Institut National des Sciences Appliquées de Toulouse

Optimizing and Adapting Language Models for Domain-Specific Task

End-of-studies Apprenticeship Report

Minh Duy Nguyen

Bachelor's Degree in Computer Science and Engineering

Supervised by: Milad Mozafari (Torus AI), David Bertoin (INSA Toulouse)

January 21, 2026

Abstract

[**INFO:** *The abstract is essentially informative in nature and should be written concisely (up to 200 words) in a way that captures the interest of the reader.*

The Abstract replaces reading the document and does not contain figures, tables, citations, etc. It should include the following topics: scope, objectives, methods, main findings, including results, conclusions and recommendations, if any.

For more information on how to write a good abstract, consult the online tutorial available on the Library website, "Publication Support Guide", section: "Structuring Technical Report".]

Write the Abstract, but only at the end.

Acknowledgement

[**INFO (optional element):** *Usually the contribution of other people or entities is mentioned, both for carrying out the study and for producing the report. They can be done on a separate page or included in the introduction.*]

Table of contents

Todo list

Figures and tables

bash
: Bash is a *Unix shell* and command language written in 1989 by Brian Fox for the GNU Project as a free software replacement for the *Bourne shell*.

firewall
: In computing, a *firewall* is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. A *firewall* typically establishes a barrier between a trusted network and an untrusted network, such as the Internet.

Figures and tables
: Figures and tables are essential elements in a report, providing visual representations of data and information. They should be clearly labeled and referenced in the text.

Chapter 1

Introduction

1.1 General Context

1.1.1 Evolution of Large Language Model

Over the past decade, the fields of Artificial Intelligence (AI) and Natural Language Processing (NLP) have witnessed groundbreaking advances, particularly since the emergence of the Transformer architecture [Vaswani et al., 2017]. Large Language Models (LLMs) such as GPT-5 [OpenAI, 2025], Claude 4 [Anthropic, 2025], Gemini 2.5 [Google, 2025], and open-source models like Llama 3 [Meta, 2024] and Qwen 2.5 [Alibaba, 2024] have demonstrated superior capabilities in understanding context, generating text, and performing logical inference across a variety of tasks.

According to a McKinsey Global Institute report (2024), the generative AI market is expected to reach a value of $4.4 trillion by 2030. However, a Gartner survey (2024) indicates that 67% of businesses struggle to deploy LLMs for specialized tasks due to accuracy limitations and operating costs.

1.1.2 The Research-Industrial Application Gap

Transferring language models from the research environment to practical industrial applications (Industrial Deployment) faces significant challenges. Although Foundation Models possess a vast amount of general knowledge, they often encounter two serious limitations when solving domain-specific tasks:

Knowledge Gap

In critical fields such as healthcare, finance, and insurance, information is often contained in complex documents (financial reports, medical technical catalogs) and changes constantly. LLMs face three main problems:

- Static Knowledge: Model knowledge is limited by the training time (knowledge cutoff), making it impossible to update new information without retraining [Lewis et al., 2020].

- Private Data Inaccessibility: The model cannot access confidential organizational documents such as contracts, internal procedures, or patient records.
- Hallucination: LLMs tend to confidently generate false information [Ji et al., 2023], which is unacceptable in medical or financial decisions where near-absolute accuracy is required.

Efficiency & Behavior Gap

For tasks requiring the model to adhere to a specific behavioral scenario, a particular language style (e.g., psychological counselor, analyst), or deployment on limited hardware infrastructure, using massive models (hundreds of billions of parameters) is not cost-effective and prone to latency. According to Hoffmann et al. (2022), the inference cost of GPT-4 can reach $0.06/1K tokens, making large-scale deployment unfeasible for many small and medium-sized enterprises.

1.1.3 Research motivation

Based on the above reality, this thesis focuses on researching and implementing advanced techniques to **optimize and adapt Language Models for specific data domains**, with two main approaches:

- Retrieval-Augmented Generation (RAG): Integrating external knowledge bases to address the Knowledge Gap.
- Parameter-Efficient Fine-Tuning (PEFT): Adapting model behavior to address the Behavior Gap.

1.2 Internship Environment

This master's thesis was completed at **Torus AI**, a technology company with the mission *Intelligence for Life*, specializing in providing advanced AI solutions to improve quality of life and business efficiency.

I worked in the Research and Development Team (R&D Team - Torus Lab) as a Machine Learning Engineer Alternant. At Torus AI, the R&D department acts as a bridge between the latest academic research (State-of-the-Art) and commercial products. The team's main task is to continuously explore emerging Generative AI technologies, assess their feasibility, and build functional prototypes (PoCs) to verify their effectiveness before integration into the main product system.

The R&D work environment demands flexible thinking: not just using existing APIs, but delving into customizing architecture, optimizing data processing pipelines, and quantitatively evaluating technical solutions.

My role in the R&D department includes:

- Researching emerging Generative AI technologies

- Evaluating feasibility and building functional prototypes (PoCs)
- Integrating solutions into product systems

Resources provided:

- GPU infrastructure: NVIDIA RTX 3090 (24GB)
- Real-world enterprise data: Medical documents (CCAM, NGAP), insurance financial reports
- API access: OpenAI GPT-4, Google Gemini, Qwen API

1.3 Problem Statement

During our work in the R&D department, we identified two core problems that needed to be addressed when applying GenAI in practice, corresponding to two main technical approaches:

Problem 1: Integration of External Knowledge from Complex Unstructured Data

Partner businesses (such as insurance companies, healthcare facilities) possess large amounts of data in the form of PDF documents containing text, tables, and images. Traditional RAG (Retrieval-Augmented Generation) methods based on plain text (text-only) fail to understand the semantics of complex tables or visual information. The question is: How to build a Multimodal RAG pipeline capable of accurately parsing, indexing, and retrievaling information from these mixed documents to support decision-making (e.g., medical refund code lookup, financial data analysis)?

Problem 2: Behavioral Adaptation and Resource Optimization for Small Models (Behavioral Adaptation & Efficiency)

For applications requiring high interactivity, counseling, or entertainment (such as psychological counseling chatbots or Tarot Readers), the requirement is not only for information accuracy but also for consistency in tone and style and rule-based reasoning. Using large models (like GPT-4) via APIs is both costly to operate and difficult to fully control behavior. The question is: Is it possible to fine-tune small language models (such as Qwen, Llama $<$ 7B parameters) using parameter optimization techniques (PEFT/LoRA) and quantization so that they achieve inference capabilities and writing styles comparable to large models, but can be run locally at low cost?

1.4 Objectives and expected results

This thesis aims to achieve the following objectives:

1. Design and implement an Advanced Multimodal RAG pipeline capable of accurately extracting and retrieving information from complex documents containing text, tables, and images in the medical and financial domains.
2. Develop a methodology for fine-tuning Small Language Models (< 2B parameters) using PEFT/LoRA techniques to achieve domain-specific behavior adaptation while enabling cost-effective local deployment.
3. Establish systematic evaluation frameworks for both RAG retrieval quality and fine-tuned model performance, enabling reproducible benchmarking.

1.5 Report structure

In addition to this introduction, this report is organized into four main chapters and appendices. Chapter 2 provides the theoretical background, introducing the foundational concepts of Retrieval-Augmented Generation (RAG) and Parameter-Efficient Fine-Tuning (PEFT), followed by an overview of Knowledge Distillation and Synthetic Data Generation techniques. Chapter 3 addresses the Knowledge Gap by presenting an Advanced Multimodal RAG pipeline for processing complex medical and financial documents. Chapter 4 tackles the Behavior Gap through LoRA fine-tuning of small Qwen models for persona-based tasks. Chapter 5 concludes the thesis by summarizing contributions and proposing future research directions. The Appendices provide supplementary materials including prompt templates and training configurations.

Chapter 2

Theoretical Background

This chapter presents the fundamental concepts and techniques that underpin the research conducted in subsequent chapters. We begin with an overview of the Transformer architecture and Large Language Models (LLMs), then proceed to discuss Retrieval-Augmented Generation (RAG) pipelines, embedding techniques, and conclude with Parameter-Efficient Fine-Tuning methods. Understanding these concepts is essential for comprehending the advanced techniques and experimental results presented in following chapters.

2.1 Transformer Architecture and Large Language Models

2.1.1 The Transformer Revolution

The Transformer architecture, introduced by Vaswani et al. (2017), marked a paradigm shift in natural language processing. Unlike recurrent neural networks (RNNs) that process sequences sequentially, Transformers leverage a self-attention mechanism that allows parallel processing of all positions in a sequence simultaneously. This fundamental change enables both faster training and better capture of long-range dependencies in text.

The core innovation of the Transformer is the *scaled dot-product attention* mechanism, defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

where Q, K, and V represent query, key, and value matrices respectively, and d_k is the dimension of the keys. This mechanism allows the model to dynamically focus on relevant parts of the input when generating each output token.

The Transformer encoder-decoder structure consists of stacked layers, each containing multi-head attention and feed-forward neural network (FFN) sublayers. The multi-head attention allows the model to attend to information from different representation subspaces:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.2}$$

where each $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

2.1.2 Large Language Models

Building upon the Transformer foundation, Large Language Models (LLMs) are trained on massive text corpora using self-supervised learning objectives, primarily next-token prediction. Models such as GPT-4 [OpenAI, 2023], Claude [Anthropic, 2024], and open-source alternatives like Llama [Touvron et al., 2023] and Qwen [Bai et al., 2023] have demonstrated remarkable capabilities in understanding context, generating coherent text, and performing complex reasoning tasks.

Despite their impressive capabilities, LLMs face two fundamental limitations when deployed for domain-specific tasks:

Knowledge Gap: LLM knowledge is frozen at training time (knowledge cutoff), preventing access to recent information or proprietary organizational data. Additionally, LLMs exhibit a tendency to generate plausible-sounding but factually incorrect information—a phenomenon known as "hallucination" [Ji et al., 2023].

Behavior Gap: For applications requiring specific communication styles, workflows, or domain expertise, generic LLMs often fail to maintain consistency, especially over extended conversations.

These limitations motivate the two main technical approaches explored in this thesis: Retrieval-Augmented Generation (Chapter 3) and Fine-tuning (Chapter 4).

2.2 Retrieval-Augmented Generation

2.2.1 Motivation and Basic Architecture

Retrieval-Augmented Generation (RAG), introduced by Lewis et al. (2020), addresses the Knowledge Gap by combining the parametric knowledge of LLMs with non-parametric retrieval from external knowledge bases. Instead of relying solely on information encoded in model weights, RAG systems retrieve relevant documents at inference time and incorporate them into the generation context.

A basic RAG pipeline consists of two main phases:

Indexing Phase (offline): Documents are processed, chunked into manageable segments, converted to vector embeddings, and stored in a vector database.

Query Phase (online): When a user submits a query, the system retrieves relevant document chunks, augments the prompt with this context, and generates an answer using the LLM.

Figure 2.1 illustrates the standard RAG pipeline architecture.

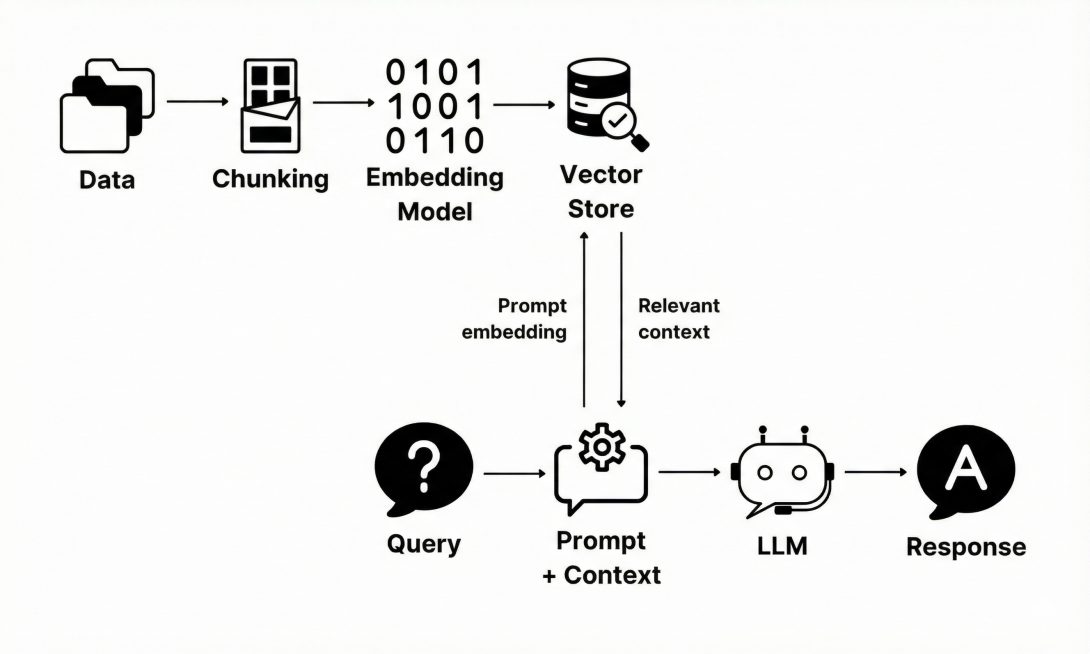


Figure 2.1: Basic RAG Pipeline Architecture

The RAG approach offers several advantages: it enables access to up-to-date information without retraining, provides transparent attribution through cited sources, and reduces hallucination by grounding responses in retrieved evidence.

2.2.2 Document Processing and Chunking

The quality of a RAG system heavily depends on how documents are processed during indexing. Raw documents—particularly PDFs containing complex layouts with tables, images, and mixed formatting—must be parsed and segmented appropriately.

Chunking strategies determine how documents are divided into retrievable units. Common approaches include:

- **Fixed-size chunking**: Dividing text into segments of predetermined token length with optional overlap. Simple but may break semantic units.
- **Semantic chunking**: Using natural boundaries (paragraphs, sections) to preserve semantic coherence.
- **Recursive chunking**: Hierarchically splitting documents using multiple delimiters.

The chunk size represents a critical trade-off: smaller chunks enable more precise retrieval but may lack sufficient context, while larger chunks provide more context but reduce retrieval precision and consume more of the LLM's context window.

2.3 Text Embeddings and Vector Search

2.3.1 Embedding Models

Text embeddings are dense vector representations that capture semantic meaning, enabling similarity-based retrieval. Modern embedding models, typically based on the Transformer architecture, map text sequences to fixed-dimensional vectors in a learned semantic space where similar texts are positioned closer together.

Notable embedding models include Sentence-BERT [Reimers and Gurevych, 2019], which adapts BERT for sentence-level representations, and more recent models like E5 [Wang et al., 2022] and BGE [Xiao et al., 2023] that achieve state-of-the-art performance on retrieval benchmarks. For multilingual applications, models like Multilingual-E5 support cross-lingual retrieval by mapping texts from different languages into a shared semantic space.

The embedding process can be formalized as a function $f : \mathcal{T} \to \mathbb{R}^d$ that maps text $t \in \mathcal{T}$ to a d-dimensional vector. Semantic similarity between texts t_1 and t_2 is then computed using metrics such as cosine similarity:

$$\text{sim}(t_1, t_2) = \frac{f(t_1) \cdot f(t_2)}{\|f(t_1)\|\|f(t_2)\|} \tag{2.3}$$

2.3.2 Vector Databases and Approximate Nearest Neighbor Search

Vector databases are specialized storage systems optimized for similarity search over high-dimensional embeddings. Given a query embedding, these systems efficiently retrieve the k most similar vectors from potentially millions of stored embeddings.

Exact nearest neighbor search has $O(n)$ complexity, which becomes prohibitive for large-scale applications. Vector databases therefore employ Approximate Nearest Neighbor (ANN) algorithms that trade slight accuracy reduction for dramatic speed improvements. Popular techniques include:

- **HNSW** (Hierarchical Navigable Small World): Constructs a multi-layer graph structure enabling logarithmic search complexity [Malkov and Yashunin, 2018].
- **IVF** (Inverted File Index): Partitions the vector space into clusters and searches only relevant partitions.
- **Product Quantization**: Compresses vectors to reduce memory footprint while maintaining search quality.

Common vector database implementations include Chroma, Pinecone, Milvus, and Qdrant, each offering different trade-offs between performance, scalability, and ease of use.

2.4 Parameter-Efficient Fine-Tuning

2.4.1 Fine-Tuning Fundamentals

Fine-tuning adapts a pre-trained model to specific downstream tasks by continuing training on task-specific data. While effective, traditional full fine-tuning—updating all model parameters—presents challenges for large models: it requires storing separate copies of the full model for each task and demands substantial computational resources.

2.4.2 Low-Rank Adaptation (LoRA)

Low-Rank Adaptation (LoRA), introduced by Hu et al. (2021), addresses these challenges by freezing pre-trained weights and injecting trainable low-rank decomposition matrices into each layer. The key insight is that weight updates during fine-tuning have low intrinsic rank, meaning they can be effectively approximated by low-rank matrices.

For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA represents the update as:

$$W = W_0 + \Delta W = W_0 + BA \tag{2.4}$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ with rank $r \ll \min(d, k)$.

During training, W_0 is frozen while A and B are optimized. This reduces trainable parameters from $d \times k$ to $r \times (d + k)$—a dramatic reduction when r is small (typically 8–64). For example, with $r = 16$ and $d = k = 4096$, trainable parameters decrease from 16.7M to 131K per matrix.

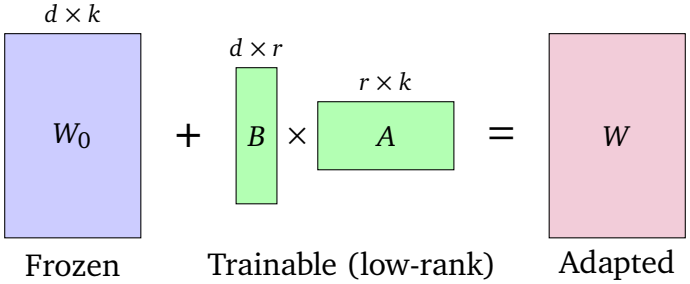


Figure 2.2: LoRA: Low-Rank Adaptation. The original weight W_0 is frozen, while only the low-rank matrices A and B are trained.

LoRA offers several practical advantages:

- **Memory efficiency**: Only low-rank matrices require gradient storage.
- **No inference latency**: Matrices can be merged with base weights post-training.
- **Task switching**: Multiple LoRA adapters can share a single base model.

2.4.3 Target Modules for Adaptation

An important design choice in LoRA is selecting which layers to adapt. The original work focused on attention projection matrices (W_Q, W_K, W_V, W_O). However, subsequent research has shown that including feed-forward network (FFN) layers can improve performance for certain tasks.

For behavioral fine-tuning—where the goal is to adapt not just what the model attends to but how it processes information—applying LoRA to both attention and FFN layers proves beneficial. This comprehensive approach enables the model to learn both new attention patterns and new transformation functions.

2.5 Chapter Summary

This chapter has established the theoretical foundation for the technical contributions presented in this thesis. We introduced the Transformer architecture and its evolution into Large Language Models, explained how RAG systems address knowledge limitations through retrieval-augmented generation, discussed embedding techniques and hybrid search strategies for effective retrieval, presented LoRA as an efficient method for model adaptation, and outlined evaluation methodologies for both retrieval and generation quality.

With this foundation, Chapter 3 will demonstrate how these concepts are integrated into an Advanced RAG pipeline for enterprise document understanding, while Chapter 4 will apply fine-tuning techniques to adapt small language models for specialized behavioral tasks.

Chapter 3

Multi-modal RAG for Complex Document Understanding

3.1 Real-world problem: Two business use cases

During our work at Torus AI, we received two requests from partners with similar challenges but in two completely different fields: finance and healthcare. Despite the distinctly different business contexts, both faced the same core problem: how to effectively extract knowledge from highly information-dense, unstructured documents — specifically, documents containing numerous tables, images, and complex multimedia data.

3.1.1 Finance use case: Q&A System for SFCR Reports (GPM)

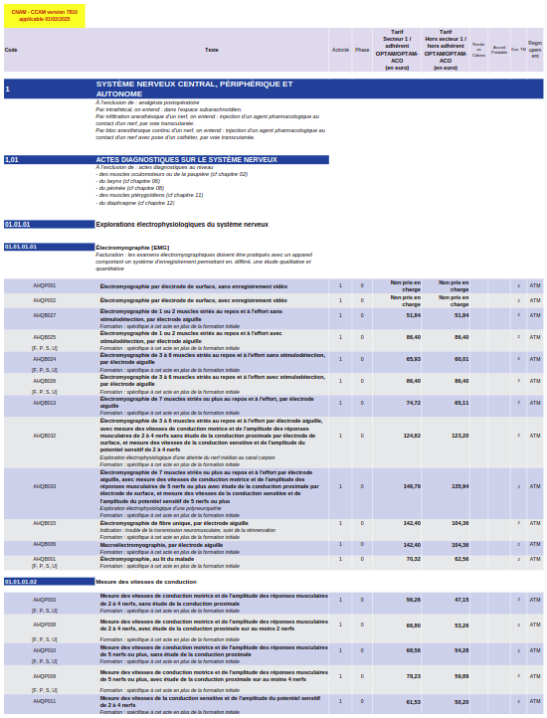
GPM (Gestion Patrimoine Mutualiste) is a mutual insurance fund under the AGMF group (Association Générale des Médecins de France). Their requirement seemed simple: build a rapid Q&A system so employees could look up information from annual financial reports.

GPM faces a massive volume of SFCR (Solvency and Financial Condition Reports) and annual accounting reports. The challenge lies not only in the document length (50-200 pages) but also in the multi-modal nature of the data. Figure 3.1b illustrates the structure of one such data page.

Crucial knowledge is often not found in plain text but is condensed into multi-dimensional balance sheets, trend charts, risk matrices, etc. Querying metrics such as "2024 Solvency Ratio" requires the system to have spatial understanding of the table structure rather than simply reading a string of characters.

3.1.2 Healthcare use case: CCAM Catalog Coding Lookup (Dr. Besnier)

Conversely, the problem from Dr. Besnier focuses on the CCAM (Classification Commune des Actes Médicaux) system—a complex classification system comprising thousands of French medical codes used to determine insurance reimbursement levels. Figure 3.1a illustrates the structure of this catalog.



(a) One page in the CCAM codes document

Le **chiffre d'affaires** de la ligne d'activité « santé assimilée à la vie » s'élève à 9 752 k€, en hausse de 2% à la fin 2024 par rapport à la fin 2023.

La **charge de sinistres** de la ligne d'activité « santé assimilée à la vie » s'établit à 6 410 k€ en 2024, soit une hausse de 25% par rapport à l'exercice 2023.

La **cession** a un solde négatif de 3 540 k€ en 2024, en hausse de 7% par rapport à l'exercice 2023 et expliquée par la mise en place en 2022 d'un traité de réassurance avec Allianz.

Les **frais** s'élèvent 3 084 k€ en 2024, en baisse de 14% par rapport à la fin 2023.

En conséquence, le **résultat technique net** de frais et de réassurance de la ligne d'activité « santé assimilée à la vie » s'établit à – 3 282 k€ au titre de l'exercice 2024 contre – 2 504 k€ en 2023, soit une baisse de 31%.

A.2.2.2. Résultat Rentes

Résultat de souscription (en k€)	31/12/2024	31/12/2023	Variation en %
Primes brutes	43 899	40 476	8%
Charges sinistres brutes	15 609	13 594	15%
Résultat technique brut	**28 290**	**26 882**	**5%**
Primes cédées	12 630	11 728	8%
Charges sinistres cédées	3 523	2 636	34%
Résultat technique cédé	**9 107**	**9 092**	**0%**
Primes nettes	31 269	28 747	9%
Charges sinistres nettes	12 086	10 958	10%
Résultat technique net	**19 183**	**17 789**	**8%**
Frais administratifs	6 451	7 883	-18%
Frais de gestion des investissements	115	118	-3%
Frais de gestion des sinistres	991	1 005	-1%
Frais d'acquisition	4 170	4 566	-9%
Frais généraux	-	-	n.s
Total frais	**11 727**	**13 573**	**-14%**
Résultat technique net y compris frais	**7 456**	**4 216**	**77%**

Le **chiffre d'affaires** de la ligne d'activité « rentes issues des contrats non-vie » s'élève à 43 899 k€, en hausse de 8% à la fin 2024 par rapport à la fin 2023. Cette hausse est expliquée par le développement du réseau commercial et les majorations tarifaires lors du renouvellement 2024.

La **charge de sinistres** de la ligne d'activité « rentes issues des contrats non-vie » s'établit à 15 609 k€ en 2024, en hausse de 15% par rapport à l'exercice 2023.

La **cession** a un solde négatif de 9 107 k€ en 2024 et en stabilité par rapport à l'exercice 2023.

Les **frais** s'élèvent 11 727 k€ en 2024, en baisse de 14% par rapport à la fin 2023.

En conséquence, le **résultat technique net** de frais et de réassurance de la ligne d'activité « rentes issues des contrats non-vie » s'établit à 7 456 k€ au titre de l'exercice 2024 contre 4 216 k€ en 2023.

(b) One page in GPM reports

Figure 3.1: Illustrating CCAM and GPM documentation.

The challenge here is that CCAM data contains thousands of medical codes organized in an extremely complex tabular structure. Each code (e.g., HAFA008) comes with strict application conditions and different reimbursement rates based on the clinical context.

A practical query might be: "*Patiente 69 ans, exérèse de carcinome basocellulaire de la lèvre, suture par lambeau à la volée*". The system needs to return appropriate CCAM codes like HAFA008 or QAMA002 — a task requiring not just semantic understanding but also exact matching with specific codes. A minor error in code matching can lead to discrepancies in medical records and the insurance reimbursement process.

3.1.3 Technical barriers and the need for an in-house pipeline

The commonalities between these two use-cases identified three main barriers that forced us to build an In-house Advanced RAG Pipeline:

1. **Strict Data Security Requirements**: Both financial documents and medical records are sensitive corporate data. Under business and industry standards, using cloud services like ChatGPT or Claude API directly with raw data is prohibited due to data leakage risks. Customers require a solution that runs entirely on-premise or within their private cloud.

2. **High Accuracy**: In these two fields, "hallucination" is unacceptable. An incorrect answer regarding revenue could lead to poor investment decisions; an inaccurate medical code directly affects patient treatment and insurance reimbursement. The system needs a transparent attribution (citation) mechanism.

3. **Native LLM Limitations**: The limited context window of language models cannot encompass the entire massive document repository, and their real-time knowledge update capability is restricted without an efficient Retrieval mechanism.

RAG is the ideal solution: retrieve relevant information from a knowledge base, then augment the prompt for the LLM to generate the answer. This allows the LLM to "learn" from new documents without the need for fine-tuning.

3.1.4 Objective: A Unified Pipeline for All Documents

Instead of developing two separate pipelines, we set a more ambitious goal: to build a unified RAG pipeline capable of effectively processing all types of multimodal documents. This pipeline must achieve:

- High accuracy for both financial and medical documents.
- Ability to process complex tables with 2D structures.
- Support for both semantic and exact match (keyword/code-specific) retrieval.
- Full local execution without dependence on external cloud services.

In the following sections, we will analyze why **simple RAGs are insufficient**, and then present our advanced pipeline architecture with each component designed to address a specific problem.

3.2 Why Simple RAG Fails: An Empirical Analysis

Before proposing a complex solution, we performed a crucial step: testing a Simple RAG baseline and quantitatively measuring its limitations. This not only justifies the need for advanced techniques but also helps pinpoint the exact "bottlenecks" to be resolved.

3.2.1 Simple RAG Baseline Configuration

The Simple RAG pipeline was implemented with the most standard components – typical of what you would find in most RAG tutorials:

- **Document Parsing**: PyPDF to extract text from PDFs.
- **Chunking**: Fixed-size chunking with 512 tokens/chunk and 50 tokens overlap.
- **Embedding Model**: `sentence-transformers/all-MiniLM-L6-v2` – the most popular model with 22M parameters.
- **Vector Store**: ChromaDB with Dense Search (cosine similarity).
- **LLM**: Gemma-3-12b-it to generate answers.

3.2.2 Issue 1: Loss of Table Structure

The most serious problem arises during the Document Parsing stage. Traditional extractors like PyPDF "flatten" the 2D structure of tables into a 1D text string. Consider this specific example from a GPM document:

Table 3.1: Structure Comparison: Original Table vs. Text Extracted by PyPDF

a) Original Table in PDF:

Sous modules (en k€)	SCR 2024	SCR 2023	Evol.
Type 1	2 692	2 487	8 %
Type 2	28 377	36 221	-22 %
Diversification	-621	-586	6 %
Risque de défaut	**30 448**	**38 121**	**-20 %**

b) After PyPDF Extraction:

```
Sous modules (en k€) SCR 2024 SCR 2023 Evolution
Type 1 2 692 2 487 8 %
Type 2 28 377 36 221 -22 %
Effet de diversification -621 -586 6 %
Risque de défaut 30 448 38 121 -20 %
```

With this text string, answering a question like "*What is the SCR 2024 for Type 2?*" becomes very difficult for the LLM because the semantic relationship between the Header (Column/Row) and the Value is severed. The system struggles to distinguish whether the number "28 377" belongs to the "SCR 2024" column or is two separate numbers "28" and "377", and whether it corresponds to the "Type 2" row due to the loss of alignment. This leads to completely inaccurate quantitative answers.

3.2.3 Issue 2: Failure with Exact Code Matching

For the medical use-case, we observed that Dense Search (Vector Search) frequently returned noisy results when dealing with specific codes (like HAFA006). This is because embedding models focus on semantic similarity. However, medical codes often do not carry natural language meaning; they are entities that require an Exact Match. Relying solely on Vector Search causes the system to suggest codes that "look similar" but are medically entirely different.

Figure 3.2 shows an example of retrieval results from Simple RAG for a query asking for the code HAFA006:

```
================================================================================
📊 ANALYSIS OF RESULTS
================================================================================

1️⃣ HAFA006 POSITION:
   ✅ Found HAFA006 at rank #3
   ⚠️  HYPOTHESIS PARTIALLY CONFIRMED: Ranked #3 (not #1, but still high)

2️⃣ SIMILAR CODES RANKED HIGHER:
   Found 5 chunks with similar HAFA00X codes:
   - Rank #1: ['HAFA004', 'HAFA002'] (score: 0.2544)
   - Rank #2: ['HAFA005', 'HAFA003'] (score: 0.2315)
   - Rank #3: ['HAFA007'] (score: 0.2199)
   - Rank #4: ['HAFA001'] (score: 0.2107)
   - Rank #5: ['HAFA009'] (score: 0.2107)
```

Figure 3.2: Dense Search results for the query “HAFA006”

The retriever returns the passage containing the code at position 3, and all higher positions are matched with other similar codes because Dense Search doesn't understand that the user needs the exact "HAFA006", not a similar code.

3.2.4 Other Issues

The "Lost in the Middle" phenomenon discovered by Liu et al. (2023): when context contains many passages (e.g., 10-20), LLMs tend to focus on the beginning and the end, ignoring information in the middle. In our tests with top-20 retrieval, when the ground-truth passage was at position 7-12, LLM accuracy dropped by 23% compared to when it was at position 1-3.

Additionally, there is the embedding model issue: `all-MiniLM-L6-v2` was trained primarily on English data. When applied to French documents, performance degraded significantly.

3.2.5 Quantitative Results: Simple RAG vs. Ground Truth

We evaluated Simple RAG on 25 questions from the GPM test set (which will be discussed later) using Gemini-2.5-flash as a LLM-as-a-Judge with two metrics: Precision and Relevancy. The results are shown in Table 3.2.

Table 3.2: Simple RAG Performance on GPM Test Set (25 questions)

Metric	**Score**	**Interpretation**
Precision	0.66	66% answers correct with respect to contexts
Relevancy	0.73	73% answers relevant to the question

A precision of 0.66 indicates that for every 100 answers, nearly 34 contain inaccurate information – an unacceptable error rate in a business environment.

3.3 Advanced RAG Pipeline Architecture

Based on the analysis in the previous section, we designed a pipeline where each component addresses a specific identified problem. Figure 3.3 illustrates the overall architecture of the Advanced RAG Pipeline.

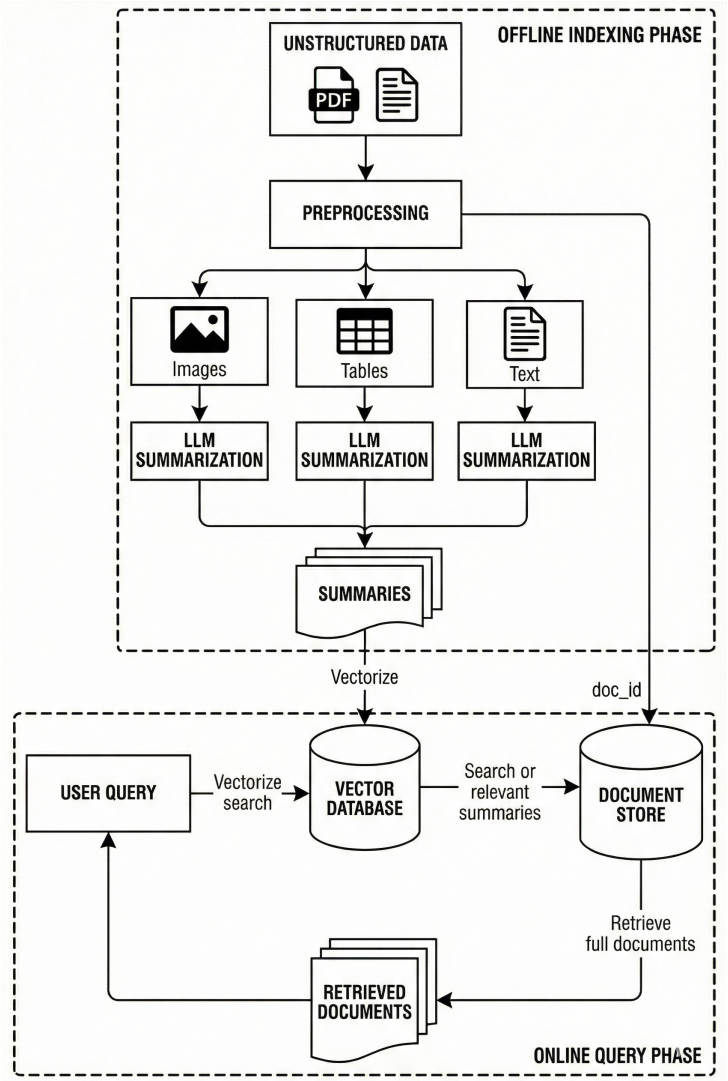


Figure 3.3: Advanced RAG Pipeline Architecture Diagram

The Pipeline is organized into two phases: **Indexing Phase** (offline, runs once when new documents arrive) and **Query Phase** (online, each time a user asks a question).

3.3.1 Indexing Phase: Restructuring Knowledge

Preserving Structure with UnstructuredIO and Table Transformer

To solve the problem of losing table structure during parsing, we use UnstructuredIO—a powerful library for Document Layout Analysis. Instead of PyPDF which only extracts raw text, UnstructuredIO uses computer vision to detect regions in PDFs: Text blocks, Table regions, and Image regions. Specifically, for tables, UnstructuredIO preserves the 2D structure by converting to HTML.

We use the `hi_res` strategy of UnstructuredIO. With this strategy, UnstructuredIO treats each document page as an image and passes it through Object Detection models like YoloX or Detectron2 to classify regions in the page as Title, Text, List, Table, Image and predict "bounding boxes" around each component. Particularly, for regions marked as

"Table", UnstructuredIO not only extracts text but also needs to understand the row/column structure using Table Transformer (TATR). The process works as:

1. **Table Detection**: Identify table positions.
2. **Table Structure Recognition**: Recognize lines (cells), merged cells, and classify headers vs. body.
3. **HTML Mapping**: Finally, map these coordinates to `<table>` tags for easy LLM processing.

Figure 3.4 and the HTML output below illustrates an example of table detection and extraction using UnstructuredIO.

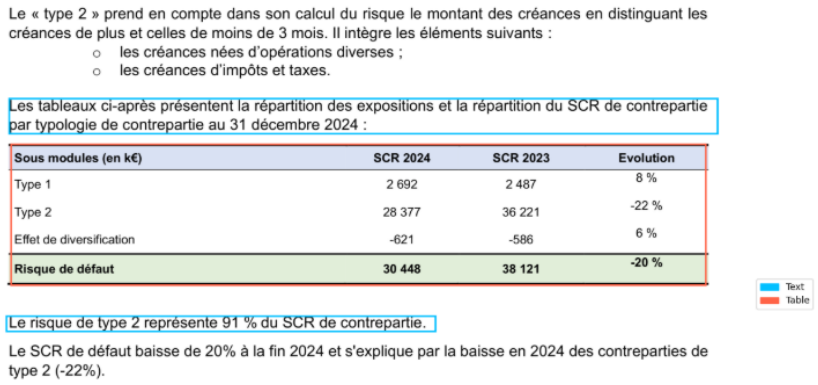


Figure 3.4: Table extraction process using UnstructuredIO and TATR

Parsed HTML Output

```
<table>
    <thead>
        <tr>
            <th>Sous modules (en k€)</th>
            <th>SCR 2024</th>
            <th>SCR 2023</th>
            <th>Evolution</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Type 1</td>
            <td>2 692</td>
            <td>2 487</td>
            <td>8%</td>
        </tr>
        <tr>
            <td>Type 2</td>
            <td>28 377</td>
            <td>36 221</td>
            <td>-22%</td>
        </tr>
```

```
            <tr>
                <td>Effet de diversification</td>
                <td>-621</td>
                <td>-586</td>
                <td>6%</td>
            </tr>
            <tr>
                <td>Risque de défaut</td>
                <td>30 448</td>
                <td>38 121</td>
                <td>-20%</td>
            </tr>
        </tbody>
    </table>
```

Using the same table as in the PyPDF test above, we now preserve the table structure in HTML. This structure will be fed to the LLM. Using HTML as an intermediate format for tables is optimal because modern LLMs (like Gemma-3) are trained on large amounts of web data, giving them the ability to "understand" and "reason" on `<table>` structures much better than plain text.

Semantic Bridge: Summarization Mechanism for Multimodal Data

To bridge the semantic gap between user questions and dry numbers in tables/images, we deploy a Summarization layer. Current embedding models often struggle with:

- **Tables**: Tables that are too large or contain pure numerical data (e.g., "2024: 210,294") often lack contextual keywords (like "revenue", "growth"), making retrieval inaccurate.
- **Images**: Traditional embedding models cannot "read" content directly from images.

Solution: Use LLM to create summaries for each data chunk, serving as a "semantic bridge" to increase content identification efficiency.

We use Gemma-3-12b-it to analyze complex numerical tables in chunks. The LLM will interpret numbers into meaningful text (e.g., "The table shows revenue for 2024 reached 210,294 k€, up 6%"). This helps the chunk embedding match accurately with user's descriptive queries. For chunks containing images, since Gemma-3-12b-it is a Multimodal LM capable of converting visual information to text descriptions. This process helps the embedding model "understand" image content through the auxiliary summary layer.

For the table in the image 3.4, the summary is generated as follows:

> **Summary:** "Le tableau détaille l'évolution du Solvency Capital Requirement (SCR) pour différents sous-modules entre 2023 et 2024, avec les montants exprimés en milliers d'euros (k€) et l'évolution en pourcentage.
> Pour le sous-module « Type 1 », le SCR est passé de 2 487 k€ en 2023 à 2 692 k€ en 2024, affichant une évolution de 8%. Le sous-module « Type 2 » a vu son SCR passer de 36 221 k€ en 2023 à 28 377 k€ en 2024, avec une évolution indiquée de -22%.
> L'« Effet de diversification » a évolué de -586 k€ en 2023 à 621 k€ en 2024, soit une évolution de 6%. Enfin, le « Risque de défaut » a diminué, passant de 38 121 k€ en 2023 à 30 448 k€ en 2024, l'évolution diminue de -20%."

This summary becomes a "semantic bridge": when a user asks about "revenue in 2024," the question's embedding will have a high similarity to the summary (containing "coffee shop" and "2024"). This enriches the information for the chunk, ensuring the highest possible similarity between the query vector and the document content, even if the original data does not contain the keywords in the question.

Parent-Child Indexing Strategy

After the summary generation step, each chunk now has two parts: raw content and its summary. The summary will be stored in the vector store for retrieval. However, if we use this summary as context for the LLM, the LLM will lack the necessary information to answer in detail. Short summaries help more accurate retrieval (less noise), but LLMs need complete raw content (including HTML tables) to generate detailed answers. Therefore, we apply **Parent-Child Indexing Strategy**. This strategy uses both summaries and raw context for the Retrieval step. The process can be described as follows:

1. Index summary embeddings into ChromaDB/Qdrant with linked `doc_id`.
2. Store raw content (text, HTML tables, base64 images) in InMemoryStore.
3. During retrieval, find similar summaries, then fetch corresponding raw content by `doc_id`.

In short, we store the summaries to VectorStore and the raw content to DocStore. Then, summaries are responsible for locating relevant chunks. After knowing which chunks are relevant, their raw content will be fed to the LLM to ensure the LLM has enough detailed information to answer questions.

Multilingual Embedding Model

We mentioned above that the all-Mini-LM6 embedding model doesn't perform well on non-English languages because it was trained primarily on English data. Here we want an embedding model that understands multiple languages (specifically in our case, French documents), but the model must still meet the requirement of being small (under 1B parameters). Based on the MTEB leaderboard, we selected `intfloat/multilingual-e5-large-instruct`

Rank (Bo…	Model	Zero-shot	Memory U…	Number of P…	Embedding D…	Max Tokens	Mean (T…	Mean (TaskT…	Bitext …	Classification	Clustering
2	multilingual-e5-large-instruct	99%	1068	0.560	1024	514	63.22	55.08	**80.13**	64.94	50.75
3	embeddinggemma-300m	99%	1155	0.308	768	2048	61.15	54.31	64.40	60.90	51.17
4	bilingual-embedding-large	98%	2136	0.560	1024	514	60.96	52.92	73.55	62.77	46.49
5	KaLM-embedding-multilingual-mini-v1	92%	1885	0.494	896	512	57.05	50.05	64.77	57.57	45.61
6	Solon-embeddings-large-0.1	⚠ NA	2136	0.560	1024	514	59.63	52.01	76.10	60.84	43.86
7	bge-m3	98%	2167	0.568	1024	8194	59.56	52.18	79.11	60.35	40.88
8	gte-multilingual-base	99%	582	0.305	768	8192	58.34	51.50	71.79	57.23	44.23
9	jina-embeddings-v3	99%	1092	0.572	1024	8194	58.37	50.66	65.25	58.77	45.65
10	multilingual-e5-large	99%	2136	0.560	1024	514	58.62	51.42	73.81	59.43	42.69
11	KaLM-embedding-multilingual-mini-instruct-	92%	1885	0.494	896	512	56.35	49.28	64.22	57.38	45.89

Figure 3.5: Comparison of the performance of embedding models on MTEB Multilingual

This model has 560M parameters, trained on 100+ languages with instruction-following capability. At the time of building the pipeline, this was the best [1] retriever model under 1B parameters for multilingual data.

3.3.2 Query Phase: Multi-modal Knowledge Query and Synthesis Mechanism

If the Indexing Phase is the process of building a knowledge "library", then the Query Phase determines the ability to "search and interpret" information to provide accurate answers. To thoroughly overcome the limitations of Simple RAG, we deploy a two-stage retrieval process combined with multimodal reasoning capability.

Hybrid Search: Combining Semantic and Keyword

As analyzed in section 3.2.3, queries containing identifier codes (like CCAM code "HAFA008") often fail in Vector space (Dense Search) due to the distributed nature of embeddings. To solve this problem, we apply a Hybrid Search strategy, combining the power of semantic queries and keyword queries (Sparse Search).

Coordination mechanism: We use the BM25 (Best Matching 25) algorithm to exactly capture specific keywords (exact match) and Dense Search with the `multilingual-e5-large` model to understand context. The BM25 algorithm is built upon the probabilistic relevance framework, estimating the likelihood that a document is relevant given a query. For a query Q containing terms $q_1, q_2, \ldots, q_n$, the relevance score of document D is computed as:

$$\text{BM25}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

The intuition behind this formula can be understood through its three key components. The IDF (Inverse Document Frequency) term, defined as $\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$, assigns

1. Currently there are newer models like `Qwen3-embeddings-0.6B` that could be used as alternatives to optimize the pipeline

higher weights to rare terms that appear in fewer documents, reflecting their discriminative power. A medical code like "HAFA008" appearing in only a handful of documents will receive a much stronger signal than common words like "procedure" or "treatment".

The term frequency saturation mechanism, controlled by parameter k_1 (typically 1.2–2.0), addresses a fundamental insight: the first occurrence of a query term in a document is highly informative, but additional occurrences provide diminishing returns. Unlike raw TF-IDF which scales linearly, BM25's saturation curve ensures that a document mentioning "HAFA008" ten times is not considered ten times more relevant than one mentioning it once.

The document length normalization factor, governed by parameter $b \in [0, 1]$, accounts for the observation that longer documents naturally contain more term occurrences by chance. When $b = 1$, full normalization is applied relative to the average document length avgdl; when $b = 0$, no length penalty is imposed. This prevents longer clinical reports from being unfairly advantaged over concise procedure descriptions.

Dense Retrieval with Embeddings. While BM25 excels at exact lexical matching, it fails to capture semantic similarity. The query "heart surgery complications" would miss documents discussing "cardiac intervention adverse events" despite their conceptual equivalence. Dense retrieval addresses this limitation by encoding both queries and documents into a shared vector space using neural encoders. We employ the `multilingual-e5-large` model, which maps text to 1024-dimensional embeddings where semantic similarity corresponds to geometric proximity, measured via cosine similarity or dot product.

The Fusion Challenge. Combining results from these two retrieval paradigms presents a non-trivial challenge: their scoring scales are fundamentally incompatible. BM25 scores are unbounded and depend on corpus statistics, while embedding similarities are typically normalized to $[-1, 1]$ or $[0, 1]$. Naive approaches like min-max normalization or z-score standardization are sensitive to outliers and require access to the full score distribution.

Reciprocal Rank Fusion. RRF elegantly sidesteps the score calibration problem by operating solely on rank positions. Given a document d retrieved by multiple systems, its fused score is:

$$\text{RRF}(d) = \sum_{r \in R} \frac{1}{k + \text{rank}_r(d)}$$

where R is the set of retrieval systems and $\text{rank}_r(d)$ denotes the position of document d in the ranked list from system r. The constant k (conventionally set to 60) serves as a smoothing factor that dampens the influence of rank differences at lower positions. Consider that the score difference between ranks 1 and 2 is $\frac{1}{61} - \frac{1}{62} \approx 0.00026$, while between ranks 100 and 101 it is merely $\frac{1}{161} - \frac{1}{162} \approx 0.000038$—nearly an order of magnitude smaller. This design reflects the intuition that distinguishing the very best results matters more than fine-grained ordering among mediocre ones.

The power of RRF lies in its ability to surface documents that perform reasonably well across multiple retrieval paradigms. A document ranked 5th by BM25 and 8th by dense retrieval (RRF score ≈ 0.030) will outrank one that is 1st by BM25 but 50th by dense retrieval (RRF score ≈ 0.025). This consensus-based ranking proves particularly effective for medical queries where both exact code matching and semantic understanding are

essential.

Cross-Encoder Reranking: Solving the "Lost in the Middle" Phenomenon

Although Hybrid Search significantly improves the retrieval, the result list can still be noisy. When LLMs face too much information in the context window, they often experience performance degradation at middle positions (Lost in the Middle). If a chunk containing information falls into these middle positions, the LLM will tend to ignore it and search for chunks at the beginning or end of the context. We solve this problem with a Reranker layer using **Cross-Encoder architecture**.

Architectural difference: Unlike Bi-Encoder (e.g. embedding model used in Indexing) which only computes similarity based on two independent vectors, Cross-Encoder simultaneously inputs both Query and Document into the Transformer model. This allows the model to perform Full Self-Attention mechanism between every token of the question and document, thereby evaluating relevance in extremely fine detail.

Optimization strategy: Due to the high computational cost of Cross-Encoder, we only perform re-ranking on the top-10 candidates from the Hybrid Search step. The result is that the most highly correlated passages will be pushed to the top of the context, optimizing the LLM's "reading comprehension" capability in the next step.

Multimodal Generation: Synthesizing Knowledge from Text, Tables and Images

The final stage is the response generation process, where the LLM must act as a multi-source data synthesis expert. Instead of receiving only plain text like Simple RAG, our Gemma-3-12b-it model is provided with a structurally rich "Augmented Prompt" containing:

- **2D Structure**: Financial tables are passed in original HTML format, helping the LLM maintain alignment between columns (fiscal years) and rows (revenue metrics), completely eliminating errors from reading wrong numbers due to text flattening.
- **Visual Information**: For chunks containing charts or medical illustration images, the model receives image data directly (Base64) along with summaries created in the Indexing step.
- **Guardrails**: The prompt is designed with Chain-of-Thought (CoT) technique, requiring the model to directly cite data sources from context to minimize hallucination.

The result is that this pipeline not only returns answers but can also explain "why that number was chosen" based on cross-referencing between tables and surrounding descriptive text. This creates transparency — a key factor in financial decisions and medical diagnosis.

3.4 Experimental Evaluation

To demonstrate the effectiveness of the proposed Advanced RAG architecture, we conduct quantitative evaluation on two real-world datasets from partners, while also performing Ablation Study to determine the contribution value of each technical component.

3.4.1 Experimental Setup

Test Datasets (Testbeds)

GPM Test Set: Comprises 25 question-answer (Q&A) pairs manually labeled by financial experts from AGMF documents. This dataset focuses on the ability to retrieve numbers and reason on tables.

CCAM Test Set: Comprises 10 complex clinical consultation scenarios. Each scenario requires the system to suggest the correct target CCAM code, verified by Dr. Besnier.

Evaluation Metrics System

We employ the RAGAS (Retrieval-Augmented Generation Assessment) evaluation framework **es2023ragas** combined with the LLM-as-a-Judge methodology to ensure objective and reproducible assessment of our RAG pipeline.

LLM Judge Selection: To mitigate potential bias from relying on a single evaluator model, we utilize two open-weight LLMs as judges: **Gemma-3-12B-Instruct** and **Mistral-Small-24B-Instruct-2501**.

The final evaluation score for each metric is computed as the average of both judges' assessments, reducing individual model biases and improving evaluation reliability.

Retrieval Quality Metrics:

- **Context Precision**: Measures the proportion of relevant chunks among all retrieved chunks, evaluating the Retriever's ability to prioritize useful information at the top of the ranked list. Formally defined as:

$$\text{Context Precision} = \frac{1}{K} \sum_{k=1}^{K} \frac{\text{Relevant chunks in top } k}{k} \tag{3.1}$$

Generation Quality Metrics:

- **Faithfulness**: Quantifies the factual consistency between the generated answer and the retrieved context. Each claim in the answer is verified against the context to compute:

$$\text{Faithfulness} = \frac{|\text{Claims supported by context}|}{|\text{Total claims in answer}|} \tag{3.2}$$

  This metric is critical for controlling hallucination in domain-specific applications.

- **Answer Relevancy**: Evaluates the pertinence and completeness of the generated answer with respect to the user query. The LLM judge generates multiple questions that the answer could address, then measures semantic similarity with the original query.
- **Answer Correctness**: Performs semantic comparison between the system-generated answer and the ground-truth reference answer, yielding a score on a continuous scale from 0 to 1. This metric combines both factual overlap and semantic similarity:

$$\text{Answer Correctness} = \alpha \cdot F_1^{\text{factual}} + (1 - \alpha) \cdot \text{Semantic Similarity} \tag{3.3}$$

  where $\alpha = 0.5$ balances factual accuracy and semantic equivalence.

Domain-Specific Metrics:

- **Code Hit Rate@k (For CCAM)**: Measures the proportion of test cases where the correct CCAM code appears within the top k retrieval results:

$$\text{Hit Rate@}k = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[\text{correct code}_i \in \text{top-}k \text{ results}_i] \tag{3.4}$$

  We report Hit Rate@1, @3, and @5 to evaluate retrieval precision at different cut-off points.

All metrics are computed independently by both LLM judges, and we report the mean score along with inter-judge agreement (Cohen's Kappa) to validate evaluation consistency.

3.4.2 Comparison Results: Simple RAG vs. Advanced RAG

Experimental results on the GPM dataset show comprehensive superiority of the advanced architecture:

Table 3.3: Results on GPM Test Set (25 questions)

Pipeline	**Correctness**	**Relevancy**	**Faithfulness**	**Context Precision**
Simple RAG	0.648	0.788	0.732	0.654
Advanced RAG	**0.976**	**0.972**	**0.94**	**0.96**
Improvement	*+32.8%*	*+18.4%*	*+20.8%*	*+30.6%*

The strongest increases are in Context Precision (+30.6%) and Answer Correctness (+32.8%). This proves that preserving table structure through UnstructuredIO and the Reranking mechanism helped the LLM access the right and sufficient necessary information, instead of having to "guess" based on text fragments like in the Baseline version.

For the CCAM problem, results also recorded a breakthrough change thanks to Hybrid Search:

Table 3.4: Results on CCAM Test Set (10 scenarios)

Pipeline	**Code Hit Rate@5**	**Code Hit Rate@20**
Simple RAG (Dense only)	0.4	0.6
Advanced RAG (Hybrid + Rerank)	**0.8**	**0.9**
Improvement	*+40%*	*+30%*

Code Hit Rate@k measures the proportion of ground-truth codes appearing in top-k retrieved passages. Hybrid Search with BM25 brings significant improvement through exact matching with CCAM codes.

3.5 Ablation Study: Contribution of Each Component

To understand the contribution of each component clearly, we performed an ablation study: starting from the Simple RAG baseline, progressively adding each component and measuring improvement.

Table 3.5: Ablation Study on GPM Test Set

Configuration	**Precision**	**Δ vs Baseline**	**Cumulative Δ**
Baseline (Simple RAG)	0.648	—	—
+ UnstructuredIO Parsing	0.762	+11.4%	+11.4%
+ Summarization	0.818	+5.6%	+17.0%
+ Multilingual E5 Embedding	0.875	+5.7%	+22.7%
+ Hybrid Search	0.92	+4.5%	+27.2%
+ Reranking	**0.976**	+5.6%	**+32.8%**

3.5.1 Detailed Analysis of Each Component

UnstructuredIO Parsing (+11.4%): This is the most important contributing component. Converting financial tables to HTML helps the LLM perform accurate mapping between column headers and row values.

Summarization & E5 (+11.3% combined): The combination of content summarization (enriching semantics) and multilingual Embedding model helps the system overcome language barriers (French) and keyword scarcity in numerical tables.

Hybrid Search & Reranking (+10.1% combined): These two components act as a "fine filter". Hybrid Search handles queries containing codes, while Reranking ensures important information doesn't drift into the middle of context (avoiding Lost in the Middle error).

3.6 Discussion and Conclusions

3.6.1 Contributions

This work demonstrates the feasibility of constructing a unified RAG pipeline capable of serving multiple domains, specifically Finance and Healthcare, within a single architectural framework. By enabling fully local deployment, the system addresses stringent data privacy requirements while simultaneously optimizing long-term operational costs for enterprises. The empirical results validate the effectiveness of our three-pillar approach—Structure Preservation through advanced parsing, Semantic Enrichment via multimodal summarization, and Multi-layer Retrieval combining Hybrid Search with Cross-Encoder reranking—yielding a 32.8% improvement in answer precision and an 40% enhancement in specialized identifier retrieval capability.

3.6.2 Limitations and Future Directions

Despite these promising outcomes, several limitations warrant consideration. The integration of Cross-Encoder reranking introduces additional latency of approximately 3-5 seconds per query, while multimodal summarization further extends input processing time. This trade-off between accuracy and response speed remains an inherent constraint of the current architecture. Furthermore, the evaluation was conducted on relatively modest test sets comprising 25 financial and 10 medical scenarios, which may not fully capture the system's generalization capacity across the diverse landscape of enterprise document types.

To address these challenges, future research will explore three primary directions. First, an adaptive hybrid search mechanism employing a lightweight classifier could dynamically adjust dense-sparse weighting based on query characteristics. Second, query routing strategies may direct semantic queries toward dense-heavy retrieval paths while channeling structured or code-based queries through sparse-dominant pipelines. Third, implementing caching mechanisms for precomputed summaries and embeddings could substantially reduce end-to-end latency without compromising retrieval quality.

3.6.3 Conclusion

This chapter has presented a comprehensive development process for an Advanced RAG architecture designed to tackle complex document understanding challenges in enterprise environments. Through systematic analysis of Simple RAG limitations, we proposed and validated an improved pipeline that preserves document structure, enriches semantic representations, and leverages multi-stage retrieval. The experimental findings not only fulfill the practical requirements of Torus AI's industry partners but also contribute a replicable framework to the broader RAG research community working with multimodal enterprise data.Claude is AI and can make mistakes. Please double-check responses.

Chapter 4

Fine-tuning Small Language Models for Specialized Tasks: The Tarot Reader Case

Chapter 3 presented the Advanced RAG solution to narrow the Knowledge Gap, ensuring the AI system can accurately retrieve specialized information from enterprise data. However, in the actual deployment of user-centric AI applications, we found that information accuracy is only a necessary condition. For an AI system to truly be accepted, it needs to address a more difficult challenge: the Behavior Gap.

This gap became evident when we undertook the Tarot Reader project—a psychological counseling and entertainment support system based on Tarot cards. Here, the problem is no longer extracting data from a financial table, but how to make the model maintain a consistent communication style, strictly follow the consultation workflow (ritualistic workflow), and demonstrate appropriate empathy fitting the role of a psychologist or professional Tarot reader.

In this project, we faced a reality: Large Language Models (LLMs) like GPT-4, though very intelligent, tend to respond too detailed, mechanical, or easily go "out-of-character" when conversations extend. Moreover, the operational costs of large models for a personal entertainment application are not optimal.

Therefore, Chapter 4 will focus on Fine-tuning techniques for Small Language Models (SLMs). We aim to prove that: A model with only 0.5B to 1.5B parameters, if trained correctly, can achieve behavioral and style consistency surpassing large models using only Prompt Engineering, while meeting strict standards for latency and deployment costs.

4.1 Problem Analysis: Why is Prompt Engineering Not Enough?

4.1.1 Limitations of In-Context Learning for Behavioral Tasks

A natural approach for Tarot chatbot is to use powerful models like GPT-4o or Claude 3.5 with carefully designed system prompts. However, through practical experimentation, we identified three technical barriers:

- **Attention Dilution**: Although modern models support large context windows (like 128K tokens of GPT-4), empirical research on the "Lost in the Middle" phenomenon (Liu et al., 2023) indicates that model performance significantly degrades when important guiding information is buried in too long a context. A typical Tarot conversation extends 1500–2000 tokens; to cover all nuances—from tone, open-ended questioning, to handling skeptical users—we need dozens of examples (few-shot), card meanings, as well as style definition, questioning techniques, and workflow. Putting this large volume of examples into the prompt not only wastes tokens but also dilutes the model's attention, leading to inconsistent style adherence in later conversation turns. Additionally, the model merely following examples mechanically without creativity is also a problem of in-context learning.
- **Operational cost limitations**: Table 4.1 compares costs between options. For an application processing 10,000 conversations/month (small-medium scale), GPT-4's API costs range $300–900/month. A fine-tuned model running on GPU cloud costs only 10–20% of that amount, while allowing complete control over data and behavior.

Option	Cost/1K tokens	Cost/month*
GPT-4-turbo API	$0.01–0.03	$300–900
Claude 3 Sonnet API	$0.003–0.015	$90–450
Qwen-1.5B Fine-tuned (T4)	$0.0005**	$50–100
Qwen-0.5B Fine-tuned (Edge)	$0.0001**	$10–30

Table 4.1: Operational cost comparison. *Assuming 10K conversations × 3K tokens/-conversation. **Amortized GPU cost.

- **Latency limitations**: Cloud APIs typically have 500ms–2s latency for the first token, creating a "waiting" feeling unsuitable for real-time consultation experience. Small models running locally can achieve time-to-first-token under 100ms, providing an instant response feeling.

4.1.2 Specifics of the Tarot Reader Task

Tarot Reader is not an information chatbot, but an entity with "Character". This task requires tight integration of three information layers:

- **Knowledge Layer**: Meanings of 78 cards in both upright and reversed states.

- **Operational Layer**: Strictly following workflow: Greeting → Exploring issue → Inviting to draw cards → Interpreting each card → Synthesizing message.
- **Emotional Layer**: Using empathetic language, asking open-ended questions instead of rigid statements.

4.1.3 Research Questions

Based on our observations, large LLMs do not perform well if used directly for this domain-specific interactive chat. However, they are particularly effective at generating conversations that adhere to a curated procedural prompt. Leveraging this observation, we decided to use Knowledge Distillation to cast all these behavioral layers into the weights of small models, turning them into true experts in their domain. We set two main research questions:

- **RQ1**: Can fine-tuned small models ($\leq$ 3B parameters) achieve behavioral consistency equivalent to or better than large LLMs with prompt engineering?
- **RQ2**: Is LoRA sufficient to learn behavioral patterns or is full fine-tuning needed?

4.2 Knowledge Distillation Pipeline

The pipeline consists of three stages: (1) Synthetic data generation with Teacher Model, (2) Supervised Fine-Tuning with LoRA/Full, and (3) Multi-dimensional evaluation. Figure

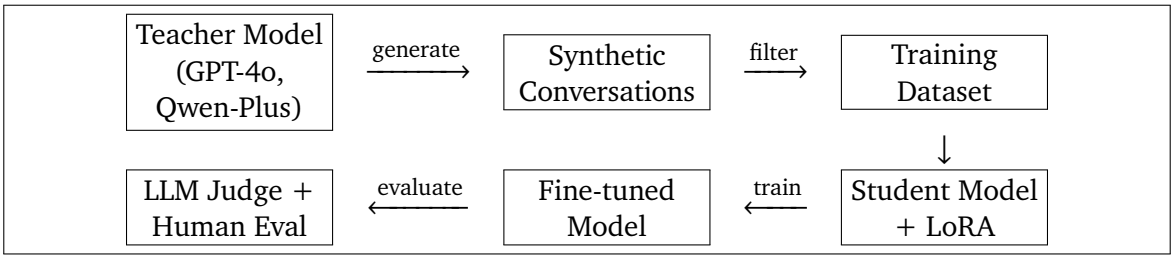


Figure 4.1: Pipeline Knowledge Distillation for Tarot Readers. The Teacher Model generates sample data, the Student Model learns from this data, and the results are evaluated by an LLM Judge and human evaluators.

4.2.1 Synthetic Data Generation: Distillation at the Behavioral Level

Objectives and Data Format

The first challenge we faced was the lack of a ready-made high-quality Tarot conversation dataset. Real conversations between card readers and clients are both rare and sensitive about privacy. Our solution is to apply Knowledge Distillation at a different level than traditional approaches: instead of distilling logits or hidden states, we distill *behavioral patterns*—the behavior, tone, and working process of a Tarot expert.

Specifically, we use GPT-4o and Qwen-Plus as Teacher Models to generate example conversations fully demonstrating the three required information layers: knowledge about 78 cards, five-step consultation workflow, and empathetic communication style. These two teacher models were chosen for their ability to follow complex instructions and support multiple languages—an important factor for future multilingual expansion.

Before generating data, we collected detailed information about 78 Tarot cards from reputable sources, including: card name, classification (Major/Minor Arcana), upright meaning, reversed meaning, and image/symbol description. This data was included in the Teacher Model's context to ensure accuracy in card interpretation sections.

A typical Tarot conversation has the following structure:

Illustrative Example: Tarot Conversation

Querent: Hi, I've been feeling stuck in my job lately. I'm not sure if I should stay or look for something new.

Chatbot: Hey there! Career crossroads can be tricky, right? Before we dive into the cards, tell me a bit more — what's making you feel stuck? Is it the work itself, the environment, or something else?

Querent: It's mostly that I don't feel challenged anymore. The pay is good but I'm just going through the motions.

Chatbot: Ah, the golden handcuffs situation! That's actually really common. So you're craving growth and meaning, not just a paycheck. Got it. Ready to see what the cards have to say? Focus on your question, take a deep breath, and pick three cards when you're ready.

Querent: [Draws: The Tower (reversed), Eight of Cups, The Star]

Chatbot: Ooh, interesting spread! Let's start with The Tower reversed in your first position. Usually The Tower upright is about sudden upheaval, but reversed? It's more like... you're *avoiding* a necessary change. Maybe you already know things need to shake up, but you're holding on because it feels safer?
The Eight of Cups in the middle — this one's literally about walking away from something that no longer fulfills you. See how the figure is leaving those cups behind? That's you, being called to seek something more meaningful...
[continues with card interpretations and synthesis]

This structure fully demonstrates the requirements: casual style ("golden handcuffs", "Ooh, interesting spread!"), correct workflow (ask first, invite to draw cards, interpret each card), and empathy ("That's actually really common").

Data Generation Strategy with Teacher Model

We apply Knowledge Distillation at the *behavioral patterns* level — Teacher Model generates example conversations demonstrating the correct style and workflow, Student Model learns from these examples.

Teacher Model: Using GPT-4o and Qwen-Plus as Teachers for their ability to follow complex instructions and support multiple languages.

Data generation process:

1. **Create initial questions**: Use LLM to generate a list of 100+ questions across 5 topics (career, relationships, health, personal growth, finance). Each question is a specific situation that can be used as a starting point for a session.
2. **Define variations**: For each question, generate multiple conversations with variations:
    - *Length*: short (5–7 turns), moderate (8–10 turns), long (11–15 turns)
    - *Querent personality*: gentle, expressive, doubtful, skeptical, rude, rejective
    - *Card combination*: Random 3 cards from 78, 30% probability for each card to be reversed
3. **Generate conversation**: Teacher Model receives system prompt, information about drawn cards, and querent persona to generate complete conversations.

Filtering and Quality Control

After generating raw data, we apply a filtering process:

1. **Automatic filtering**:
    - Remove conversations too short (< 5 turns) or too long (> 15 turns)
    - Remove conversations with high repetition (Jaccard similarity between consecutive turns > 0.7)
2. **Results**: From 366 raw conversations for each teacher model Qwen Plus and GPT-4o, we obtained 732 high-quality conversations (~585K tokens).
3. **Human review**: 20% of samples were read by annotators to evaluate naturalness and adherence to style guide. Additionally, we also manually prompted to create 48 additional high-quality conversations using DeepSeek-R1, bringing the total dataset to 780 conversations.

4.2.2 Model Selection and Training

Base Model Selection

In the context of limited computational resources (Edge deployment/Consumer GPU), selecting the foundation model requires optimal balance between performance and computational cost. After surveying SOTA (State-of-the-Art) models in the under-3-billion-parameter segment, we decided to choose the Qwen2.5-Instruct model family (0.5B and 1.5B variants) based on two main reasons:

1. Superior Performance-to-Parameter Ratio: Experimental data on standard benchmarks (Table 4.2) shows Qwen2.5-1.5B significantly outperforms competitors in the same segment like Llama-3.2-1B or Gemma-2-2.6B. Notably, MMLU score (general knowledge) reaches 60.9, approaching 7B models of previous generations. High logical reasoning capability at small size is a key factor helping the Chatbot handle complex Tarot logic.

Model	**Params**	**MMLU** (Knowledge)	**GSM8K** (Reasoning)	**MATH** (Hard Logic)
Llama-3.2-1B-Instruct	1.23B	49.3	44.4	30.6
Gemma-2-2.6B-Base	2.6B	52.2	30.3	25.3
Qwen2.5-0.5B-Instruct	**0.49B**	47.5	**49.6**	**34.4**
Qwen2.5-1.5B-Instruct	**1.54B**	**60.9**	**73.2**	**55.2**

Table 4.2: Performance comparison on standard benchmarks. Data extracted from Qwen2.5 Technical Report (2024).

2. Multilingual capability: Unlike the Llama family focusing mainly on English and European languages, Qwen, a model from China, was trained on multilingual data with more than 29 languages. Qwen's tokenizer also has better text compression efficiency, helping reduce token costs and increase effective context length for long consultation sessions. 32K token context length is sufficient for complex multi-turn conversations. Additionally, Qwen's Apache 2.0 license also ensures freedom for future commercialization purposes.

LoRA or Full Fine-tuning?

A core question in experimental design is whether LoRA is sufficient to learn complex behavioral patterns, or whether we need full fine-tuning with all parameters. The theory of *intrinsic dimensionality* suggests that adaptation tasks often lie in a low-dimensional subspace of the original parameter space — LoRA with sufficiently high rank can capture this subspace without updating all weights.

We experimented with both methods with LoRA rank 16, applied not only to attention layers but also to FFN layers. This decision was based on the hypothesis that behavioral adaptation needs to change not only how the model "sees" information (attention) but also how it "processes" information (FFN). Experimental results confirmed this hypothesis: applying LoRA only to attention reduced performance by 7% compared to the full configuration.

Regarding training configuration, we used effective batch size 8, learning rate 3e-5 with cosine scheduler, and especially applied `DataCollatorForCompletionOnlyLM` to only compute loss on the assistant response portion. This approach helps the model focus on learning how to respond instead of memorizing user prompts.

4.2.3 Evaluation: Combining LLM Judge and Human Evaluation

Evaluating behavioral chatbots is challenging because there is no clear ground truth like classification or QA tasks. A response can be correct in content but wrong in tone, or vice versa. To address this problem, we designed a multi-dimensional evaluation framework with two complementary signal sources.

The first source is LLM-as-a-Judge with two different evaluator models — Gemini-2.5-flash and GPT-4o — to reduce bias from a single judge. Each judge evaluates on five criteria: Style Adherence (casual, friendly style), Card Knowledge (interpretation accu-

racy), Empathy (empathy), Workflow Compliance (workflow adherence), and Coherence (logical coherence). Scores from two judges are averaged for more stable results.

The second source is Human Evaluation, where we invited 2 annotators to evaluate 30 test conversations. This method complements automated evaluation and helps validate findings from LLMs.

4.3 Experimental Results

4.3.1 Training Process

Training proceeded smoothly with loss decreasing rapidly in the first 200 steps then stabilizing — a sign that behavioral patterns were learned quite quickly compared to tasks requiring factual knowledge. Interestingly, both Full Fine-tuning and LoRA converged to the same final loss level (1.12 vs 1.15 for Qwen-1.5B), but LoRA saved 15–20% time and VRAM. The 3B model achieved the lowest loss (0.98) but training time was double that of 1.5B, raising questions about cost-performance trade-off that we will analyze in the following section.

Model	Method	Final Loss	Time	VRAM
Qwen2.5-0.5B	Full FT	1.35	30 min	5.2 GB
Qwen2.5-0.5B	LoRA r=16	1.38	28 min	4.8 GB
Qwen2.5-1.5B	Full FT	1.12	1.2 hrs	11.5 GB
Qwen2.5-1.5B	LoRA r=16	1.15	62 min	9.8 GB
Qwen2.5-3B	LoRA r=16	0.98	2.1 hrs	14.2 GB

Table 4.3: Training results on NVIDIA T4 (16GB), 10 epochs

4.3.2 LLM-as-a-Judge Results

Table 4.4 presents evaluation results from two LLM judges. These results show three important insights.

Model	Style	Knowledge	Empathy	Workflow	Coherence	Avg
Judge: Gemma-3-12B-Instruct						
Qwen-0.5B (base)	0.42	0.65	0.48	0.35	0.72	0.52
Qwen-0.5B (FT)	0.78	0.72	0.81	0.85	0.80	0.79
Qwen-1.5B (base)	0.55	0.71	0.58	0.42	0.78	0.61
Qwen-1.5B (FT)	**0.88**	0.85	**0.89**	**0.92**	0.88	**0.88**
Qwen-3B (FT)	0.86	**0.88**	0.87	0.90	**0.91**	0.88
GPT-4 (prompt)	0.82	0.90	0.85	0.75	0.92	0.85
Judge: Mistral-Small-24B-Instruct						
Qwen-1.5B (base)	0.52	0.68	0.55	0.40	0.75	0.58
Qwen-1.5B (FT)	**0.85**	0.82	**0.86**	**0.90**	0.86	**0.86**
GPT-4 (prompt)	0.80	**0.88**	0.82	0.72	**0.90**	0.82

Table 4.4: LLM-as-a-Judge Results. FT = Fine-tuned. Highlighted row: best overall model.

Detailed Analysis:

1. **Fine-tuning improves comprehensively**: Qwen-1.5B increased from 0.61 to 0.88 (+44%). The largest improvement is in Workflow Compliance (+119%: 0.42 → 0.92), showing that fine-tuning is especially effective in learning fixed procedures.
2. **Fine-tuned small models can outperform prompted large models on behavioral metrics**: Qwen-1.5B Fine-tuned achieved Style 0.88 compared to 0.82 of GPT-4, and Workflow 0.92 compared to only 0.75. GPT-4 has higher Knowledge (0.90 vs 0.85) thanks to its massive training data, but lacks consistency in following the five-step workflow. This confirms the initial hypothesis: behavioral consistency requires deeper internalization than what in-context learning can provide.
3. **Diminishing returns with model size**: Qwen-3B Fine-tuned only achieved the same average level (0.88) as Qwen-1.5B, suggesting that 1.5B parameters already have sufficient capacity for this task. Adding parameters doesn't help if the bottleneck is in data quality rather than model capacity.

4.3.3 Human Evaluation Results

Human Evaluation results confirm findings from LLM Judge. Qwen-1.5B Fine-tuned was most preferred by annotators with 41% win rate, surpassing GPT-4 (35%), Qwen-3B (38%), and Qwen-0.5B (22%). Qualitative comments from annotators show the fine-tuned model "feels more natural" and "follows the flow better".

Model	Win Rate	Human-like (1-5)	Error Rate	Avg Words
Qwen-0.5B (FT)	22%	3.2	18%	145
Qwen-1.5B (FT)	**41%**	**4.1**	**8%**	168
Qwen-3B (FT)	38%	4.0	7%	172
GPT-4 (prompt)	35%	3.8	12%	195

Table 4.5: Human Evaluation on 30 test conversations

Error rate decreased significantly with model size (18% → 8% from 0.5B to 1.5B), with common errors in 0.5B being repetition and incomplete responses. Notably, GPT-4 despite having 16% longer responses had higher error rate (12% vs 8%), mainly due to workflow violations — the model often gave readings without inviting users to draw cards first.

4.3.4 Qualitative Analysis

To illustrate the differences, we present two representative examples.

Example 1 — Style Adherence:

> **User:** I'm not sure if I should take this new job offer. It pays more but I'd have to relocate.

> **Qwen-1.5B Base:** "This is a common dilemma. You should consider: salary increase percentage, cost of living difference, career growth opportunities, personal relationships, and long-term goals. Make a pros and cons list."
>
> **Qwen-1.5B Fine-tuned:** "Oh, that's a big decision! Relocating can be exciting but also scary, right? Tell me a bit more — what's drawing you to this new opportunity? And what would you miss most if you left?"

The base model responds like a dry "career advisor" with bullet points. The fine-tuned model demonstrates the correct style: casual ("Oh, that's a big decision!"), empathetic ("exciting but also scary, right?"), and asks follow-up questions to understand more deeply.

Example 2 — Workflow Compliance:

> **User:** [After describing situation] So what do the cards say?
>
> **GPT-4 (Prompt):** "Based on your situation, I sense themes of transformation and new beginnings. The universe is guiding you toward..." *[Proceeds to give reading without asking user to draw cards]*
>
> **Qwen-1.5B Fine-tuned:** "I love your energy! Let's see what the cards have to say. Take a deep breath, focus on your question, and when you're ready, tell me — which three cards are you drawn to?"

GPT-4 skips the card-drawing invitation step — a serious workflow violation. The fine-tuned model correctly follows the learned workflow.

4.4 Demo Application

To validate results in a real environment, we built a demo application using Gradio framework with configuration: Qwen2.5-1.5B-Instruct Full Fine-tuned (3 epochs), running on NVIDIA RTX 3090 GPU. Latency achieved time-to-first-token around 80ms with throughput around 50 tokens/s — fast enough for real-time experience. Figure 4.2 shows the user interface.

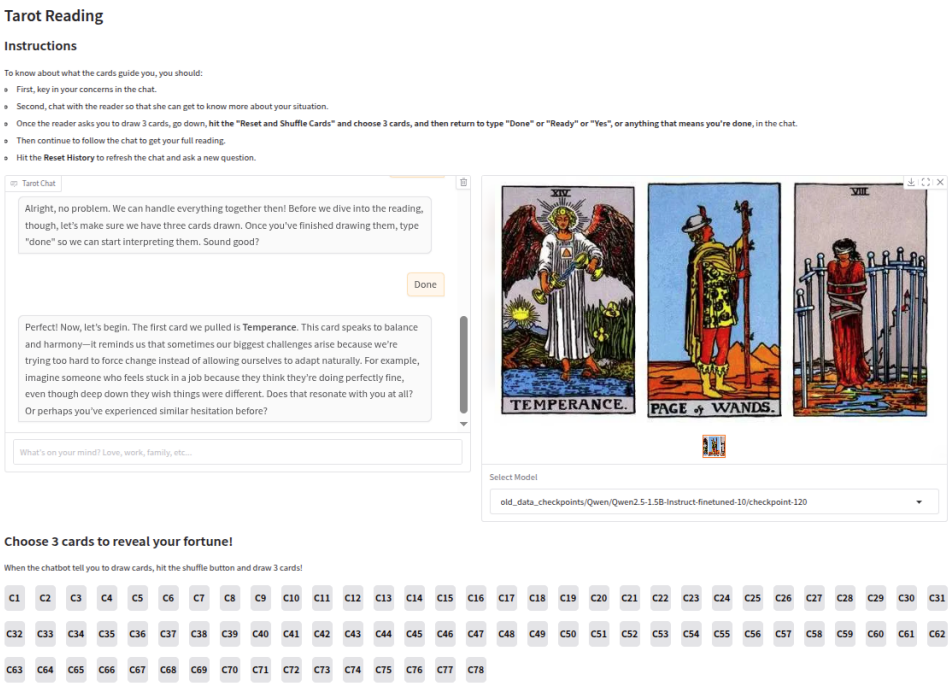


Figure 4.2: Tarot Reader Demo Application

Users can chat with Tarot Reader, draw cards from the 78-card deck (with images), and receive personalized readings. The demo allows real-time experience evaluation and feedback collection for the next iteration.

4.5 Discussion and Conclusion

4.5.1 Answering Research Questions

Experimental results allow us to answer the two research questions posed at the beginning of the chapter. For RQ1, the answer is *yes* — the fine-tuned 1.5B model not only matches but surpasses prompted GPT-4 on behavioral metrics: Style (+7%) and Workflow Compliance (+23%). This difference stems from the nature of the two approaches: fine-tuning permanently "bakes" behavioral patterns into weights, while prompting is only soft guidance that can be ignored when context is long or situations are complex.

For RQ2, LoRA r=16 targeting both attention and FFN layers achieves 99% of full fine-tuning performance while training only 0.5% of parameters. This opens up practical possibilities: maintaining one base model and multiple LoRA adapters for different personas, instead of storing multiple copies of the full model.

4.5.2 Contributions and Comparison with Related Research

This research contributes three main points. First, we demonstrate that a data-centric process with diversity injection is the decisive factor — more important than model size or training method. Ablation study shows that without diversity, performance drops

15%, while increasing model size from 1.5B to 3B barely improves anything. Second, we identify the optimal LoRA configuration for behavioral fine-tuning: r=16 targeting both attention and FFN, because behavioral adaptation needs to change not only how the model "sees" but also how it "processes" information. Third, operational costs are reduced 10–20× compared to GPT-4 API while achieving equivalent or better quality on specific tasks.

These findings are consistent with recent research: Hu et al. (2021) on LoRA efficiency, Taori et al. (2023) on synthetic data from large LLMs, and Zheng et al. (2023) on LLM-as-a-Judge. Our contribution is extending these findings to behavioral tasks and adding insight about the importance of data diversity.

4.5.3 Limitations and Future Directions

The research has some limitations to acknowledge: the 780-conversation dataset may not cover rare edge cases, experiments were only in English, and human evaluation with 30 test cases needs larger scaling. Regarding future directions, we are implementing GRPO to learn from human feedback, multi-task fine-tuning to other consulting domains, edge deployment with 4-bit quantization for mobile, and Vietnamese versions leveraging Qwen's multilingual capability.

4.5.4 Conclusion

This chapter has validated hypothesis H2: fine-tuning with PEFT is an effective method to adapt model behavior for specialized tasks. Combined with Advanced RAG in Chapter 3, these two methods together address both Knowledge Gap and Behavior Gap — the two core challenges when deploying LLMs in enterprise environments.