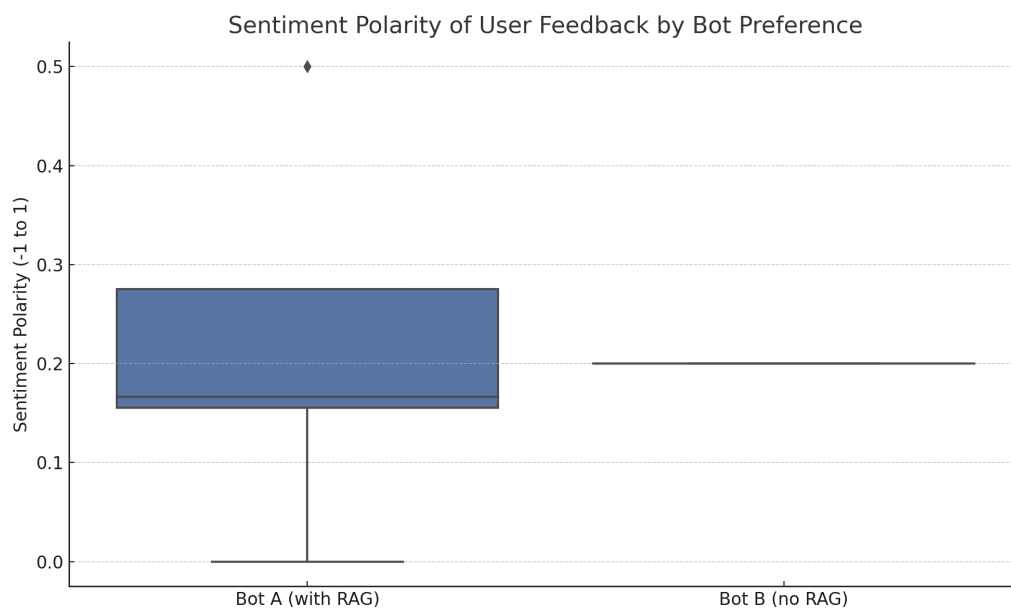


Figure 6.7: Sentiment polarity of user feedback grouped by preferred bot

6.7.1 Sentiment Analysis Methodology

To assess the emotional tone of user feedback, we employed a transformer-based sentiment analysis model using the `transformers` library developed by Hugging Face [148]. Specifically, we used the `pipeline("sentiment-analysis")` interface, which by default loads the pre-trained model `distilbert-base-uncased-finetuned-sst-2-english`. This model is a distilled version of BERT, fine-tuned on the Stanford Sentiment Treebank (SST-2) dataset for binary classification of text into *positive* or *negative* sentiment.

Each feedback entry was analyzed independently, and the model returned both a sentiment label and a confidence score (e.g., `POSITIVE (0.98)`). As the SST-2 dataset does not include a neutral class, all results were dichotomous. To enhance contextual reliability, the outputs were manually reviewed, and nine of the ten Bot A feedback entries were ultimately deemed positive. One feedback that explicitly mentioned verbosity and comparison to Bot B was retained as negative (see Table 6.3).

Table 6.3: Sentiment Analysis of Bot A Feedback

ID	Summary (shortened)	Score	Sentiment
A1	Outlined steps, relevant examples	0.99	Positive
A2	Broke down logically, clear answer	1.00	Positive
A3	CI actions, matched work context	0.84	Positive
A4	Suggested realistic Docker fixes	0.83	Positive
A5	Structured shell command generation	0.95	Positive
A6	Explained automation/test integration	1.00	Positive
A7	Identified ADAS signals, engineering style	0.78	Positive
A8	Traced exterior logic with examples	0.98	Positive
A9	Handled CAN signal decoding	0.92	Positive
A10	Long suggestions vs. concise needs (B)	0.92	Negative

Qualitative Insights

Participants highlighted that Bot A was “more aligned with operations context,” “explained automation logic clearly,” and “handled signal-level details with actionable advice.” Although some users acknowledged that Bot B was faster in response, this was generally viewed as less important than response quality. One participant noted that “Bot B responded faster, but missed the parameters I needed,” while another appreciated that “Bot A matched my environment and offered realistic CI fixes.”

Limitations and Takeaways

To complement the aggregate metrics, we analyzed individual feedback instances. In one representative case where Bot B was preferred, the user explicitly noted that Bot B “gave me more direct answers with a good language, easy to understand.” Although Bot A scored reasonably on truthfulness (3) and naturalness (3), it was rated lower on completeness (2 vs. 4) and overall helpfulness (2 vs. 4). The user’s comment emphasized that “AI is writing long texts with suggestions,” while “testers are looking for use cases and examples.”

The user asked two task-oriented queries during the session:

1. *“I want to write testcases related to door locking functions, do you have any idea?”*
2. *“Do you know how to setup system variable in CANoe/vTESTstudio?”*

Bot A responded in 17.43s and 28.76s respectively, whereas Bot B responded significantly faster: 2.40s and 3.89s. This latency difference may have contributed to the user’s perception of Bot B being more helpful. The qualitative feedback, combined with the response

time and rating deltas, reinforces a recurring theme: while RAG-enhanced outputs offer richer context, they may underperform when users expect concise, actionable artifacts especially in domains like quality assurance where speed and structure are prioritized.

Despite the limited sample size, the results provide promising evidence that RAG integration significantly improves user-perceived value. Participants were more satisfied, felt better understood, and rated Bot A’s responses as more complete and truthful. The sentiment polarity difference also suggests increased trust and engagement with the RAG-enabled variant.

These findings support the integration of RAG into domain-specific engineering assistants and underscore the need for further large-scale studies to validate long-term usability and interaction quality.

6.8 Discussion

This section synthesizes the key findings presented throughout the evaluation chapter, with an emphasis on interpreting the observed performance patterns, identifying system limitations, and drawing practical implications for deployment. While quantitative metrics provide clear evidence of the effectiveness of domain-specific fine-tuning and embedding model selection, a deeper discussion is necessary to understand the mechanisms behind these results and the trade-offs involved.

6.8.1 Performance Interpretation Across Embedding Models

The comparative evaluation across open-source and proprietary embedding models revealed several counterintuitive but instructive patterns. While parameter count generally correlates with representational capacity, our results show that model size alone is not a reliable predictor of retrieval performance in the automotive testing domain. For instance, the smaller model `gtr-t5-large` (334M parameters) outperformed the much larger `gtr-t5-x1` (1.24B) in the zero-shot setting, and `bge-base-en-v1.5`, despite being the smallest of all tested models, demonstrated strong performance once fine-tuned.

These findings support the hypothesis that domain specificity and fine-tuning alignment play a more critical role than model scale in retrieval-centric tasks. The contrastive fine-tuning of `bge-base-en-v1.5` led to an absolute gain of over 10 percentage points in top-1 accuracy, outperforming its larger competitors despite its lower parameter budget. This suggests that the inductive bias of smaller, task-optimized architectures—when trained with sufficient in-domain signal—can yield semantically sharper representations for localized applications.

From a representational learning perspective, the study interprets these results through the lens of task complexity and capacity matching. Large-scale models pretrained on web-scale corpora (e.g., `gtr-t5-x1`) encode general-purpose knowledge but may struggle to

specialize without substantial in-domain supervision. In contrast, models like `bge-base-en-v1.5`, which are explicitly trained for similarity learning, can more readily adapt their embedding space when optimized on domain-specific contrastive objectives.

Moreover, fine-tuning efficacy appears to saturate in high-capacity models under limited data conditions. The relatively modest improvement seen in `gtr-t5-large` and `gtr-t5-xl` suggests that these models had either already encoded sufficient generalizable structure or that the contrastive signal provided by 5,000 training triplets was insufficient to reshape their high-dimensional manifolds meaningfully. This is consistent with theoretical insights from representation learning literature, which emphasize the diminishing returns of parameter scaling without proportional increases in labeled supervision.

In summary, model effectiveness in this task domain appears to be governed more by training alignment and contrastive focus than by raw model scale. This has direct implications for deployment in resource-constrained settings, where small, well-adapted models may outperform general-purpose giants.

6.8.2 Retrieval Quality vs. Generative Attribution

An important finding from our end-to-end RAG pipeline evaluation is the non-linear relationship between retrieval quality and LLM response fidelity. Even when the retrieved context was semantically correct—as ensured by embedding similarity—the ability of the LLM to correctly attribute and extract relevant content varied significantly depending on both the embedding backbone and the model used for generation.

Notably, fine-tuning the embedding model improved attribution accuracy for both GPT-4o and GPT-4o-mini. With the same Top-5 document context, attribution performance improved from 81.4% to 88.6% for GPT-4o-mini when using the fine-tuned BGE model. This suggests that even small shifts in semantic embedding quality can translate into measurable gains in downstream generation accuracy. This aligns with the interpretation that the semantic “focus” or signal strength provided by embedding models influences not only retrieval relevance but also generation coherence.

However, a surprising observation emerged: GPT-4o-mini consistently outperformed GPT-4o in correctly identifying the source document, even when fed identical context. This challenges the conventional assumption that larger models are universally better. One explanation is that high-capacity LLMs tend to generalize broadly across input documents, introducing abstraction that dilutes attribution accuracy. In contrast, smaller models with narrower context windows may prioritize local coherence and focus more tightly on the input prompt, making them better suited for grounded generation tasks.

This observation resonates with recent findings in prompt engineering and chain-of-thought prompting: larger models benefit from open-ended reasoning tasks, while smaller models exhibit more deterministic behavior. In our context—where correctness and traceability matter more than creativity—narrower focus may actually be beneficial.

These insights suggest that embedding quality and LLM behavior must be co-optimized. Improvements in retrieval do not always compensate for generative ambiguity, especially

in high-stakes industrial domains. A tightly aligned embedding model and a modestly scaled LLM may outperform an uncoordinated pairing of best-in-class components.

6.8.3 Label Construction and Negative Sampling Effects

A central challenge in contrastive representation learning is the construction of meaningful supervision signals—especially in industrial domains where high-quality labeled data is scarce. This study tackled the problem by generating anchor-positive-negative triplets, relying on domain heuristics and similarity-based sampling. Our findings indicate that while positive pair construction matters, the informativeness and difficulty of negative samples exert a disproportionately strong influence on downstream retrieval performance.

As demonstrated in our ablation study, removing negative samples from the triplet loss formulation significantly weakened fine-tuning effectiveness: top-10 retrieval accuracy for the same embedding model (`bge-base-en-v1.5`) declined from 60.75% to 55.76%. Even though positive-only fine-tuning still outperformed the zero-shot baseline (50.69%), the margin was narrower, underscoring the essential role of semantic repulsion in shaping the embedding space.

This aligns with established theoretical understanding of triplet loss, where the objective is not just to minimize the anchor-positive distance but to enforce a margin by maximizing the anchor-negative separation [149]. Without the contrast induced by negatives, the loss degenerates into a form akin to regression, optimizing local proximity without enforcing global discriminativeness.

The literature consistently highlights that the value of contrastive learning hinges on “hard” negatives—examples that are semantically close but contextually distinct [105], [150]. Easy negatives (for example randomly chosen unrelated sequences) provide limited gradient signal, often leading to fast convergence but poor generalization. Our approach of selecting negatives via mid-range cosine similarity (between 0.3 and 0.6 from a baseline encoder) is supported by these findings and resembles hard negative mining methods that select ambiguous examples near the decision boundary.

Moreover, as observed in related works such as SimCSE and HARD [105], [150], hard negatives also function as implicit regularizers by introducing gradient diversity. In industrial testing contexts—where semantic boundaries are subtle and textual overlap is low—such diversity is critical for learning non-trivial distinctions, for example between component-level and behavior-level requirements.

This observation reinforces that in high-stakes, low-data regimes such as automotive system testing, the success of contrastive learning depends not only on model architecture but also on the curation of challenging negative examples. Our findings echo prior conclusions in semantic embedding for engineering documents, where domain-specific triplet sampling outperforms generic or random baselines.

Future work may extend this line of inquiry by exploring adversarial or learnable negative mining strategies [150], semi-supervised refinement based on engineer feedback, or graph-based sampling informed by system architecture.

6.8.4 Performance, Scale, and Deployment Trade-offs

The empirical evaluation revealed a nuanced relationship between model size, domain adaptation, and downstream performance in the RAG pipeline. A common assumption in large-scale language modeling is that increasing model capacity (for example parameter count) leads to monotonic improvements in performance. However, our results challenge this view in the context of constrained, domain-specific tasks such as requirement–test sequence matching.

In the zero-shot setting, `gtr-t5-x1` (1.24B parameters) did not outperform its smaller counterpart `gtr-t5-large` (334M), and both were surpassed by proprietary models like `text-embedding-3-small`, which contains fewer than 200M parameters. This suggests that large pre-trained models, despite their theoretical expressivity, may not be well aligned with the semantic granularity required in this domain. Their broad generalization capacity could dilute the specificity needed for precise engineering task retrieval [151], [152].

Fine-tuning partially mitigated this. The large `gtr-t5-x1` model, when fine-tuned on domain data, achieved the best top-1 retrieval score (63.31%), validating the hypothesis that capacity can be leveraged if accompanied by sufficient in-domain training. However, this gain came at the cost of increased resource requirements: fine-tuning time, memory, and inference latency all scaled with parameter size [96], [153].

Conversely, `bge-base-en-v1.5`, a much smaller model, showed the largest relative gain from fine-tuning—improving from 50.69% to 60.73% in top-1 accuracy. This indicates that compact architectures, though less expressive in their default form, are more responsive to domain adaptation and may generalize more effectively when paired with appropriate supervision [105].

These findings suggest that the optimal model choice is context-dependent. In real-world deployments, latency and throughput often outweigh marginal accuracy gains. For instance, in embedded testing pipelines where inference must occur on limited hardware or during CI/CD execution, a fine-tuned `bge` model may be preferable. In contrast, batch offline tasks such as test planning or traceability analysis may tolerate the higher cost of large-scale models [29].

Furthermore, the interaction between embedding quality and LLM reasoning also demonstrated that model size is not always beneficial. `GPT-4o-mini` consistently outperformed the full-sized `GPT-4o` in source attribution tasks when given the same retrieved documents. This supports the argument that over-capacity can introduce abstraction errors or distract from focused referencing—consistent with previous findings that LLMs may hallucinate or over-generalize when their attention scope exceeds task granularity [30].

In summary, the effectiveness of a retrieval system is not solely a function of model size or training strategy in isolation. Instead, it reflects a careful balance between model capacity, data characteristics, fine-tuning methods, and deployment constraints. Future design of RAG systems in engineering contexts should therefore adopt a holistic perspective—optimizing not just for peak accuracy, but for adaptability, explainability, and system-level integration.

6.8.5 Cross-Component Generalization and Use Case Reflection

Beyond isolated retrieval accuracy metrics, a key objective of this study was to assess how well the implemented system generalizes across different functional domains within the automotive testing pipeline. The curated benchmark dataset was intentionally constructed to span multiple ECU domains—including exterior lighting, CAN signal validation, powertrain diagnostics, and infotainment protocols—each with its own terminologies, structure, and testing conventions.

Despite this variation, fine-tuned models such as `bge-base-en-v1.5-HIL` demonstrated consistent performance across subsystems, suggesting that contrastive learning can capture underlying semantic relationships that transcend specific engineering contexts. In particular, the model showed robust performance in signal-heavy test sequences, where textual cues (for example signal names, activation triggers, threshold values) played a strong role in anchoring semantic similarity.

However, several limitations emerged in more abstract or loosely structured domains such as infotainment or general UX behavior validation. In these settings, requirements were often written in less formal language, lacked consistent signal grounding, or included user-facing phrases rather than system-level descriptors. Embedding models, including those fine-tuned, showed reduced alignment in such cases, likely due to weaker lexical overlap and fuzzier structural patterns. This suggests that for components with high abstraction or user-centric phrasing, more advanced representation techniques or hybrid retrieval (e.g., combining lexical and semantic matching) may be necessary. Alternatively, organizations may consider establishing more standardized writing conventions for requirement authoring—especially in human-facing modules—to improve machine readability and facilitate integration with large language model-based systems.

From a usability perspective, the A/B evaluation confirmed that the application of RAG had a clear downstream impact on user-perceived relevance and helpfulness of the assistant. Specifically, responses grounded in retrieved documents were consistently rated as more complete, accurate, and aligned with the users' intent compared to baseline outputs without document grounding. While the A/B setup did not isolate the effect of embedding fine-tuning, it demonstrated that the mere presence of relevant context significantly improved perceived truthfulness and user satisfaction. Interestingly, users consistently favored responses that were not only factually correct but also well-scoped, concise, and expressed using familiar domain-specific terminology. These findings highlight the importance of tight retrieval-generation integration and underscore the value of grounding mechanisms in high-stakes engineering environments.

Overall, these findings indicate that the implemented system—though initially agnostic to subsystem boundaries—exhibited emergent generalization across functional components. However, bridging the gap between structured technical content and human-understandable, operational language remains a challenge. Embedding model improvements must be complemented by interface and prompt engineering strategies that align closely with user mental models and real-world workflow constraints.

Chapter 7

Summary and Conclusions

7.1 Summary

This thesis investigated the effectiveness of LLMs in domain-specific retrieval applications, with a particular focus on requirement and test sequence alignment in automotive engineering contexts. The study was motivated by the lack of structured, labeled datasets and the limitations of traditional rule-based matching systems in handling semantic ambiguity across heterogeneous documentation formats.

To address this, a comprehensive framework was developed involving (i) the collection of requirement and test data, (ii) the creation of contrastive training data in triplet format, and (iii) the evaluation of fine-tuned embedding models within a RAG system. Both zero-shot and fine-tuned variants of multiple embedding architectures—including BGE, GTR-T5, and proprietary models—were benchmarked using a stratified retrieval accuracy test set and real-world user feedback.

Empirical results demonstrated that domain-specific fine-tuning consistently improved semantic alignment, particularly in structured system-level documentation where lexical overlap and identifier consistency were high. The best-performing model, a fine-tuned version of `bge-base-en-v1.5`, achieved over 60% top-1 retrieval accuracy, outperforming larger zero-shot models by a significant margin. Additionally, embedding enhancements translated into measurable downstream improvements when paired with smaller LLMs such as `GPT-4o-mini`, confirming the critical influence of retrieval quality on generation outcomes.

Complementing quantitative results, qualitative analysis from A/B testing indicated improved user satisfaction and perceived relevance for responses generated with RAG-based retrieval over pure LLM prompting. However, the degree of improvement varied by domain and abstraction level, highlighting the need for adaptive representation techniques.

This chapter concludes by reflecting on the implications of these findings, the constraints encountered during the study, and promising directions for future work.

7.2 Limitations and Future Work

Despite the promising results, several limitations must be acknowledged.

First, the training and evaluation data were collected from a limited scope within the automotive domain, primarily focusing on structured requirement–test sequence pairs in component-level system testing. While this setting offered clean alignment cues (signal IDs, subsystem names), it may not generalize to higher-level tasks such as user experience (UX) validation or infotainment logic, where documentation is often less structured and more subjective. The models evaluated here exhibited reduced performance in such abstract domains, indicating a need for more robust semantic representations or hybrid retrieval techniques that combine lexical and semantic signals.

Second, the fine-tuning process relied on a curated contrastive dataset of approximately 5,200 triplets. While effective, this dataset was constructed heuristically and may contain latent label noise, particularly in the selection of negative samples. Although hard negative mining partially mitigated this, future work could explore more principled data labeling strategies, including human-in-the-loop annotation, probabilistic labeling, or weak supervision with knowledge graphs.

Third, the LLM component of the RAG pipeline was only partially explored. GPT-4o and GPT-4o-mini were treated as black-box generators, and no fine-tuning or system prompt adaptation was performed. Moreover, the A/B evaluation included only a small sample of end users (10 participants), limiting the statistical robustness of observed trends. Longitudinal evaluation with broader user groups and domain-adaptive prompting could offer deeper insight into long-term usability and trustworthiness.

Fourth, performance metrics were centered on top- k retrieval accuracy and source attribution. These metrics, while suitable for benchmarking, do not capture broader usability dimensions such as relevance ranking, task completion efficiency, or human error reduction. Integrating task-level success metrics and user cognitive load assessments would offer a more holistic view of system utility in practice.

Finally, deployment constraints such as inference latency, memory consumption, and integration with legacy systems were discussed qualitatively but not empirically benchmarked. In future iterations, hardware-aware model selection, quantization techniques, and streaming inference may be explored to support real-time testing workflows.

In light of these limitations, several directions are proposed for future work:

- Extend the dataset to include higher-level abstractions (for example UX requirements, behavioral validations) and assess cross-domain generalization.
- Incorporate additional supervision strategies, such as ranking loss, reinforcement learning from user feedback, or multi-task learning with auxiliary tasks.
- Investigate user profiling and personalization for retrieval, especially in teams with domain-specific subcomponents or recurring testing themes.

- Conduct longitudinal deployment studies in real CI/CD environments to capture system-level impact and scalability under real-time constraints.
- Explore open-weight LLMs (for example Mistral or DeepSeek) for tighter coupling between retriever and generator, enabling end-to-end fine-tuning or adapter-based customization.

Overall, this thesis offers a practical blueprint for integrating LLM-based retrieval systems into automotive engineering pipelines, while laying the groundwork for broader adaptation in similarly structured domains.

7.3 Conclusion

This thesis concludes that fine-tuned embedding models, when integrated into a structured RAG pipeline, are highly effective for domain-specific retrieval tasks in industrial environments. Contrary to the intuition that larger models always perform better, results showed that moderate-scale models such as `bge-base-en-v1.5`, when adapted with high-quality contrastive supervision, outperformed larger zero-shot counterparts across both retrieval and generation metrics. Embedding quality proved to be a critical determinant of downstream performance, even when the LLM generator remained fixed.

Furthermore, the use of carefully mined hard negatives significantly improved representation learning, affirming the central role of supervision design. A/B testing further demonstrated that users favored responses backed by retrieved documents, especially when embeddings were fine-tuned to the task. Notably, smaller generators like `GPT-4o-mini` sometimes provided better source attribution than their larger counterparts, suggesting that compact LLMs may offer superior grounding in controlled retrieval settings.

Overall, the findings advocate for lightweight, domain-adapted RAG systems as a practical and effective solution for enhancing semantic alignment, traceability, and user satisfaction in structured engineering tasks.

Bibliography

- [1] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2017.
- [2] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*. John Wiley & Sons, 2011.
- [3] A. Bertolino, “Software testing research: Achievements, challenges, dreams”, *Future of Software Engineering (FOSE)*, pp. 85–103, 2007.
- [4] E. D. Cruise, “Automated software testing: Introduction, management, and performance”, *Addison Wesley*, 2001.
- [5] J. A. Whittaker, “What is software testing? and why is it so hard?”, *IEEE Software*, vol. 17, no. 1, pp. 70–79, 2000.
- [6] S. Matta and V. Garousi, “Mbtmodelgenerator: A software tool for reverse engineering of model-based testing (mbt) models from clickstream data of web applications”, *arXiv preprint arXiv:2506.08179*, 2025.
- [7] H. Jin, L. Hu, X. Li, *et al.*, “Jailbreakzoo: Survey, landscapes, and horizons in jail-breaking large language and vision-language models”, *arXiv preprint arXiv:2407.01599*, 2024.
- [8] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, “Software testing with large language models: Survey, landscape, and vision”, *IEEE Transactions on Software Engineering*, 2024.
- [9] D. Garikapati, Y. Liu, M. Brown, T. Littlehale, H. Yamamoto, and C. Bao, “Dual-cockpit human and hardware-in-the-loop test bench for autonomous vehicle development”, *IEEE Transactions on Intelligent Vehicles*, 2024.
- [10] J. Cheng, Z. Wang, X. Zhao, Z. Xu, M. Ding, and K. Takeda, “A survey on testbench-based vehicle-in-the-loop simulation testing for autonomous vehicles: Architecture, principle, and equipment”, *Advanced Intelligent Systems*, vol. 6, no. 6, p. 2300778, 2024.
- [11] J. F. Gaspar, R. F. Pinheiro, M. J. Mendes, M. Kamarlouei, and C. G. Soares, “Review on hardware-in-the-loop simulation of wave energy converters and power take-offs”, *Renewable and Sustainable Energy Reviews*, vol. 191, p. 114144, 2024.
- [12] S. Howick, I. Megiddo, *et al.*, “A framework for conceptualising hybrid system dynamics and agent-based simulation models”, *European Journal of Operational Research*, vol. 315, no. 3, pp. 1153–1166, 2024.

- [13] Z. Ali and Q. I. Ali, “An efficient design of a basic autonomous vehicle based on can bus”, *International Transactions on Electrical Engineering and Computer Science*, vol. 3, no. 1, pp. 41–56, 2024.
- [14] H. Hao, G. Xu, and Z. Yang, “Hardware-in-the-loop simulation of electric vehicle powertrain system”, May 2009, pp. 1–5. DOI: 10.1109/APPEEC.2009.4918397.
- [15] M. Jooriah, D. Datsenko, J. Almeida, A. Sousa, J. Silva, and J. Ferreira, “A co-simulation platform for v2x-based cooperative driving automation systems”, in *2024 IEEE Vehicular Networking Conference (VNC)*, IEEE, 2024, pp. 227–230.
- [16] H. Kim, J. Kwak, and J. Cho, “Autosar-compatible level-4 virtual ecu for the verification of the target binary for cloud-native development”, *Electronics*, vol. 13, no. 18, p. 3704, 2024.
- [17] Bosch GmbH, *CAN Specification Version 2.0*, Classical CAN protocol standard (Part A/B), 1991.
- [18] S. Annilsson, *Controller area network (can)*, Webpage, Detailed overview including Bosch CAN-2.0 and ISO-11898 specs, 2025.
- [19] V. I. GmbH, *Capl scripting*, Webpage, ”CAPL is an acronym for Communication Access Programming Language, which is a programming language used in Vector testing tools chain.”, 2022.
- [20] Wikipedia, *Capl*, <https://zh.wikipedia.org/wiki/CAPL>, 2024.
- [21] Wikipedia, *Canoe*, <https://zh.wikipedia.org/wiki/CANoe>, 2023.
- [22] Wikipedia, *Canalyzer*, <https://en.wikipedia.org/wiki/CANalyzer>, 2024.
- [23] OpenAI, “Gpt-4 technical report”, *arXiv preprint arXiv:2303.08774*, 2023.
- [24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding”, *arXiv preprint arXiv:1810.04805*, 2018.
- [25] A. Radford, J. Wu, *et al.*, “Language models are unsupervised multitask learners”, *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.
- [26] C. Raffel, N. Shazeer, A. Roberts, *et al.*, “Exploring the limits of transfer learning with a unified text-to-text transformer”, *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.
- [27] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, “A survey on large language models for code generation”, *arXiv preprint arXiv:2406.00515*, 2024.
- [28] S. Kim, D. Huang, Y. Xian, O. Hilliges, L. Van Gool, and X. Wang, “Palm: Predicting actions through language models”, in *European Conference on Computer Vision*, Springer, 2024, pp. 140–158.
- [29] R. Bommasani, D. A. Hudson, E. Adeli, *et al.*, “On the opportunities and risks of foundation models”, *arXiv preprint arXiv:2108.07258*, 2021.
- [30] Z. Ji, N. Lee, R. Frieske, *et al.*, “Survey of hallucination in natural language generation”, *ACM computing surveys*, vol. 55, no. 12, pp. 1–38, 2023.
- [31] Y. Hu and Y. Lu, “Rag and rau: A survey on retrieval-augmented language model in natural language processing”, *arXiv preprint arXiv:2404.19543*, 2024.

- [32] P. Lewis, E. Perez, A. Piktus, *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks”, *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.
- [33] H. Touvron, T. Lavril, *et al.*, “Llama: Open and efficient foundation language models”, *arXiv preprint arXiv:2302.13971*, 2023.
- [34] G. Izacard and E. Grave, “Leveraging passage retrieval with generative models for open domain question answering”, *arXiv preprint arXiv:2007.01282*, 2020.
- [35] Y. Wu, P. Chen, Z. Ding, and A. Yan, “Zero-shot dense retrieval based on query expansion”, in *International Conference on Artificial Intelligence Security and Privacy*, Springer, 2024, pp. 143–155.
- [36] Z. Sun, X. Wang, Y. Tay, Y. Yang, and D. Zhou, “Recitation-augmented language models”, *arXiv preprint arXiv:2210.01296*, 2022.
- [37] J. Yang, “Rethinking tokenization: Crafting better tokenizers for large language models”, *International Journal of Chinese Linguistics*, vol. 11, no. 1, pp. 94–109, 2024.
- [38] J. Ni, C. Qu, J. Lu, *et al.*, “Large dual encoders are generalizable retrievers”, *arXiv preprint arXiv:2112.07899*, 2021.
- [39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding”, in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.
- [40] J. Howard and S. Ruder, “Universal language model fine-tuning for text classification”, in *ACL*, 2018.
- [41] N. Houlsby *et al.*, “Parameter-efficient transfer learning for nlp”, in *ICML*, 2019.
- [42] E. J. Hu, Y. Shen, P. Wallis, *et al.*, “Lora: Low-rank adaptation of large language models”, *arXiv preprint arXiv:2106.09685*, 2021.
- [43] X. Li and P. Liang, “Prefix-tuning: Optimizing continuous prompts for generation”, in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.
- [44] B. Zaken *et al.*, “Bitfit: Simple parameter-efficient fine-tuning for transformers”, *arXiv preprint arXiv:2106.10199*, 2021.
- [45] E. Ben Zaken *et al.*, “Parameter-efficient methods for adapting pretrained models”, *arXiv preprint arXiv:2206.10577*, 2022.
- [46] L. Xue *et al.*, “Mt5: A massively multilingual pre-trained text-to-text transformer”, in *NAACL*, 2021.
- [47] Z. Feng *et al.*, “Codebert: A pre-trained model for programming and natural languages”, in *EMNLP*, 2020.
- [48] Y. Wang *et al.*, “Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation”, in *EMNLP*, 2021.

- [49] A. Araujo, M. Golo, B. Viana, F. Sanches, R. Romero, and R. Marcacini, “From bag-of-words to pre-trained neural language models: Improving automatic classification of app reviews for requirements engineering”, in *Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*, SBC, 2020, pp. 378–389.
- [50] D. Hendrycks, K. Lee, and M. Mazeika, “Using pre-training can improve model robustness and uncertainty”, in *International conference on machine learning*, PMLR, 2019, pp. 2712–2721.
- [51] J. Austin *et al.*, “Program synthesis with large language models”, *arXiv preprint arXiv:2108.07732*, 2021.
- [52] J. Maynez and et al., “On faithfulness and factuality in abstractive summarization”, in *ACL*, 2020.
- [53] P. Manakul and M. Gales, “Selfcheckgpt: Zero-resource hallucination detection for generative large language models”, in *ACL*, 2023.
- [54] J. Menick, M. Trebacz, V. Mikulik, *et al.*, *Teaching language models to support answers with verified quotes*, 2022. arXiv: 2203.11147 [cs.CL]. [Online]. Available: <https://arxiv.org/abs/2203.11147>.
- [55] S. J. Mielke and et al., “Between facts and fiction: Modeling uncertainty in neural language generation”, in *ACL*, 2022.
- [56] E. Bender, T. Gebru, *et al.*, “On the dangers of stochastic parrots: Can language models be too big?”, *FAccT*, 2021.
- [57] S. Gehman and et al., “Realtotoxicityprompts: Evaluating neural toxic degeneration in language models”, in *Findings of EMNLP*, 2020.
- [58] A. Abid, M. Farooqi, and J. Zou, “Persistent anti-muslim bias in large language models”, *Nature Machine Intelligence*, 2021.
- [59] J. Xu and et al., “Bot-adversarial dialogue for safe conversational agents”, in *EMNLP*, 2021.
- [60] S. Zhang and et al., “Language models are agents”, in *arXiv preprint arXiv:2308.08155*, 2023.
- [61] H. Xu, W. Zhang, Z. Wang, *et al.*, “Redagent: Red teaming large language models with context-aware autonomous language agent”, *arXiv preprint arXiv:2407.16667*, 2024.
- [62] L. Weidinger and et al., “Ethical and social risks of harm from language models”, *arXiv preprint arXiv:2112.04359*, 2021.
- [63] A. Tamkin and et al., “Understanding the capabilities, limitations, and societal impact of large language models”, *arXiv preprint arXiv:2102.02503*, 2021.
- [64] Anthropic, *Model context protocol (mcp): Connecting llms to tools, data, and streams*, <https://github.com/anthropics/mcp>, Accessed 2024-06, 2024.
- [65] S. Yao and et al., “React: Synergizing reasoning and acting in language models”, in *NeurIPS*, 2022.
- [66] T. Schick and et al., “Toolformer: Language models can teach themselves to use tools”, *arXiv preprint arXiv:2302.04761*, 2023.

- [67] OpenAI, *Openai function calling api*, <https://platform.openai.com/docs/guides/function-calling>, 2023.
- [68] G. Mialon and et al., “Augmented language models: A survey”, in *arXiv preprint arXiv:2302.07842*, 2023.
- [69] C. Xu and et al., “Toollm: Facilitating complex reasoning via tool-use over language models”, in *EMNLP*, 2023.
- [70] X. Gao and et al., “Pal: Program-aided language models”, in *ICLR*, 2023.
- [71] C. Qin and et al., “Toolbench: Benchmarking tool-augmented llms for real-world use”, in *arXiv preprint arXiv:2307.05331*, 2023.
- [72] Y. Liu and et al., “Agentbench: Evaluating foundation models as agents”, *arXiv preprint arXiv:2308.11458*, 2023.
- [73] C. Zheng and et al., “Challenges and applications of llm agents”, *arXiv preprint arXiv:2310.06874*, 2023.
- [74] T. Zhang and et al., “Llm4code: Lessons and challenges from fine-tuning gpt on proprietary source code”, in *NeurIPS*, 2023.
- [75] N. Carlini and et al., “Extracting training data from large language models”, in *USENIX Security*, 2021.
- [76] X. He *et al.*, “Extracting private training data from language models via unlearnable examples”, *arXiv preprint arXiv:2211.10802*, 2022.
- [77] J. Whitehouse and et al., “Privacy risks in publicly shared llm embeddings”, *arXiv preprint arXiv:2305.09069*, 2023.
- [78] N. Muennighoff and et al., “Crosslingual retrieval for multilingual llms”, *arXiv preprint arXiv:2212.10496*, 2022.
- [79] F. Liu and et al., “Retromae: Pre-training retrieval-augmented encoder for dense retrieval”, in *ACL*, 2022.
- [80] S. Ramalingam, *RAG in Action: Building the Future of AI-Driven Applications*. Libertatem Media Private Limited, 2023.
- [81] A. Balaguer, V. Benara, R. L. d. F. Cunha, *et al.*, “Rag vs fine-tuning: Pipelines, tradeoffs, and a case study on agriculture”, *arXiv preprint arXiv:2401.08406*, 2024.
- [82] M. Alizadeh, M. Kubli, Z. Samei, *et al.*, “Open-source large language models outperform crowd workers and approach chatgpt in text-annotation tasks”, *arXiv preprint arXiv:2307.02179*, vol. 101, 2023.
- [83] J. Chen, Z. Cai, K. Ji, *et al.*, “Huatuogpt-o1, towards medical complex reasoning with llms”, *arXiv preprint arXiv:2412.18925*, 2024.
- [84] H. Bousselham, A. Mourhir, *et al.*, “Fine-tuning gpt on biomedical nlp tasks: An empirical evaluation”, in *2024 International Conference on Computer, Electrical & Communication Engineering (ICCECE)*, IEEE, 2024, pp. 1–6.
- [85] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, “Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution”, in *2024 IEEE LLM Aided Design Workshop (LAD)*, IEEE, 2024, pp. 1–5.

- [86] J. Van Herck, M. V. Gil, K. M. Jablonka, *et al.*, “Assessment of fine-tuned large language models for real-world chemistry and material science applications”, *Chemical science*, vol. 16, no. 2, pp. 670–684, 2025.
- [87] W. Lu, R. K. Luu, and M. J. Buehler, “Fine-tuning large language models for domain adaptation: Exploration of training strategies, scaling, model merging and synergistic capabilities”, *npj Computational Materials*, vol. 11, no. 1, p. 84, 2025.
- [88] Q. Zhang, Z. Tian, Y. Lu, J. Niu, and C. Ye, “Experimental study on performance assessments of hvac cross-domain fault diagnosis methods oriented to incomplete data problems”, *Building and Environment*, vol. 236, p. 110264, 2023, ISSN: 0360-1323. DOI: <https://doi.org/10.1016/j.buildenv.2023.110264>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360132323002913>.
- [89] H. Yu, T. Cheng, Y. Cheng, and R. Feng, “Finemedlm-o1: Enhancing the medical reasoning ability of llm from supervised fine-tuning to test-time training”, *arXiv preprint arXiv:2501.09213*, 2025.
- [90] A. Aljohani and H. Do, “From fine-tuning to output: An empirical investigation of test smells in transformer-based test code generation”, in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, 2024, pp. 1282–1291.
- [91] P. Bécharde and O. M. Ayala, “Multi-task retriever fine-tuning for domain-specific and efficient rag”, *arXiv preprint arXiv:2501.04652*, 2025.
- [92] R. Ren, J. Ma, and Z. Zheng, “Large language model for interpreting research policy using adaptive two-stage retrieval augmented fine-tuning method”, *Expert Systems with Applications*, vol. 278, p. 127330, 2025.
- [93] Z. Nguyen, A. Annunziata, V. Luong, *et al.*, “Enhancing q&a with domain-specific fine-tuning and iterative reasoning: A comparative study”, *arXiv preprint arXiv:2404.11792*, 2024.
- [94] S. Alghisi, M. Rizzoli, G. Roccabruna, S. M. Mousavi, and G. Riccardi, “Should we fine-tune or rag? evaluating different techniques to adapt llms for dialogue”, *arXiv preprint arXiv:2406.06399*, 2024.
- [95] K. VM, H. Warrier, Y. Gupta, *et al.*, “Fine tuning llm for enterprise: Practical guidelines and recommendations”, *arXiv preprint arXiv:2404.10779*, 2024.
- [96] X. Mei, J. Shun, and K. Chao, “Efficient fine-tuning with low-rank adaptation for large-scale ai models”, *Available at SSRN 5173161*, 2024.
- [97] B. Wang, C. Ren, J. Yang, *et al.*, “Mac-sql: A multi-agent collaborative framework for text-to-sql”, *arXiv preprint arXiv:2312.11242*, 2023.
- [98] Y. Ge, W. Hua, K. Mei, *et al.*, “Openagi: When llm meets domain experts”, *Advances in Neural Information Processing Systems*, vol. 36, pp. 5539–5568, 2023.
- [99] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang, “Realm: Retrieval-augmented language model pre-training”, in *ICML*, 2020.
- [100] H. Trivedi, R. West, *et al.*, “Learning to retrieve reasoning paths over wikipedia graphs for question answering”, *arXiv preprint arXiv:2201.09941*, 2022.

- [101] V. Garousi, M. Felderer, Ç. M. Karapıçak, and U. Yılmaz, “Testing embedded software: A survey of the literature”, *Information and Software Technology*, vol. 104, pp. 14–45, 2018.
- [102] F. Elhambakhsh, D. Grandi, and H. Ko, “A domain adaptation of large language models for classifying mechanical assembly components”, *arXiv preprint arXiv:2505.01627*, 2025.
- [103] Z. Englhardt, R. Li, D. Nissanka, *et al.*, “Exploring and characterizing large language models for embedded system development and debugging”, in *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–9.
- [104] S. Gururangan, A. Marasović, S. Swayamdipta, *et al.*, “Don’t stop pretraining: Adapt language models to domains and tasks”, *arXiv preprint arXiv:2004.10964*, 2020.
- [105] T. Gao, X. Yao, and D. Chen, “Simcse: Simple contrastive learning of sentence embeddings”, *arXiv preprint arXiv:2104.08821*, 2021.
- [106] F. Liu, Z. Kang, and X. Han, “Optimizing rag techniques for automotive industry pdf chatbots: A case study with locally deployed ollama models”, *arXiv preprint arXiv:2408.05933*, 2024.
- [107] H. Qian, Z. Liu, K. Mao, Y. Zhou, and Z. Dou, “Grounding language model with chunking-free in-context retrieval”, *arXiv preprint arXiv:2402.09760*, 2024.
- [108] S. R. Bhat, M. Rudat, J. Spiekermann, and N. Flores-Herr, “Rethinking chunk size for long-document retrieval: A multi-dataset analysis”, *arXiv preprint arXiv:2505.21700*, 2025.
- [109] A. J. Yepes, Y. You, J. Milczek, S. Laverde, and R. Li, “Financial report chunking for effective retrieval augmented generation”, *arXiv preprint arXiv:2402.05131*, 2024.
- [110] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks”, *arXiv preprint arXiv:1908.10084*, 2019.
- [111] L. Wang, N. Yang, X. Huang, *et al.*, “Text embeddings by weakly-supervised contrastive pre-training”, *arXiv preprint arXiv:2212.03533*, 2022.
- [112] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei, “Multilingual e5 text embeddings: A technical report”, *arXiv preprint arXiv:2402.05672*, 2024.
- [113] L. Merrick, D. Xu, G. Nuti, and D. Campos, “Arctic-embed: Scalable, efficient, and accurate text embedding models”, *arXiv preprint arXiv:2405.05374*, 2024.
- [114] Pinecone, *Choosing an embedding model*, Pinecone blog, 2025.
- [115] D. Zhou, H. Tong, L. Wang, *et al.*, “Representation learning to advance multi-institutional studies with electronic health record data”, *arXiv preprint arXiv:2502.08547*, 2025.
- [116] W. Zhang and J. Zhang, “Hallucination mitigation for retrieval-augmented large language models: A review”, *Mathematics*, vol. 13, no. 5, p. 856, 2025.

- [117] T. Merth, Q. Fu, M. Rastegari, and M. Najibi, “Superposition prompting: Improving and accelerating retrieval-augmented generation”, in *Forty-first International Conference on Machine Learning*, 2024.
- [118] C. Sharma, “Retrieval-augmented generation: A comprehensive survey of architectures, enhancements, and robustness frontiers”, *arXiv preprint arXiv:2506.00054*, 2025.
- [119] S. Gupta, R. Ranjan, and S. N. Singh, “A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions”, *arXiv preprint arXiv:2410.12837*, 2024.
- [120] Z. Chen, C. Xu, D. Wang, *et al.*, “Rulerag: Rule-guided retrieval-augmented generation with language models for question answering”, *arXiv preprint arXiv:2410.22353*, 2024.
- [121] I. Radeva, I. Popchev, L. Doukovska, and M. Dimitrova, “Web application for retrieval-augmented generation: Implementation and testing”, *Electronics*, vol. 13, no. 7, p. 1361, 2024.
- [122] D. Garigliotti, “Explainable llm-powered rag to tackle tasks in the unstructured-structured data spectrum”, 2023.
- [123] A. Marshall, C. Bieck, J. Dencik, B. C. Goehring, and R. Warrick, “How generative ai will drive enterprise innovation”, *Strategy & Leadership*, vol. 52, no. 1, pp. 23–28, 2024.
- [124] Y. Mei, T. Nie, J. Sun, and Y. Tian, “Seeking to collide: Online safety-critical scenario generation for autonomous driving with retrieval augmented large language models”, *arXiv preprint arXiv:2505.00972*, 2025.
- [125] W. Jiang, S. Subramanian, C. Graves, G. Alonso, A. Yazdanbakhsh, and V. Daduq, “Rago: Systematic performance optimization for retrieval-augmented generation serving”, *arXiv preprint arXiv:2503.14649*, 2025.
- [126] S. Agarwal, S. Sundaresan, S. Mitra, *et al.*, “Cache-craft: Managing chunk-caches for efficient retrieval-augmented generation”, *arXiv preprint arXiv:2502.15734*, 2025.
- [127] R. Lakatos, P. Pollner, A. Hajdu, and T. Joo, “Investigating the performance of retrieval-augmented generation and fine-tuning for the development of ai-driven knowledge-based systems”, *arXiv preprint arXiv:2403.09727*, 2024.
- [128] M. Hindi, L. Mohammed, O. Maaz, and A. Alwarafy, “Enhancing the precision and interpretability of retrieval-augmented generation (rag) in legal technology: A survey”, *IEEE Access*, 2025.
- [129] H. Tatsat and A. Shater, “Beyond the black box: Interpretability of llms in finance”, *arXiv preprint arXiv:2505.24650*, 2025.
- [130] O. Komera and R. Manche, “Black-box behavior in large language models: Challenges and implications”, 2023.
- [131] S. Rosenberg, “Behind the curtain: The scariest ai reality”, *Axios*, 2025.
- [132] C. Olah *et al.*, “Ai is a black box. anthropic figured out a way to look inside”, *Wired*, 2024.

- [133] V. Ocleppo, “Enhancing requirements engineering with large language models: From elicitation and classification to traceability, ambiguity management and api recommendation”, Ph.D. dissertation, Politecnico di Torino, 2025.
- [134] A. Z. Khan, S. Iftikhar, R. H. Bokhari, and Z. I. Khan, “Issues/challenges of automated software testing: A case study”, *Pak. J. Comput. Inf. Syst*, vol. 3, no. 2, pp. 61–75, 2018.
- [135] D. Kici, G. Malik, M. Cevik, D. Parikh, and A. Basar, “A bert-based transfer learning approach to text classification on software requirements specifications.”, in *Canadian AI*, 2021.
- [136] A. J. Ratner, S. H. Bach, H. R. Ehrenberg, and C. Ré, “Snorkel: Fast training set generation for information extraction”, in *Proceedings of the 2017 ACM international conference on management of data*, 2017, pp. 1683–1686.
- [137] B. Nuseibeh and S. Easterbrook, “Requirements engineering: A roadmap”, in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 35–46.
- [138] N. Reimers and I. Gurevych, *Making monolingual sentence embeddings multilingual using knowledge distillation*, 2020. arXiv: 2004.09813 [cs.CL]. [Online]. Available: <https://arxiv.org/abs/2004.09813>.
- [139] K. Song, Y. Tan, T. Qin, J. Lu, and T.-Y. Liu, “Mpnet: Masked and permuted pre-training for language understanding”, in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/file/c3a690be93aa602ee2dc0ccab5b7b67e-Paper.pdf>.
- [140] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks”, in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Nov. 2019. [Online]. Available: <http://arxiv.org/abs/1908.10084>.
- [141] B. A. of Artificial Intelligence (BAAI), “Baai general embedding (bge) models: Bge-base-en-v1.5”, Beijing Academy of Artificial Intelligence, Tech. Rep., 2023, Model and technical documentation available at Hugging Face.
- [142] OpenAI, *Text-embedding-ada-002: New and improved embedding model*, OpenAI blog, Dec. 15, 2022, Accessed: 2025-06-23. [Online]. Available: <https://openai.com/index/new-and-improved-embedding-model/>.
- [143] OpenAI, *Text-embedding-3-small: A smaller, efficient embedding model*, OpenAI blog, Jan. 25, 2024, Accessed: 2025-06-23. [Online]. Available: <https://openai.com/blog/new-embedding-models-and-api-updates>.
- [144] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with gpus”, *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.
- [145] E. Öztürk and A. Mesut, “Performance analysis of chroma, qdrant, and faiss databases”.
- [146] X. Xie, H. Liu, W. Hou, and H. Huang, “A brief survey of vector databases”, in *2023 9th International Conference on Big Data and Information Analytics (BigDIA)*, IEEE, 2023, pp. 364–371.

- [147] Microsoft Azure, *Vector search in azure cosmos db for mongodb vcore*, <https://learn.microsoft.com/en-us/azure/cosmos-db/mongodb/vcore/vector-search-overview>, Accessed: 2025-06-12, 2024.
- [148] T. Wolf, L. Debut, V. Sanh, *et al.*, “Transformers: State-of-the-art natural language processing”, in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.
- [149] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.
- [150] J. Robinson, C.-Y. Chuang, S. Sra, and S. Jegelka, “Contrastive learning with hard negative samples”, *arXiv preprint arXiv:2010.04592*, 2020.
- [151] Y. Huang, K. Tang, M. Chen, and B. Wang, “A comprehensive survey on evaluating large language model applications in the medical industry”, *arXiv preprint arXiv:2404.15777*, 2024.
- [152] L. Huang, W. Yu, W. Ma, *et al.*, “A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions”, *ACM Transactions on Information Systems*, vol. 43, no. 2, pp. 1–55, 2025.
- [153] J. Kaplan, S. McCandlish, T. Henighan, *et al.*, “Scaling laws for neural language models”, *arXiv preprint arXiv:2001.08361*, 2020.

List of Figures

4.1	High-level system architecture for hybrid retrieval and generation	30
5.1	Software 3 Test cases Data Extraction	42
5.2	Cluster formation of intention-related texts (Sample 1)	43
5.3	Cluster formation of intention-related texts (Sample 2)	43
5.4	Overview of the fine-tuning training data generation	45
6.1	Data source distribution from internal systems (Software 1–3 and Signal References). Training and evaluation data are based on Software 3.	51
6.2	Top-1 retrieval accuracy vs. model size on full benchmark (15% queries, zero-shot setting)	54
6.3	Pre- and post-fine-tuning accuracy comparison across models (Top-1 retrieval)	55
6.4	Top-1 accuracy comparison for <code>bge-base-en-v1.5</code> under different training regimes	57
6.5	Source attribution accuracy under different embedding and LLM combinations	59
6.6	User evaluation ratings for Bot A (with RAG) vs. Bot B (no RAG)	60
6.7	Sentiment polarity of user feedback grouped by preferred bot	61

List of Tables

3.1	Database and Search Queries for Literature Review	18
3.2	Reviewed Research by Category	18
3.3	Comparison of Open-Source and Closed-Source Embedding Models	23
3.4	Comparison of Fine-Tuning and RAG for Domain Adaptation	28
5.1	Candidate Embedding Models Evaluated During Selection Phase	44
6.1	Triplet accuracy on 10% benchmark subset	53
6.2	Effect of fine-tuning on Top-1 retrieval accuracy	54
6.3	Sentiment Analysis of Bot A Feedback	62

Appendix A

User Feedback Summary

This appendix presents representative user feedback collected during the A/B evaluation phase. Each entry has been standardized and paraphrased based on raw survey data to reflect core evaluation dimensions—such as helpfulness, speed, and overall satisfaction. While not quoted verbatim, all entries preserve the meaning and intent of the original user responses.

- **User Domain:** Quality/Operation
Preferred Bot: B
Helpful Reason: Asking similar questions, Chatbot B gave me more direct answers with a good language, easy to understand.
Speed: B
Overall Satisfaction: 4
Additional Comments: Chatbot B gave what the tester needs faster, Testcases covering multiple scenarios. What A is doing mostly is writing long texts with suggestions. Testers are looking for usecases and examples.
- **User Domain:** Other
Preferred Bot: A
Helpful Reason: A clearly outlined the operational steps needed, with relevant examples.
Speed: B
Overall Satisfaction: 4
Additional Comments: —
- **User Domain:** Other
Preferred Bot: A
Helpful Reason: Bot A broke down each step logically and answered all parts of my question.
Speed: B
Overall Satisfaction: 4
Additional Comments: —

- **User Domain:** Quality/Operation
Preferred Bot: A
Helpful Reason: Gave direct actions for CI issues and matched what I need but the output style need to be improved.
Speed: B
Overall Satisfaction: 3
Additional Comments: Maybe provide a downloadable checklist.
- **User Domain:** Quality/Operation
Preferred Bot: A
Helpful Reason: Understood my Docker-related CI problem and suggested realistic fixes.
Speed: B
Overall Satisfaction: 3
Additional Comments: —
- **User Domain:** Quality/Operation
Preferred Bot: A
Helpful Reason: —
Speed: —
Overall Satisfaction: 4
Additional Comments: —
- **User Domain:** Quality/Operation
Preferred Bot: A
Helpful Reason: Better at explaining steps for automation logic and testcases.
Overall Satisfaction: 3
Additional Comments: —
- **User Domain:** Exterior Function
Preferred Bot: A
Helpful Reason: Better at identifying locking signals and giving suggestions.
Speed: B
Overall Satisfaction: 4
Additional Comments: —
- **User Domain:** Other
Preferred Bot: A
Helpful Reason: helped trace testcases logic with concrete examples.
Tone: Neutral
Overall Satisfaction: 3
Additional Comments: —
- **User Domain:** Exterior Function
Preferred Bot: A
Helpful Reason: correctly handled signal decoding based on input format compared with b
Speed: B

Overall Satisfaction: 4

Additional Comments: —