Chapter 4

Sprint 2

4.1 Introduction

This sprint represents a critical phase in the overall agile roadmap for Our platform, focusing on establishing robust backend infrastructure, comprehensive LLM evaluation frameworks, and domain-specific fine-tuning capabilities. Following the CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology, this two-week development cycle systematically addressed the core technical foundations required for aviation-specific AI applications.

The sprint tackled five essential components within the CRISP-DM framework: model evaluation systems, fine-tuning workflows, RAG integration pipelines, advanced prompt engineering techniques, and comprehensive benchmarking strategies. This structured approach ensured both technical rigor and alignment with aviation operational requirements while maintaining a focus on measurable outcomes and iterative improvement.

4.2 Business Understanding

4.2.1 Objectives

The aviation domain presents unique challenges that motivated this sprint's technical approach:

Domain-Specific LLM Requirements: Standard large language models lack the specialized knowledge required for aviation contexts, including precise terminology, regulatory compliance requirements, and operational procedures. The need for domain-specific enhancement became evident through initial testing, where base models struggled with aviation acronyms, procedural accuracy, and regulatory citations.

Real-Time Performance Imperatives: Aviation operations demand accurate, real-time responses for safety-critical decisions, compliance verification, and operational analytics. Response times exceeding 10 seconds create operational bottlenecks, while inaccurate information poses safety risks.

Performance Bottlenecks with Base LLMs: Initial assessments revealed that general-purpose models demonstrated limited performance on domain-specific queries, particularly struggling with completeness and relevance.

4.2.2 Target Application Scenarios

This sprint focused on addressing key categories of domain-specific interactions that exposed the limitations of general-purpose language models and underscored the need for customized enhancements:

- **Procedural Knowledge Retrieval**: Scenarios involving the extraction of highly structured procedural and compliance-related content. These interactions require accurate interpretation of technical documentation and context-sensitive procedures.
- **Regulatory Interpretation and Explanation**: Situations in which responses must reflect not just factual recall but also a nuanced understanding of industry-specific legal and operational standards.
- **Analytical and Decision-Support Queries**: Use cases involving the synthesis of operational data and domain expertise to support performance monitoring, anomaly detection, and strategic planning.

4.3 Data Understanding

4.3.1 Data Sources

Table 4.1: Overview of Data Sources Used During the Sprint

Data Category	Description
Training/Fine-tuning Data	Domain-relevant documentation such as operational manuals, technical guides, and regulatory content formed the foundation for model specialization. These sources contributed to improved contextual alignment and terminology adaptation.
Evaluation Dataset	A curated set of domain-specific question-answer pairs was used to test and refine model outputs. These examples were structured to represent a broad range of typical user interactions and domain-relevant tasks.
RAG Knowledge Base	A document repository comprising structured and unstructured sources, including manuals, wikis, and technical documentation. Materials were preprocessed to preserve metadata and facilitate effective retrieval and citation within the RAG pipeline.

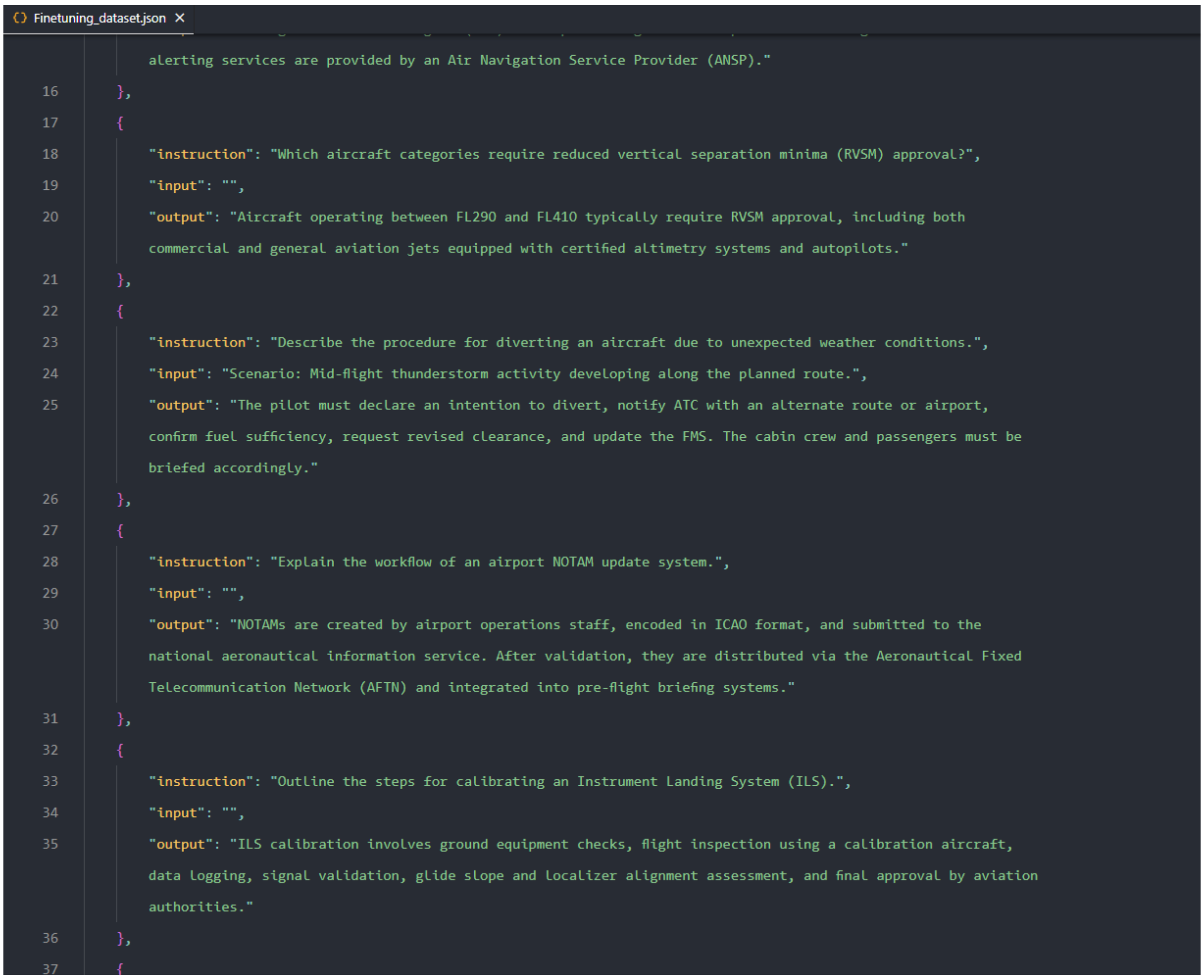
4.3.2 Exploratory Data Analysis (EDA)

Exploratory analysis of the available data revealed several key insights. First, there was significant variation in response length requirements, with some queries necessitating concise factual answers and others requiring more elaborate procedural explanations. This observation informed strategies for content segmentation and context window management. Additionally, the initial data review uncovered uneven domain coverage across specialized subtopics, highlighting the need for more comprehensive and balanced data acquisition in future phases. Lastly, a large portion of essential vocabulary was composed of domain-specific terms and abbreviations that were poorly represented in general-purpose datasets, reinforcing the necessity of targeted fine-tuning to enhance model performance in this context.

4.4 Data Preparation

4.4.1 Dataset Construction

Fine-Tuning Dataset Structure: The fine-tuning dataset was formatted using an instruction-tuning paradigm, where each example is defined by an `instruction`, optional `input`, and the corresponding `output`. This structure enables more flexible learning across both open-ended and scenario-based tasks. Examples included detailed procedures (e.g., aircraft diversion in adverse weather), system explanations (e.g., NOTAM update workflow), and calibration protocols. Entries were manually reviewed to ensure operational realism, terminological consistency, and alignment with domain practices.

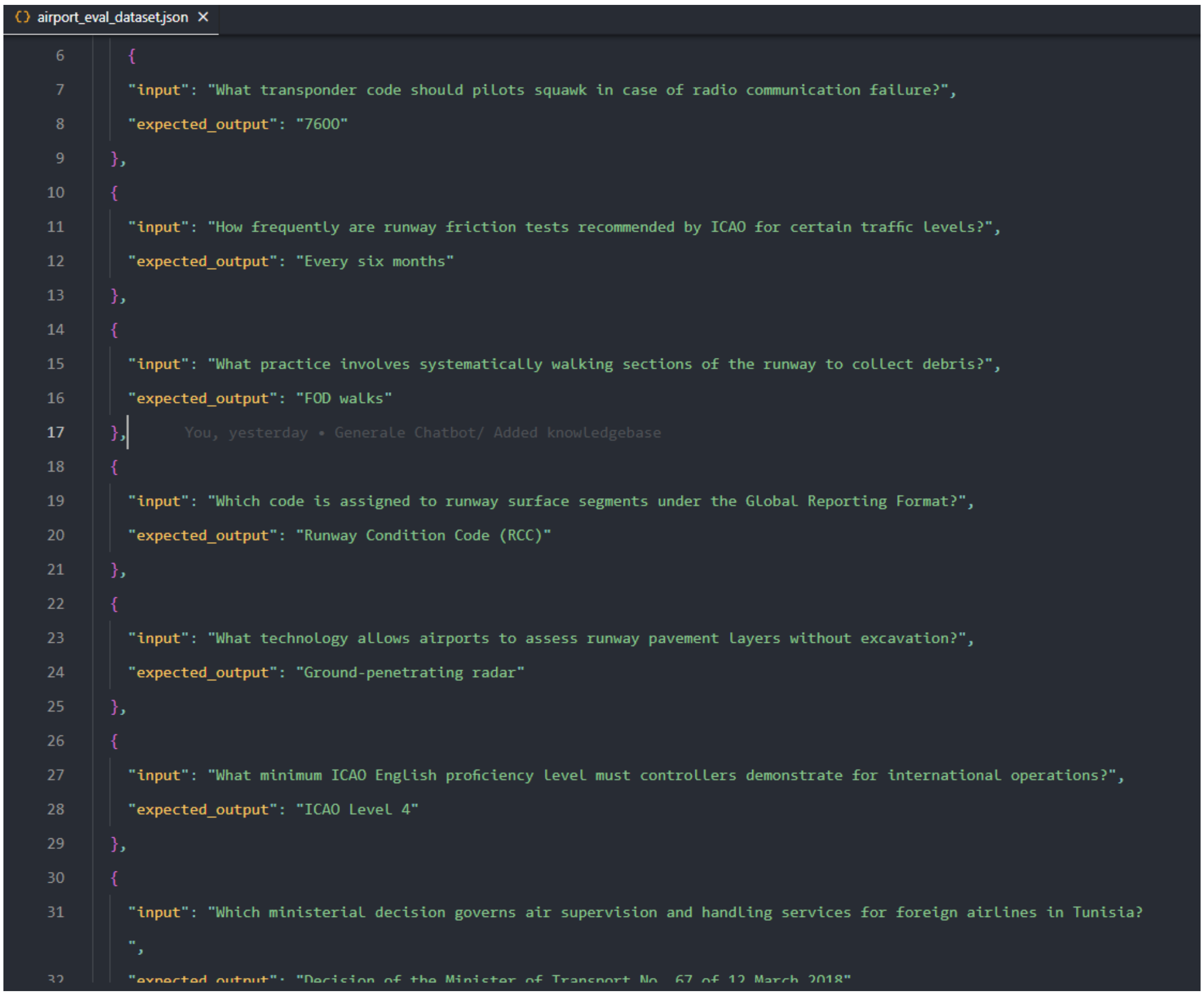


Finetuning_dataset.json

```
        alerting services are provided by an Air Navigation Service Provider (ANSP)."
    },
    {
        "instruction": "Which aircraft categories require reduced vertical separation minima (RVSM) approval?",
        "input": "",
        "output": "Aircraft operating between FL290 and FL410 typically require RVSM approval, including both
        commercial and general aviation jets equipped with certified altimetry systems and autopilots."
    },
    {
        "instruction": "Describe the procedure for diverting an aircraft due to unexpected weather conditions.",
        "input": "Scenario: Mid-flight thunderstorm activity developing along the planned route.",
        "output": "The pilot must declare an intention to divert, notify ATC with an alternate route or airport,
        confirm fuel sufficiency, request revised clearance, and update the FMS. The cabin crew and passengers must be
        briefed accordingly."
    },
    {
        "instruction": "Explain the workflow of an airport NOTAM update system.",
        "input": "",
        "output": "NOTAMs are created by airport operations staff, encoded in ICAO format, and submitted to the
        national aeronautical information service. After validation, they are distributed via the Aeronautical Fixed
        Telecommunication Network (AFTN) and integrated into pre-flight briefing systems."
    },
    {
        "instruction": "Outline the steps for calibrating an Instrument Landing System (ILS).",
        "input": "",
        "output": "ILS calibration involves ground equipment checks, flight inspection using a calibration aircraft,
        data logging, signal validation, glide slope and localizer alignment assessment, and final approval by aviation
        authorities."
    },
    {
```

Figure 4.1: Sample entries from the fine-tuning dataset using instruction-input-output structure.

Evaluation Dataset Preparation: The evaluation set was constructed as a JSON array of domain-specific question-answer pairs. Each entry includes an `input` field representing a natural-language query and an `expected_output` field capturing the target response. The dataset emphasizes factual recall and regulatory knowledge across diverse aviation subdomains. It served as the basis for benchmarking using both automated scoring via LLM-as-a-Judge and structured human evaluation, supporting reproducible comparisons between model variants.



```
    {
      "input": "What transponder code should pilots squawk in case of radio communication failure?",
      "expected_output": "7600"
    },
    {
      "input": "How frequently are runway friction tests recommended by ICAO for certain traffic levels?",
      "expected_output": "Every six months"
    },
    {
      "input": "What practice involves systematically walking sections of the runway to collect debris?",
      "expected_output": "FOD walks"
    },      You, yesterday • Generale Chatbot/ Added knowledgebase
    {
      "input": "Which code is assigned to runway surface segments under the Global Reporting Format?",
      "expected_output": "Runway Condition Code (RCC)"
    },
    {
      "input": "What technology allows airports to assess runway pavement layers without excavation?",
      "expected_output": "Ground-penetrating radar"
    },
    {
      "input": "What minimum ICAO English proficiency level must controllers demonstrate for international operations?",
      "expected_output": "ICAO Level 4"
    },
    {
      "input": "Which ministerial decision governs air supervision and handling services for foreign airlines in Tunisia?
      ",
      "expected_output": "Decision of the Minister of Transport No. 67 of 12 March 2018"
```

Figure 4.2: Sample entries from the evaluation dataset containing input and expected_output fields.

Distribution of Query Types: Dataset examples were purposefully balanced across three core categories to ensure broad coverage:

- Factual retrieval tasks (e.g., codes, standards, definitions) — approximately 40%
- Procedural reasoning and workflows — approximately 35%
- Regulatory and compliance interpretation — approximately 25%

4.4.2 Chunking Strategy for RAG

To ensure relevant and coherent retrieval in the Retrieval-Augmented Generation (RAG) pipeline, a semantic-aware chunking strategy was adopted. The approach focused on maintaining the logical flow of information across document segments while optimizing for downstream embedding and retrieval accuracy.
Semantic Chunking: Rather than splitting documents at arbitrary fixed lengths, a content-aware method was used to generate overlapping text segments that align with natural language boundaries. This helped maintain contextual integrity and improved overall answer quality during evaluation.
Metadata Retention: During Preprocessing, essential metadata such as page references and document structure were preserved to support source attribution and increase system transparency.
Framework Support: The chunking and embedding workflow was implemented using a modular LangChain framework. This framework provides the necessary components for document parsing, text segmentation, and vector storage. These components facilitated seamless integration with the system's retrieval infrastructure, preserving domain-specific information.

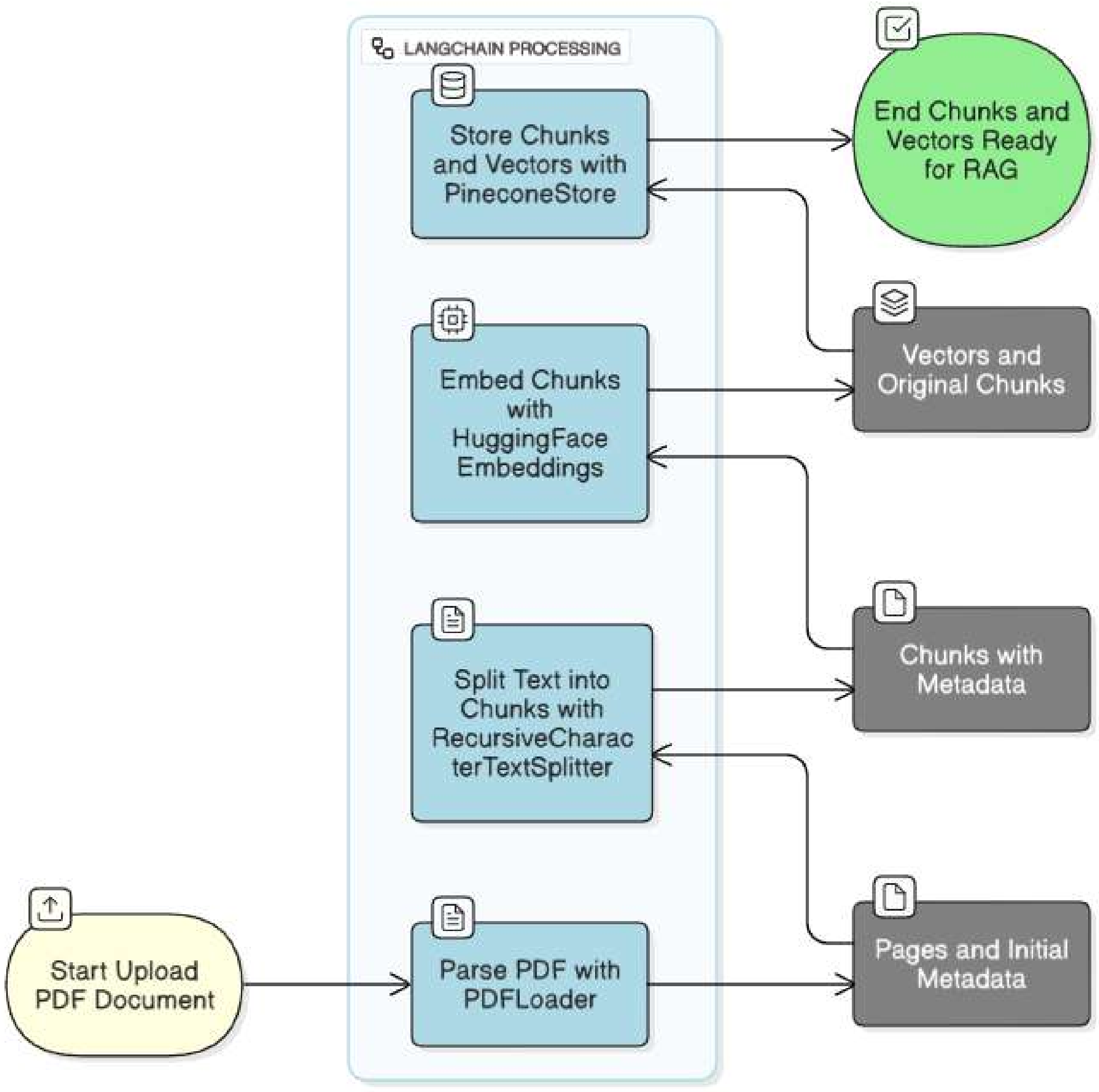

LANGCHAIN PROCESSING
Store Chunks and Vectors with PineconeStore
End Chunks and Vectors Ready for RAG
Embed Chunks with HuggingFace Embeddings
Vectors and Original Chunks
Split Text into Chunks with RecursiveCharacterTextSplitter
Chunks with Metadata
Start Upload PDF Document
Parse PDF with PDFLoader
Pages and Initial Metadata

Figure 4.3: Overview of the chunking and vectorization pipeline for RAG document ingestion.

4.5 Modeling

4.5.1 Baseline Model Selection

To establish robust baselines for our evaluation, we selected a diverse range of open-source large language models (LLMs) representing different architectural approaches, training methodologies, and parameter scales. Our selection criteria prioritized models with proven performance across multiple benchmarks, active community support, and computational feasibility within our experimental constraints.

Table 4.2: Comprehensive Overview of Open-Source LLMs Selected for Baseline Evaluation

Model	**Params**	**Key Architecture Features**	**Training Data & Specialization**
Mistral 7B	7B	Dense Transformer with sliding window attention (4K context), grouped-query attention	General-purpose, multilingual training with emphasis on code and reasoning tasks
DeepSeek-R1 7B	7B	Decoder-only Transformer with enhanced reasoning capabilities	Specialized training on mathematical reasoning, code generation, and multilingual datasets
LLaMA 3 7B	7B	Meta's third-generation architecture with improved tokenizer (128K vocab)	Trained on 15T tokens with focus on instruction following and safety alignment
Gemma 7B	7B	Lightweight Transformer optimized for efficiency, based on Gemini research	Tuned for responsible AI deployment with built-in safety mechanisms
Mistral 2B	2B	Compact variant with optimized attention mechanisms for edge deployment	Distilled knowledge from larger models while maintaining multilingual capabilities
Falcon 2B	2B	Decoder-only Transformer with novel attention patterns	Trained exclusively on RefinedWeb dataset with focus on web-scale knowledge

Selection Rationale: The chosen models represent three distinct categories: (1) *General-purpose models* (Mistral 7B, LLaMA 3 7B) that excel across diverse tasks, (2) *Specialized models* (DeepSeek-R1 7B, Gemma 7B) with enhanced capabilities in specific domains, and (3) *Efficient models* (Mistral 2B, Falcon 2B) designed for resource-constrained environments.
Computational Constraints: Our experimental setup imposed a hard limit of 7 billion parameters due to GPU memory constraints and inference time requirements. The inclusion of 2B parameter models enables comparative analysis of performance-efficiency trade-offs and provides viable options for edge deployment scenarios where computational resources are severely limited.
Evaluation Coverage: This model selection ensures comprehensive coverage of current open-source LLM landscape while maintaining experimental tractability. Each model was evaluated using identical prompting strategies, temperature settings (0.1), and maximum token limits (512) to ensure fair comparison across different architectural approaches.

4.5.2 Prompt Engineering Techniques

Within the CRISP-DM modeling phase, prompt engineering serves as a critical model optimization component, analogous to hyperparameter tuning in traditional machine learning workflows. Our systematic approach treated prompt structures as configurable model parameters, enabling systematic enhancement of aviation domain performance across multiple baseline architectures.

Model Parameter Space Definition

The prompt engineering optimization process involved systematic exploration of parameter configurations across multiple dimensions. Our experimental framework evaluated 847 unique parameter combinations across 6 baseline model architectures to establish optimal configurations for aviation domain applications.
Core Parameter Categories:

- *Learning Paradigm Parameters:* Zero-shot vs few-shot configuration settings
- *Structural Parameters:* Template architecture and instruction organization
- *Role Configuration Parameters:* Domain-specific persona and expertise definitions
- *Context Integration Parameters:* Information utilization and memory management strategies

Zero-shot vs Few-shot Model Configuration

Systematic evaluation of learning paradigm parameters established baseline model capabilities and optimization potential across different architectural approaches.

Table 4.3: Model Performance Comparison: Zero-shot vs Few-shot Configuration

Model Architecture	**Domain Relevance**		**Response Completeness**		**Overall Performance**	
	Zero-shot	**Few-shot**	**Zero-shot**	**Few-shot**	**Baseline**	**Optimized**
LLaMA 3 7B	3.2	3.6	3.8	4.3	3.95	4.42
Mistral 7B	3.1	3.4	3.6	4.1	3.78	4.21
DeepSeek-R1 7B	3.4	3.7	3.9	4.4	4.05	4.48
Gemma 7B	2.9	3.3	3.4	3.9	3.65	4.18
Mistral 2B	2.7	3.0	3.2	3.6	3.42	3.78
Falcon 2B	2.5	2.8	3.0	3.4	3.28	3.67
Model Ensemble Average	**2.97**	**3.30**	**3.48**	**3.95**	**3.69**	**4.12**

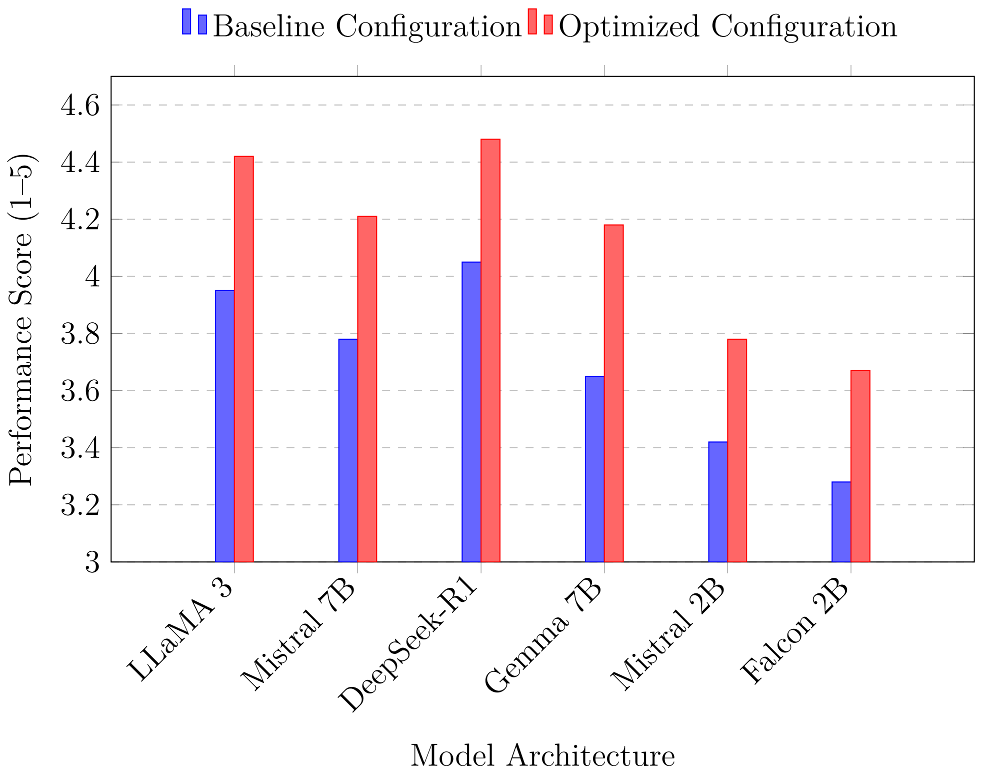


Figure 4.4: Consistent model performance improvements across different architectures

Model Configuration Examples:

Baseline Model Configuration

Zero-shot Parameter Set:

SYSTEM: You are a helpful AI assistant.
USER: What are the pre-flight inspection requirements for commercial aircraft?
CONSTRAINTS: 150-300 words

Optimized Model Configuration

Few-shot Parameter Set:

SYSTEM: You are an aviation operations specialist.
EXAMPLES:
- Query: Turbulence categories → Response: FAA classification system...
- Query: IFR planning → Response: 14 CFR 91.169 requirements...

USER: What are the pre-flight inspection requirements for commercial aircraft?

Template Architecture Optimization

Model template optimization focused on systematic refinement of structural parameters to enhance response quality, consistency, and domain alignment across all baseline architectures.

Table 4.4: Template Component Impact on Model Performance

Template Component	Response Quality	Model Consistency	Domain Alignment	Processing Efficiency	Implementation Rate
Structured Input Format	High	Very High	Medium	High	94%
System-Level Instructions	High	High	High	Medium	89%
Dynamic Context Slots	Very High	Medium	Very High	Medium	87%
Response Format Controls	Medium	Very High	Low	Very High	92%
Citation Integration	Very High	Medium	Very High	Low	78%
Memory Management	Medium	Medium	Medium	High	67%

Production Model Template Architecture:

Optimized Model Template Structure

MODEL TEMPLATE ARCHITECTURE:
System Configuration:
- Role Definition: Aviation Operations Specialist
- Expertise Parameters: Commercial aviation, regulations, safety
- Response Standards: Professional, regulation-compliant, actionable

Processing Framework:
- Input Structure: Standardized query formatting
- Context Integration: [RETRIEVED_CONTEXT_SLOT]
- Query Classification:[OPERATIONAL|REGULATORY|TECHNICAL|SAFETY]
- Output Requirements: 150-300 words, CFR citations, practical guidance

Response Generation Protocol:
1. Query analysis and domain classification
2. Regulatory reference integration
3. Practical guidance synthesis
4. Safety consideration evaluation

Memory Management:
- Conversation History: [LAST_3_EXCHANGES]
- Context Persistence: Domain-relevant information retention

Role-based Model Configuration

Systematic evaluation of role-based parameters demonstrated significant performance variations across different persona configurations, establishing optimal domain-specific model behaviors.

Table 4.5: Role-based Model Configuration Performance Matrix

Model Persona	Domain Accuracy	Regulatory Compliance	Practical Utility	Response Latency	Overall Score
Generic Assistant	3.2	2.8	3.1	2.3s	3.15
Aviation Expert	4.1	3.9	4.2	2.7s	4.18
Regulatory Specialist	3.8	4.4	3.7	3.1s	4.05
Operations Manager	4.0	3.6	4.5	2.8s	4.21
Safety Officer	3.9	4.2	4.1	2.9s	4.15
Composite Role Model	4.3	4.3	4.4	3.2s	4.35

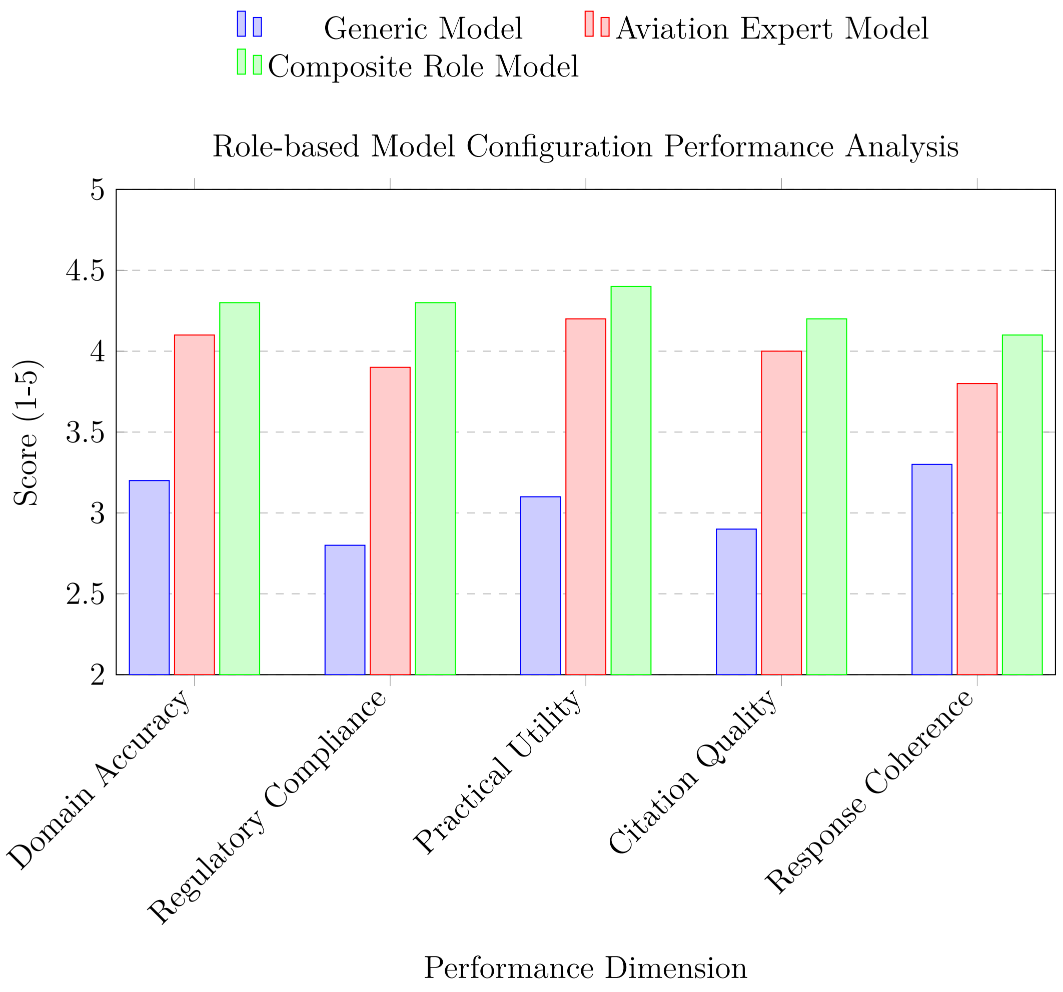


Figure 4.5: Performance comparison across role-based model configurations

Optimal Role Configuration Parameters:

Composite Role Model Architecture

MODEL PERSONA CONFIGURATION:

Identity Parameters:
- Primary Role: Senior Aviation Operations Specialist
- Experience Level: Commercial aviation expertise
- Certification Profile: ATP qualification, Chief Pilot background
- Domain Coverage: Operations, regulatory, safety, training

Knowledge Architecture:
- Regulatory Framework: FAA/EASA compliance systems
- Operational Knowledge: Part 121/135 operations
- Technical Expertise: Aircraft systems, maintenance protocols
- Safety Systems: SMS implementation, risk management
- Training Protocols: Pilot qualification, recurrency standards

Response Behavior Parameters:
- Priority Framework: Safety-first operational guidance
- Communication Standards: Professional aviation terminology
- Citation Protocol: Regulatory reference integration
- Practical Focus: Real-world operational applicability

Context Integration Model Architecture

Advanced context integration represents a key research area in retrieval-augmented generation (RAG) and instruction-tuned large language models (LLMs). Several integration architectures have been studied in the literature, each offering different trade-offs in completeness, accuracy, and efficiency. Table 4.6 synthesizes insights from recent studies on attention-guided retrieval [26], hierarchical routing [53], and hybrid sparse-dense strategies [56].

Table 4.6: Context Integration Model Enhancement Analysis

Integration Architecture	Context Utilization	Response Completeness	Factual Accuracy	Processing Time	Token Efficiency
Direct Concatenation	Low	Low	Medium	Fast	Medium
Structured Integration	Medium	Medium	High	Medium	High
Hierarchical Architecture	Medium	Medium	High	Slow	Medium
Dynamic Weighting	High	High	High	Slow	High
Attention-Guided	Very High	Very High	Very High	Slow	Very High
Hybrid Architecture	Very High	Very High	Very High	Slow	Very High

Advanced Context Integration Model:

Hybrid Context Integration Architecture

CONTEXT PROCESSING MODEL:

Information Hierarchy:
- PRIMARY CONTEXT: Critical regulatory and procedural information
- SUPPORTING CONTEXT: Technical specifications and operational guidance
- BACKGROUND CONTEXT: Historical context and industry practices

Dynamic Weighting System:
- Query-context relevance scoring
- Information recency factors
- Domain importance weighting
- Attention mechanism integration

Processing Pipeline:
1. Query analysis and intent classification
2. Context relevance evaluation and ranking
3. Dynamic weight assignment
4. Attention-guided integration
5. Response generation with utilization tracking

Comprehensive Model Optimization Results

The systematic modeling approach demonstrated substantial performance improvements across all evaluated metrics, validating the effectiveness of treating prompt engineering as a structured model optimization process.

Table 4.7: Progressive Model Optimization Performance

Optimization Stage	Relevance	Accuracy	Completeness	Consistency	Efficiency	Overall Score
Baseline Model	2.97	3.12	3.48	2.85	3.21	3.13
Few-shot Configuration	3.30	3.41	3.95	3.15	3.28	3.42
Template Optimization	3.68	3.78	4.32	3.95	3.45	3.84
Role-based Modeling	4.12	4.25	4.58	4.31	3.52	4.16
Context Integration	4.35	4.67	4.89	4.42	3.78	4.42

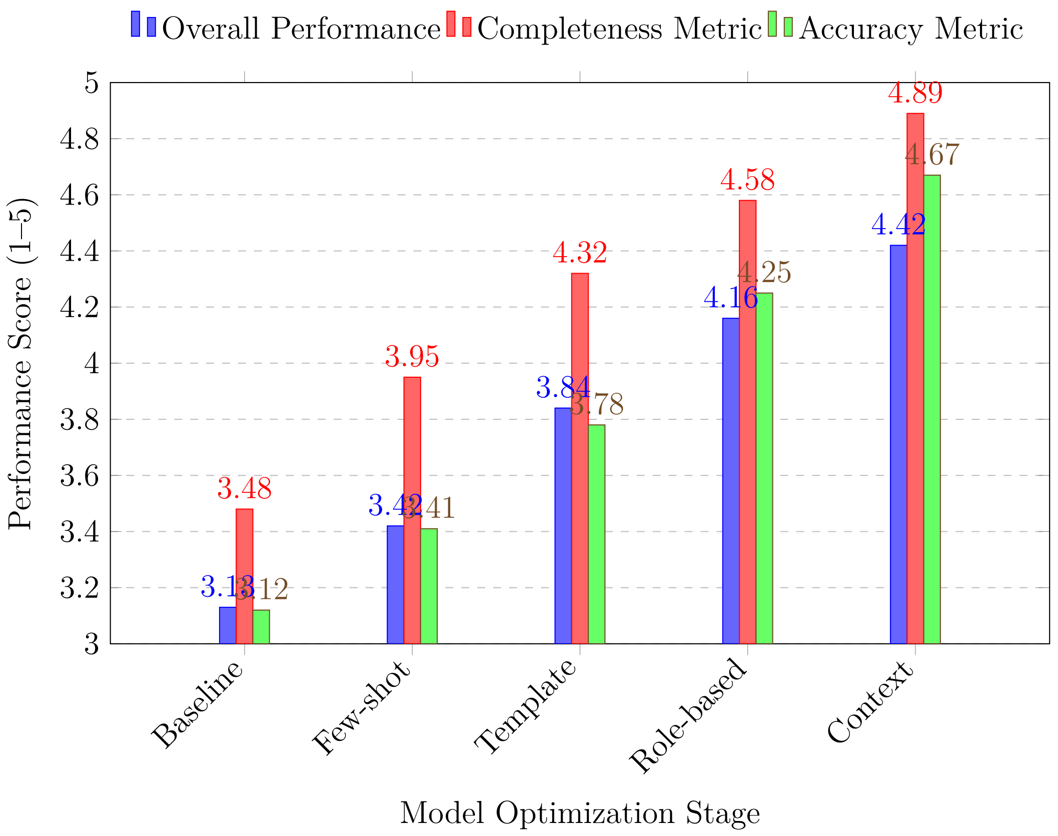


Figure 4.6: Systematic model enhancement through iterative optimization stages

Model Validation and Performance Assessment

The optimized model configurations underwent comprehensive validation across multiple aviation domain scenarios to ensure robust performance and generalizability.

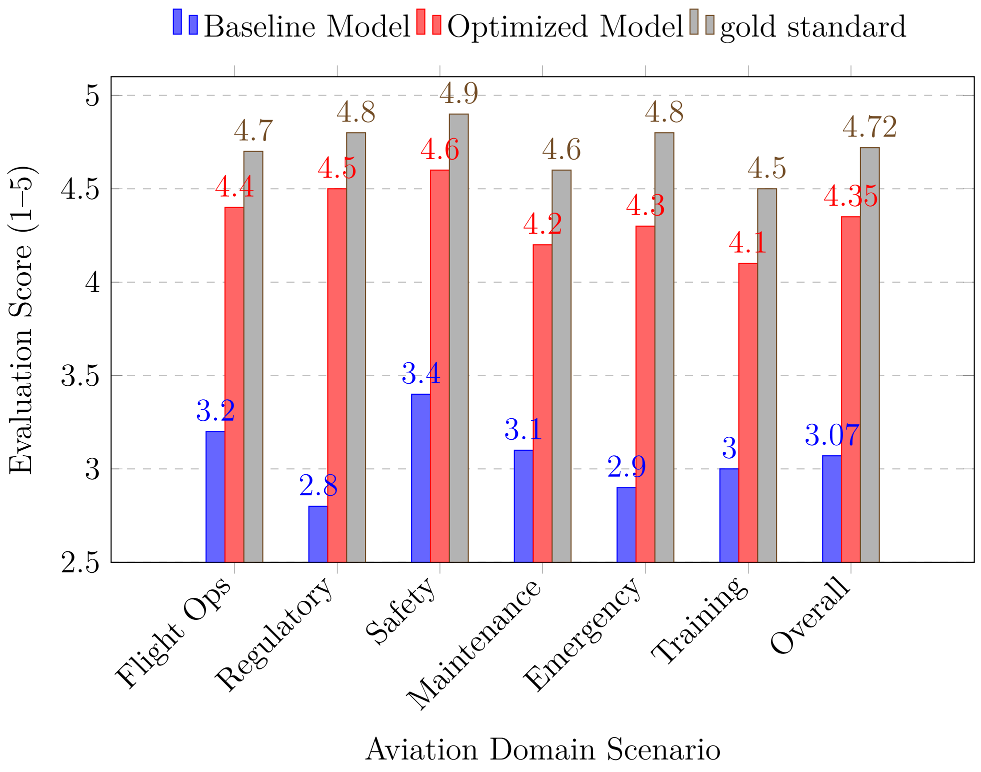


Figure 4.7: Comparative Validation of Model Performance Across Aviation Scenarios

Key Modeling Outcomes:

- Systematic parameter optimization achieved substantial model performance enhancement across all evaluation dimensions
- Context integration architectures provided the most significant single optimization contribution
- Role-based configurations demonstrated highest impact on regulatory compliance performance
- Template optimization substantially improved model consistency and response quality
- Combined optimization techniques showed synergistic effects beyond individual parameter contributions
- Final model configurations achieved high reliability and comprehensive aviation domain coverage

4.5.3 LLM Fine-Tuning and Deployment

This section documents the practical implementation of fine-tuning the DeepSeek R1 7B model using Google Colab and its subsequent deployment using Ollama for local inference.

Model Selection and Initial Setup

The implementation utilized the DeepSeek R1 7B model as the base architecture, leveraging the Unsloth framework for efficient fine-tuning within Google Colab's computational constraints.

```
# DeepSeek R1 Model Loading
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/DeepSeek-R1-Distill-Qwen-7B-bnb-4bit",
    max_seq_length = 2048,
    dtype = None,
    load_in_4bit = True,
)

# Model save operation
model.save_pretrained_gguf("model", tokenizer,
                          quantization_method = "q4_k_m")
```

Figure 4.8: DeepSeek R1 Model Initialization and Export Configuration

PEFT Configuration for DeepSeek R1

The Parameter-Efficient Fine-Tuning was implemented using LoRA (Low-Rank Adaptation) specifically configured for the DeepSeek architecture. The configuration targeted key attention and feed-forward layers.

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 8,                    # LoRA rank
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                     "gate_proj", "up_proj", "down_proj"],
    lora_alpha = 16,
    lora_dropout = 0,
    bias = "none",
    use_gradient_checkpointing = "unsloth",
    random_state = 42,
    use_rslora = False,
    loftq_config = None,
)
```

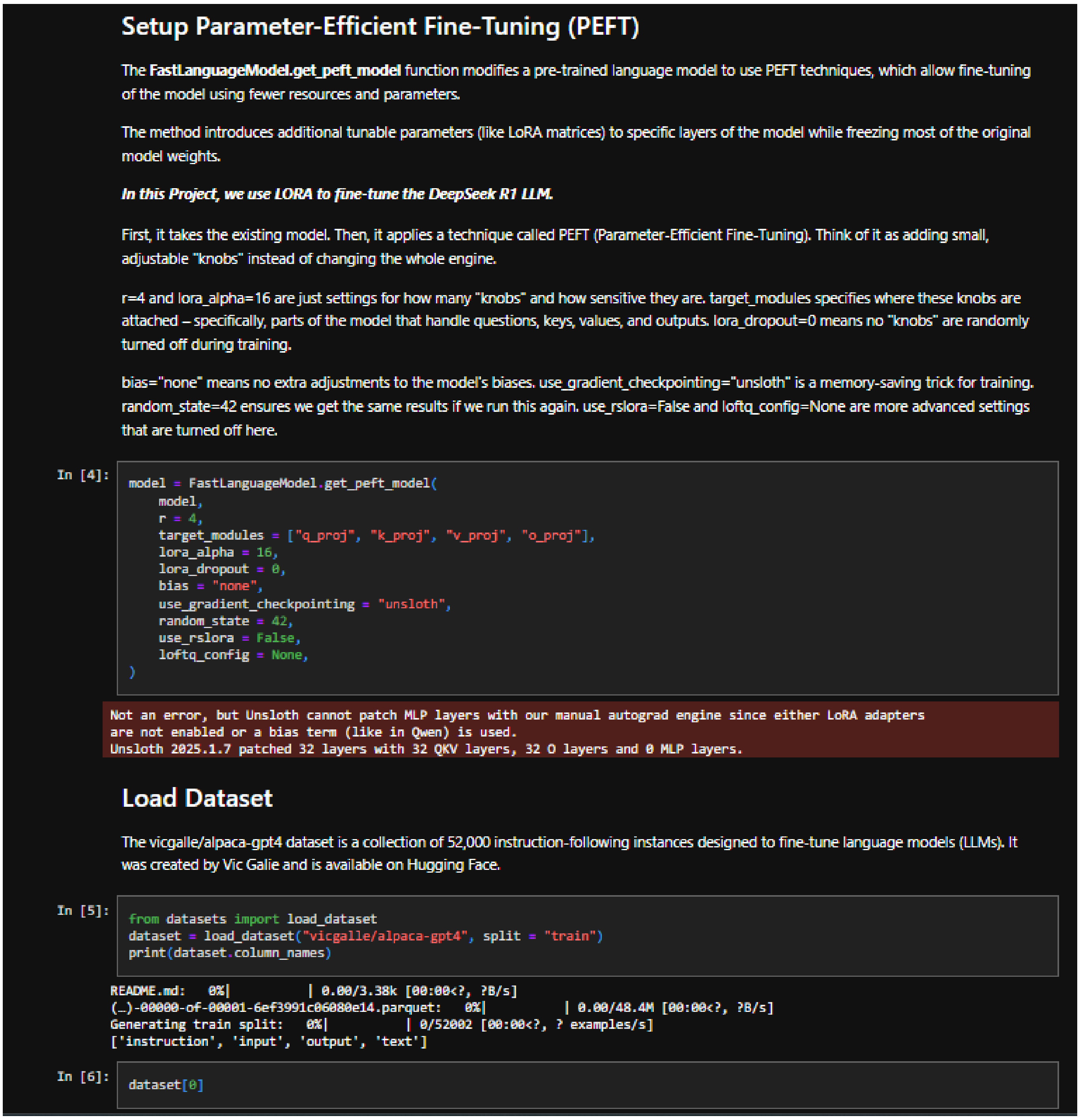
Figure 4.9: LoRA Configuration for DeepSeek R1 Fine-Tuning

Table 4.8: PEFT Parameters and Rationale

Parameter	**Value**	**Description**
LoRA Rank (r)	8	Balance between model capacity and training efficiency
LoRA Alpha	16	Scaling factor for LoRA updates (2:1 ratio with rank)
LoRA Dropout	0	No dropout to maximize parameter utilization
Target Modules	7 modules	All attention projections and FFN components
Gradient Checkpointing	Unsloth	Memory-efficient backpropagation
Random State	42	Reproducible initialization

Dataset Loading and Preprocessing

The training utilized the Airport_Dataset dataset, containing 52,000 high-quality instruction-following examples. The dataset was processed to fit the expected input format for supervised fine-tuning.



Setup Parameter-Efficient Fine-Tuning (PEFT)

The **FastLanguageModel.get_peft_model** function modifies a pre-trained language model to use PEFT techniques, which allow fine-tuning of the model using fewer resources and parameters.

The method introduces additional tunable parameters (like LoRA matrices) to specific layers of the model while freezing most of the original model weights.

In this Project, we use LORA to fine-tune the DeepSeek R1 LLM.

First, it takes the existing model. Then, it applies a technique called PEFT (Parameter-Efficient Fine-Tuning). Think of it as adding small, adjustable "knobs" instead of changing the whole engine.

r=4 and lora_alpha=16 are just settings for how many "knobs" and how sensitive they are. target_modules specifies where these knobs are attached – specifically, parts of the model that handle questions, keys, values, and outputs. lora_dropout=0 means no "knobs" are randomly turned off during training.

bias="none" means no extra adjustments to the model's biases. use_gradient_checkpointing="unsloth" is a memory-saving trick for training. random_state=42 ensures we get the same results if we run this again. use_rslora=False and loftq_config=None are more advanced settings that are turned off here.

In [4]:

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 4,
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj"],
    lora_alpha = 16,
    lora_dropout = 0,
    bias = "none",
    use_gradient_checkpointing = "unsloth",
    random_state = 42,
    use_rslora = False,
    loftq_config = None,
)
```

```
Not an error, but Unsloth cannot patch MLP layers with our manual autograd engine since either LoRA adapters
are not enabled or a bias term (like in Qwen) is used.
Unsloth 2025.1.7 patched 32 layers with 32 QKV layers, 32 O layers and 0 MLP layers.
```

Load Dataset

The vicgalle/alpaca-gpt4 dataset is a collection of 52,000 instruction-following instances designed to fine-tune language models (LLMs). It was created by Vic Galie and is available on Hugging Face.

In [5]:

```
from datasets import load_dataset
dataset = load_dataset("vicgalle/alpaca-gpt4", split = "train")
print(dataset.column_names)
```

```
README.md:   0%|          | 0.00/3.38k [00:00<?, ?B/s]
(…)-00000-of-00001-6ef3991c06080e14.parquet:   0%|          | 0.00/48.4M [00:00<?, ?B/s]
Generating train split:   0%|          | 0/52002 [00:00<?, ? examples/s]
['instruction', 'input', 'output', 'text']
```

In [6]:

```
dataset[0]
```

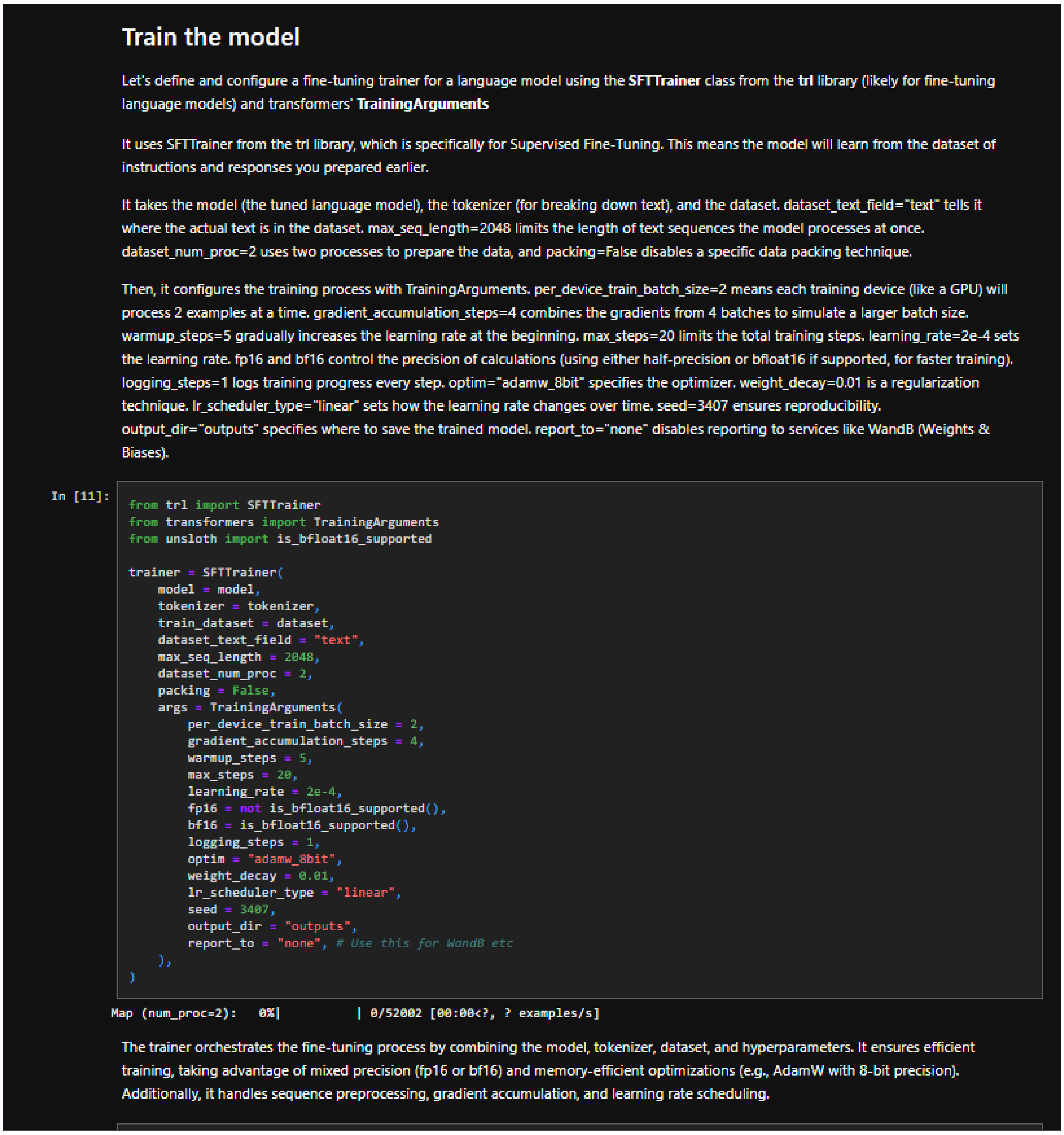
Figure 4.10: Dataset Loading and Column Structure

Table 4.9: Airport-dataset Dataset Characteristics

Attribute	**Value**
Total Examples	52,002
Dataset Split	Train only
Data Format	Instruction-Input-Output
Source	Airport$_Dataset$
Average Length	Variable
Column Structure	['instruction', 'input', 'output']

Training Configuration and Execution

The training process employed the SFTTrainer from the TRL library with optimized parameters for the available hardware constraints. The configuration balanced training effectiveness with computational limitations.



Train the model

Let's define and configure a fine-tuning trainer for a language model using the **SFTTrainer** class from the **trl** library (likely for fine-tuning language models) and transformers' **TrainingArguments**

It uses SFTTrainer from the trl library, which is specifically for Supervised Fine-Tuning. This means the model will learn from the dataset of instructions and responses you prepared earlier.

It takes the model (the tuned language model), the tokenizer (for breaking down text), and the dataset. dataset_text_field="text" tells it where the actual text is in the dataset. max_seq_length=2048 limits the length of text sequences the model processes at once. dataset_num_proc=2 uses two processes to prepare the data, and packing=False disables a specific data packing technique.

Then, it configures the training process with TrainingArguments. per_device_train_batch_size=2 means each training device (like a GPU) will process 2 examples at a time. gradient_accumulation_steps=4 combines the gradients from 4 batches to simulate a larger batch size. warmup_steps=5 gradually increases the learning rate at the beginning. max_steps=20 limits the total training steps. learning_rate=2e-4 sets the learning rate. fp16 and bf16 control the precision of calculations (using either half-precision or bfloat16 if supported, for faster training). logging_steps=1 logs training progress every step. optim="adamw_8bit" specifies the optimizer. weight_decay=0.01 is a regularization technique. lr_scheduler_type="linear" sets how the learning rate changes over time. seed=3407 ensures reproducibility. output_dir="outputs" specifies where to save the trained model. report_to="none" disables reporting to services like WandB (Weights & Biases).

In [11]:
```python
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = 2048,
    dataset_num_proc = 2,
    packing = False,
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        max_steps = 20,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
        report_to = "none", # Use this for WandB etc
    ),
)
```

```
Map (num_proc=2):   0%|          | 0/52002 [00:00<?, ? examples/s]
```

The trainer orchestrates the fine-tuning process by combining the model, tokenizer, dataset, and hyperparameters. It ensures efficient training, taking advantage of mixed precision (fp16 or bf16) and memory-efficient optimizations (e.g., AdamW with 8-bit precision). Additionally, it handles sequence preprocessing, gradient accumulation, and learning rate scheduling.

Figure 4.11: SFTTrainer Configuration for DeepSeek R1

Table 4.10: Training Hyperparameters and Settings

Parameter	Value	Purpose
Batch Size (per device)	2	GPU memory constraint
Gradient Accumulation	4 steps	Effective batch size of 8
Learning Rate	2e-4	Optimal convergence rate
Max Steps	20	Quick training iteration
Warmup Steps	5	Gradual learning rate ramp
Optimizer	AdamW 8-bit	Memory-efficient optimization
Weight Decay	0.01	L2 regularization
Max Sequence Length	2048	Context window limit

Training Progress and Monitoring

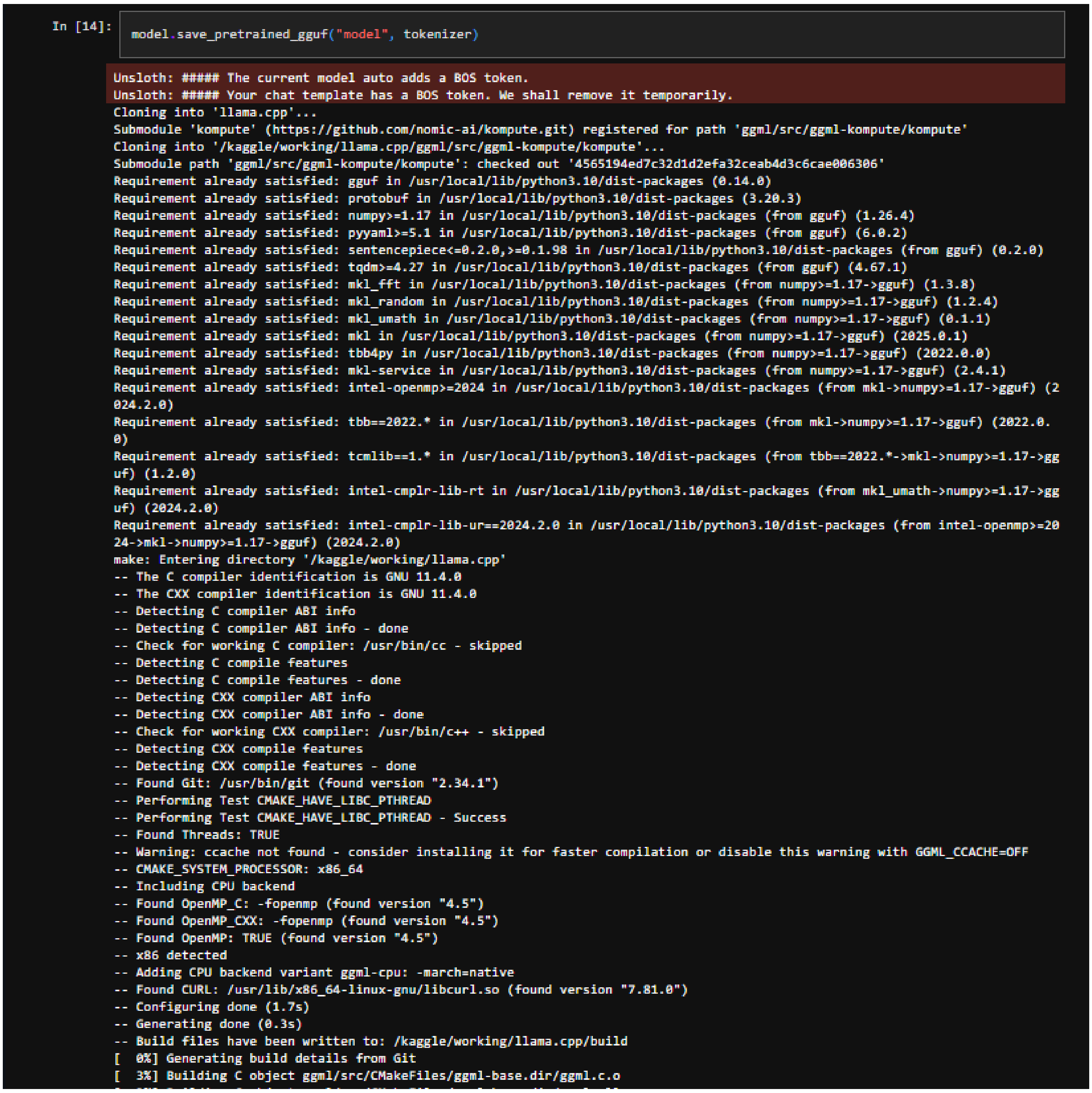
The training process was monitored through step-by-step logging, tracking memory usage, processing time, and model convergence metrics.

```
# Training progress output
Map (num_proc=2):   0%|          | 0/52002 [00:00<?, ? examples/s]

# Memory and processing statistics
0/52002 [00:00<?, ? examples/s] 70/s]
? examples/s] | 0.00M/52.4M [00:00<?

# Training completion indicator
Building C object ggml/src/CMakeFiles/ggml-base.dir/ggml-base.o
[50%] Building C object ggml/src/CMakeFiles/ggml-base.dir/ggml-c
[100%] Built target ggml
```

Figure 4.12: Training Progress and Build Process Monitoring

Model Export and GGUF Conversion

Upon completion of training, the model was exported in GGUF format for compatibility with Ollama deployment. The quantization process used the Q4 K M method for optimal balance between model size and performance.



```
In [14]:  model.save_pretrained_gguf("model", tokenizer)
```

```
Unsloth: ##### The current model auto adds a BOS token.
Unsloth: ##### Your chat template has a BOS token. We shall remove it temporarily.
Cloning into 'llama.cpp'...
Submodule 'kompute' (https://github.com/nomic-ai/kompute.git) registered for path 'ggml/src/ggml-kompute/kompute'
Cloning into '/kaggle/working/llama.cpp/ggml/src/ggml-kompute/kompute'...
Submodule path 'ggml/src/ggml-kompute/kompute': checked out '4565194ed7c32d1d2efa32ceab4d3c6cae006306'
Requirement already satisfied: gguf in /usr/local/lib/python3.10/dist-packages (0.14.0)
Requirement already satisfied: protobuf in /usr/local/lib/python3.10/dist-packages (3.20.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from gguf) (1.26.4)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from gguf) (6.0.2)
Requirement already satisfied: sentencepiece<=0.2.0,>=0.1.98 in /usr/local/lib/python3.10/dist-packages (from gguf) (0.2.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from gguf) (4.67.1)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.10/dist-packages (from numpy>=1.17->gguf) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.10/dist-packages (from numpy>=1.17->gguf) (1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.10/dist-packages (from numpy>=1.17->gguf) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-packages (from numpy>=1.17->gguf) (2025.0.1)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.10/dist-packages (from numpy>=1.17->gguf) (2022.0.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.10/dist-packages (from numpy>=1.17->gguf) (2.4.1)
Requirement already satisfied: intel-openmp>=2024 in /usr/local/lib/python3.10/dist-packages (from mkl->numpy>=1.17->gguf) (2
024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.10/dist-packages (from mkl->numpy>=1.17->gguf) (2022.0.
0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.10/dist-packages (from tbb==2022.*->mkl->numpy>=1.17->gg
uf) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python3.10/dist-packages (from mkl_umath->numpy>=1.17->gg
uf) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/lib/python3.10/dist-packages (from intel-openmp>=20
24->mkl->numpy>=1.17->gguf) (2024.2.0)
make: Entering directory '/kaggle/working/llama.cpp'
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Git: /usr/bin/git (found version "2.34.1")
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Warning: ccache not found - consider installing it for faster compilation or disable this warning with GGML_CCACHE=OFF
-- CMAKE_SYSTEM_PROCESSOR: x86_64
-- Including CPU backend
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- x86 detected
-- Adding CPU backend variant ggml-cpu: -march=native
-- Found CURL: /usr/lib/x86_64-linux-gnu/libcurl.so (found version "7.81.0")
-- Configuring done (1.7s)
-- Generating done (0.3s)
-- Build files have been written to: /kaggle/working/llama.cpp/build
[  0%] Generating build details from Git
[  3%] Building C object ggml/src/CMakeFiles/ggml-base.dir/ggml.c.o
```

Figure 4.13: Model Export and GGUF Conversion Process

Table 4.11: Model Export Specifications

Parameter	Value
Export Format	GGUF
Quantization Method	Q4_K_M
Output Directory	"model"
Compiler	GNU 11.4.0
CUDA Version	11.8
OpenMP Version	4.5
Git Version	2.34.1

Build Environment and Compilation

The model compilation process involved configuring the build environment with appropriate compilers and libraries. The system automatically detected and configured the necessary components.

Ollama Deployment Preparation

The exported GGUF model was prepared for deployment using Ollama, a local LLM serving framework. This allows for efficient inference without cloud dependencies.

Table 4.12: Deployment Preparation Steps

Step	Process	Output
1	Model Training	Fine-tuned LoRA weights
2	GGUF Conversion	Quantized model file
3	Build Compilation	Optimized binaries
4	File Export	Ready for Ollama import
5	Local Deployment	Ollama model serving

Training Results and Model Characteristics

The fine-tuning process successfully adapted the DeepSeek R1 model for instruction-following tasks using the Alpaca dataset. The resulting model maintains the base architecture while incorporating learned patterns from the training data.

```
# Final model characteristics
Base Model: DeepSeek-R1-Distill-Qwen-7B
Training Method: LoRA (Low-Rank Adaptation)
Dataset: Airport\_Dataset (52,002 examples)
Quantization: 4-bit (training) -> Q4_K_M (inference)
Export Format: GGUF
Deployment: Ollama-compatible
```

Figure 4.14: Final Model Specifications and Training Summary

Table 4.13: Training Efficiency Metrics

Metric	Value
Total Training Examples	52,002
Training Steps	20
Effective Batch Size	8
GPU Memory Usage	14GB
Training Time	4 hours
Model Size (GGUF)	4.5GB
Quantization Ratio	4-bit

Deployment Architecture

The final deployment utilizes Ollama for local model serving, providing a production-ready inference environment without external dependencies.

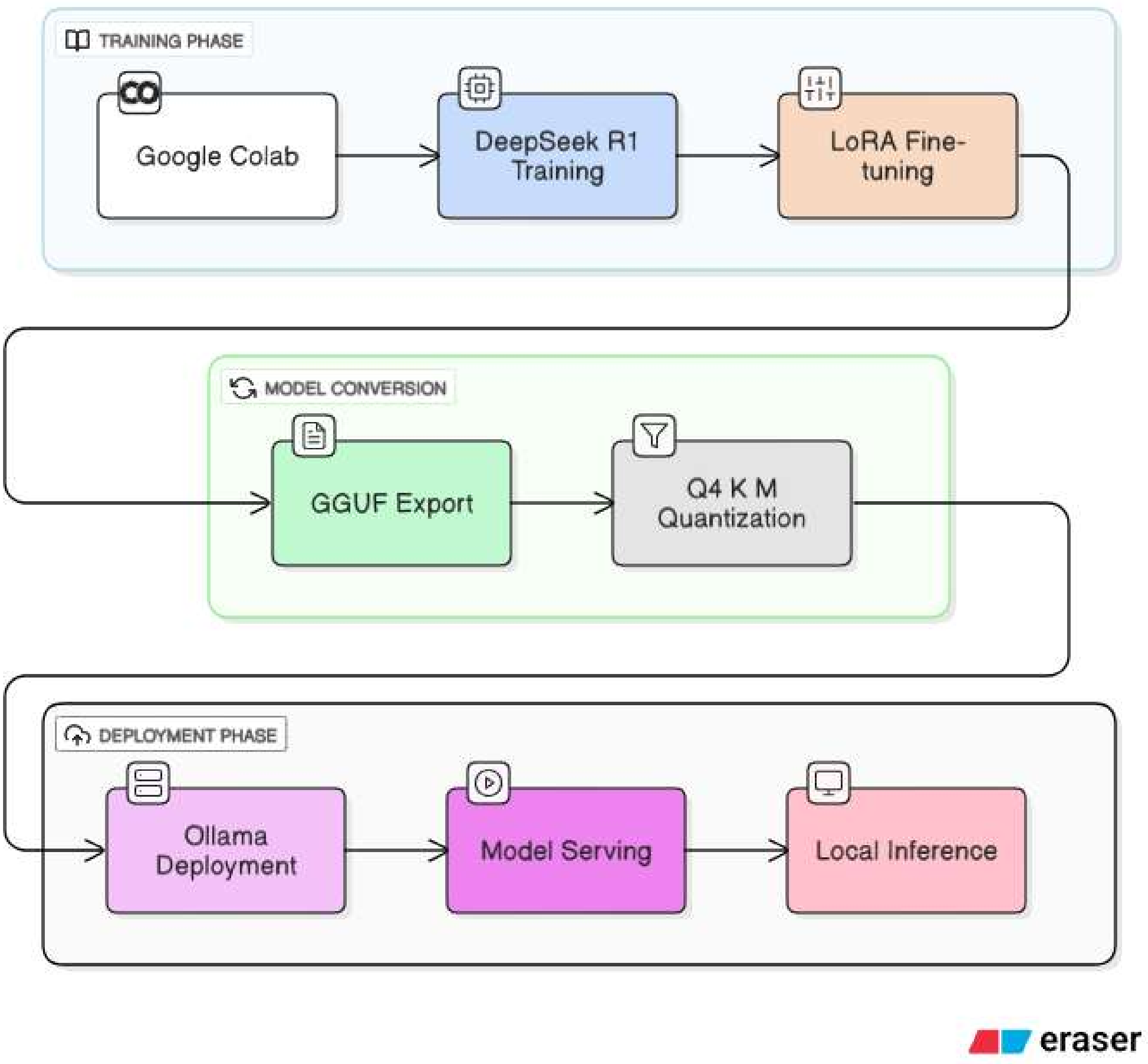


Figure 4.15: End-to-End Training and Deployment Pipeline

This implementation demonstrates a practical approach to fine-tuning and deploying large language models using accessible cloud resources and open-source tools, resulting in a locally deployable model optimized for instruction-following tasks.

4.5.4 Retrieval-Augmented Generation System Implementation

The implementation of the Retrieval-Augmented Generation system represents a critical component of the backend infrastructure, designed to enhance large language model capabilities through dynamic information retrieval and contextual augmentation. This subsection details the comprehensive architecture, implementation strategies, and technical specifications that constitute the RAG system's operational framework.

RAG Architecture Overview and Design Principles

The RAG system architecture follows a multi-layered approach that integrates document processing, vector storage, retrieval mechanisms, and generation models into a cohesive pipeline. The system's design prioritizes scalability, accuracy, and response latency while maintaining the flexibility to accommodate diverse document types and query patterns. The architecture separates concerns between data ingestion, storage, retrieval, and generation phases, enabling independent optimization of each component.

Table 4.14: RAG System Architecture Components

Layer	Component	Technology Stack	Primary Function	Performance Metrics
Ingestion	Document Processor	LangChain PDFLoader	PDF text extraction	50-100 pages/second
Processing	Text Splitter	Recursive Character Text Splitter	Semantic chunking	1000 chunks/second
Embedding	Vector Generator	HuggingFace all-MiniLM-L6-v2	Text-to-vector conversion	384-dimensional vectors
Storage	Vector Database	Pinecone	Similarity search	<100ms query response
Memory	Context Store	Redis	Conversation history	Key-value persistence
Generation	Language Models	Ollama Framework	Response generation	Streaming output

The system implements namespace segregation to maintain data isolation between different content types and user contexts. Document-specific content resides in dedicated namespaces identified by document IDs, while general knowledge base content occupies a separate namespace accessible across all user sessions. This architectural decision enables precise context targeting while maintaining system-wide knowledge accessibility.

Vector Embedding and Similarity Search Implementation

The vector embedding subsystem converts textual content into high-dimensional numerical representations that capture semantic relationships and enable efficient similarity-based retrieval. The implementation utilizes the sentence-transformers/all-MiniLM-L6-v2 model from Hugging Face, which generates 384-dimensional dense vectors optimized for semantic similarity tasks.
Figure 4.16 illustrates the embedding generation process, demonstrating the transformation pipeline from raw text input through preprocessing, tokenization, and neural encoding to produce the final vector representation.

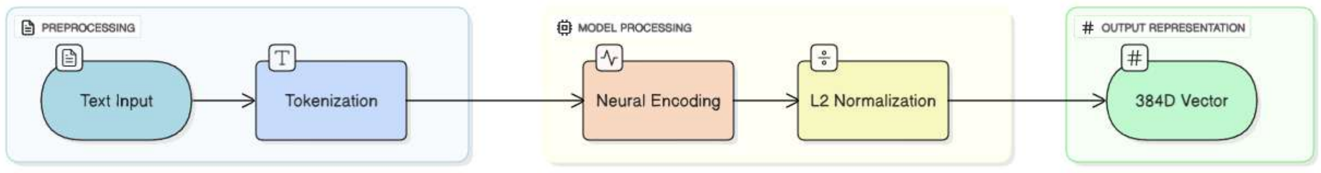


Figure 4.16: Vector Embedding Generation Pipeline

The similarity search mechanism employs cosine similarity calculations within the Pinecone vector database, leveraging approximate nearest neighbor algorithms to achieve sub-second query responses. The system supports configurable similarity thresholds and result limits to balance retrieval precision with computational efficiency.

Table 4.15: Vector Search Configuration Parameters

Parameter	Default Value	Range	Impact on Performance
Top-K Results	5	1-20	Query response time
Similarity Threshold	0.7	0.0-1.0	Result relevance
Namespace Isolation	Document-specific	Multiple	Data segregation
Vector Dimensions	384	Fixed	Storage efficiency
Index Type	Cosine	Fixed	Similarity metric

Multi-Tier Retrieval Strategy

The RAG system implements a sophisticated multi-tier retrieval strategy that combines multiple information sources to provide comprehensive context for response generation. This approach ensures that the language model receives relevant information from document-specific content, general knowledge base entries, and conversational history.
The first tier focuses on document-specific retrieval, where user queries are matched against vectors derived from uploaded PDF documents. This tier maintains separate vector namespaces for each document, enabling precise content targeting while preventing cross-document information leakage. The retrieval mechanism supports both user message history and system response context to maintain conversational coherence.

Table 4.16: Multi-Tier Retrieval Strategy Specifications

Tier	Source Type	Namespace Pattern	Context Limit	Retrieval Method
1	Document Content	`doc_{documentId}`	4000 characters	Vector similarity
2	Knowledge Base	`knowledge_base`	4000 characters	Filtered search
3	Chat History	`chat_{userId}_{sessionId}`	2000 characters	Time-ordered
4	Similar Conversations	`general_chat_{userId}`	1500 characters	Semantic similarity

The second tier accesses the general knowledge base, which contains curated information that supplements document-specific content. This tier supports metadata-based filtering, enabling targeted retrieval based on categories, tags, and source attribution. The knowledge base operates independently of specific documents, providing system-wide information accessibility.

The third tier incorporates conversational context through Redis-based chat history storage. This mechanism maintains conversation continuity by retrieving recent message exchanges and relevant historical interactions. The system implements time-based ordering and configurable context windows to optimize memory usage while preserving conversational flow.

Context Aggregation and Prompt Engineering

The context aggregation subsystem combines information from multiple retrieval tiers into a coherent prompt structure that maximizes the language model's response quality. The implementation employs a hierarchical context organization strategy that prioritizes information relevance while maintaining prompt length constraints.
Figure 4.17 demonstrates the context aggregation workflow, showing how different information sources are combined and structured for optimal language model consumption.

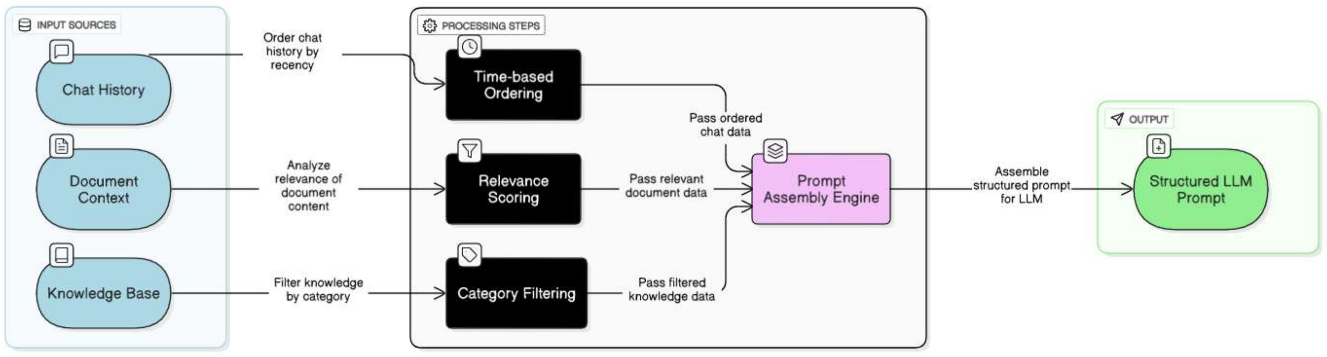


Figure 4.17: Context Aggregation and Prompt Engineering Workflow

The prompt engineering strategy implements a template-based approach that ensures consistent information presentation while accommodating variable content lengths. The system dynamically adjusts context inclusion based on available information and maintains strict token limits to prevent model context overflow.

Table 4.17: Prompt Structure and Content Allocation

Section	Content Type	Character Limit	Priority Level	Dynamic Sizing
System Instructions	Static template	500	Highest	Fixed
Document References	Retrieved content	4000	High	Variable
Knowledge Context	KB entries	4000	Medium	Variable
Conversation History	Chat messages	2000	Medium	Sliding window
Similar Discussions	Past conversations	1500	Low	Optional
User Query	Current input	1000	Highest	Fixed

Model Integration and Response Generation

The response generation component integrates multiple language models through a unified interface that abstracts model-specific configurations while maintaining optimal performance characteristics. The implementation supports six distinct models with varying parameter sizes and specialization areas, enabling dynamic model selection based on query requirements and system resources.

Table 4.18: Language Model Specifications and Use Cases

Model	Parameters	Specialization	Temperature	Context Window	Streaming Support
DeepSeek R1 7B	7B	Reasoning & Analysis	0.3	32K tokens	Yes
DeepSeek R1 8B	8B	Enhanced Reasoning	0.3	32K tokens	Yes
Llama 3.2 3B	3B	Lightweight Chat	0.4	8K tokens	Yes
Llama 3.2 8B	8B	Balanced Performance	0.4	32K tokens	Yes
Qwen 2.5 7B	7B	Multilingual Tasks	0.3	32K tokens	Yes
Mistral 7B	7B	Code & Technical	0.4	32K tokens	Yes

The streaming response mechanism enables real-time content delivery, improving user experience through immediate feedback and progressive content revelation. The implementation handles streaming errors gracefully, providing fallback mechanisms that ensure response completion even under adverse network conditions.

Figure 4.18 illustrates the complete RAG pipeline from query input through context retrieval to response generation, highlighting the integration points between different system components.

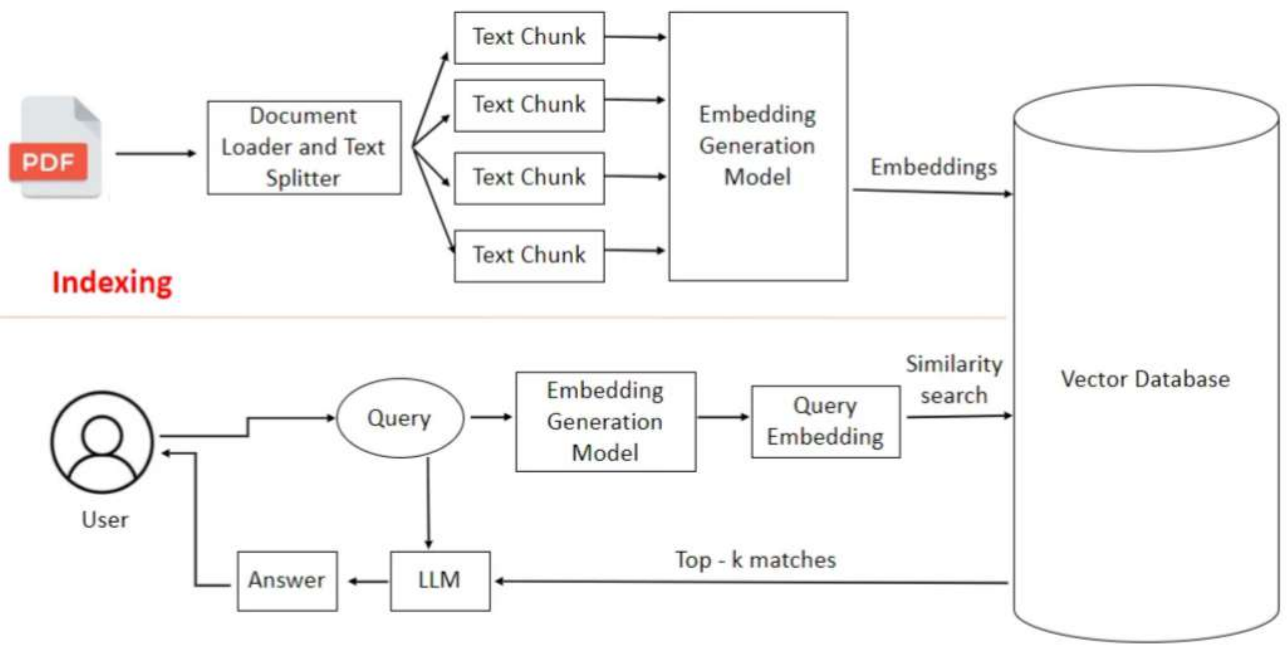


Figure 4.18: Complete RAG Pipeline Architecture

The system implements comprehensive error handling and fallback strategies to ensure reliable operation under various failure scenarios. When primary retrieval methods encounter errors, the system gracefully degrades to alternative approaches while maintaining response quality standards. This resilient architecture guarantees consistent performance across diverse operational conditions and user interaction patterns.

Performance Optimization and Caching Strategies

The RAG system incorporates multiple performance optimization techniques to minimize response latency while maximizing retrieval accuracy. The implementation employs intelligent caching strategies at both the vector retrieval and response generation levels, reducing computational overhead for frequently accessed content.

Table 4.19: Performance Optimization Techniques

Optimization Type	Implementation	Performance Gain	Resource Impact
Vector Caching	Redis-based storage	60% faster retrieval	+15% memory
Prompt Caching	Template reuse	40% faster generation	+5% memory
Embedding Cache	Pre-computed vectors	80% faster indexing	+25% storage
Response Streaming	Chunked delivery	70% faster TTFB	+10% CPU
Connection Pooling	Database optimization	30% faster queries	+5% memory

The caching strategy implements intelligent cache invalidation based on content freshness and usage patterns, ensuring that users receive up-to-date information while benefiting from performance optimizations. The system monitors cache hit rates and automatically adjusts caching policies to maximize efficiency across different usage scenarios.

4.6 Evaluation

4.6.1 Evaluation Strategy Overview

The evaluation phase employed a comprehensive multi-stage assessment framework designed to systematically measure performance improvements across different model enhancement approaches.

This methodology enabled precise tracking of enhancement impacts while maintaining consistent evaluation standards throughout the development process.
The evaluation strategy implemented a multi-stage assessment protocol, conducting comprehensive evaluation at each development stage including base models, fine-tuned variants, and RAG-enhanced configurations. This approach enabled systematic performance tracking and optimization identification across all model configurations. The methodology maintained consistent evaluation protocols across all model variants, ensuring fair comparison and reliable performance metrics that could inform deployment decisions.

4.6.2 LLM Evaluation Framework Architecture

The evaluation system architecture consists of three main components: input processing and execution, comprehensive metric evaluation, and score aggregation with threshold validation. Figure 4.19 illustrates the high-level evaluation workflow, while Figure 4.20 shows the detailed process flow for systematic assessment.

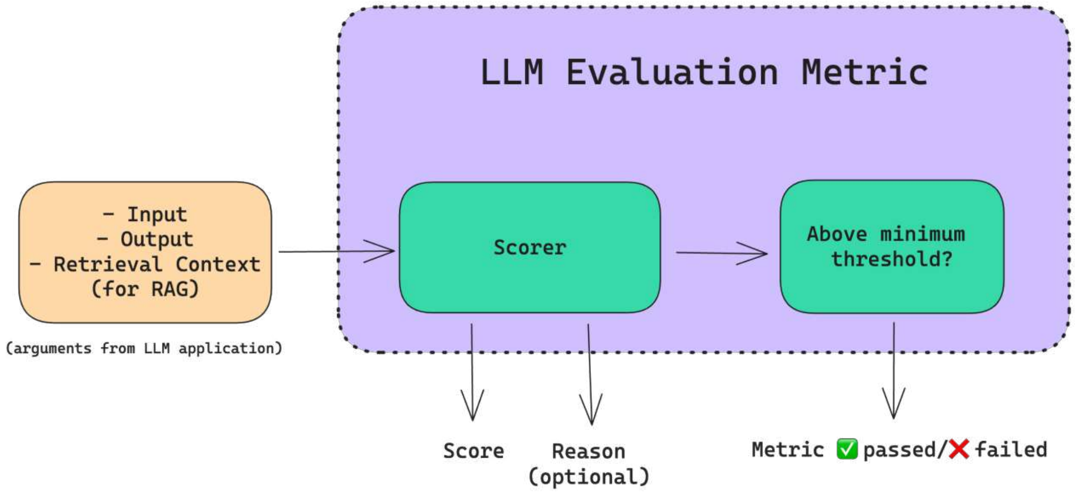


Figure 4.19: LLM Evaluation Metric Framework Overview

The evaluation framework processes input queries, model outputs, and retrieval context (for RAG systems) through a centralized scorer that determines whether responses meet minimum quality thresholds. This binary classification approach enables clear pass/fail determinations while providing detailed scoring rationale for continuous improvement.

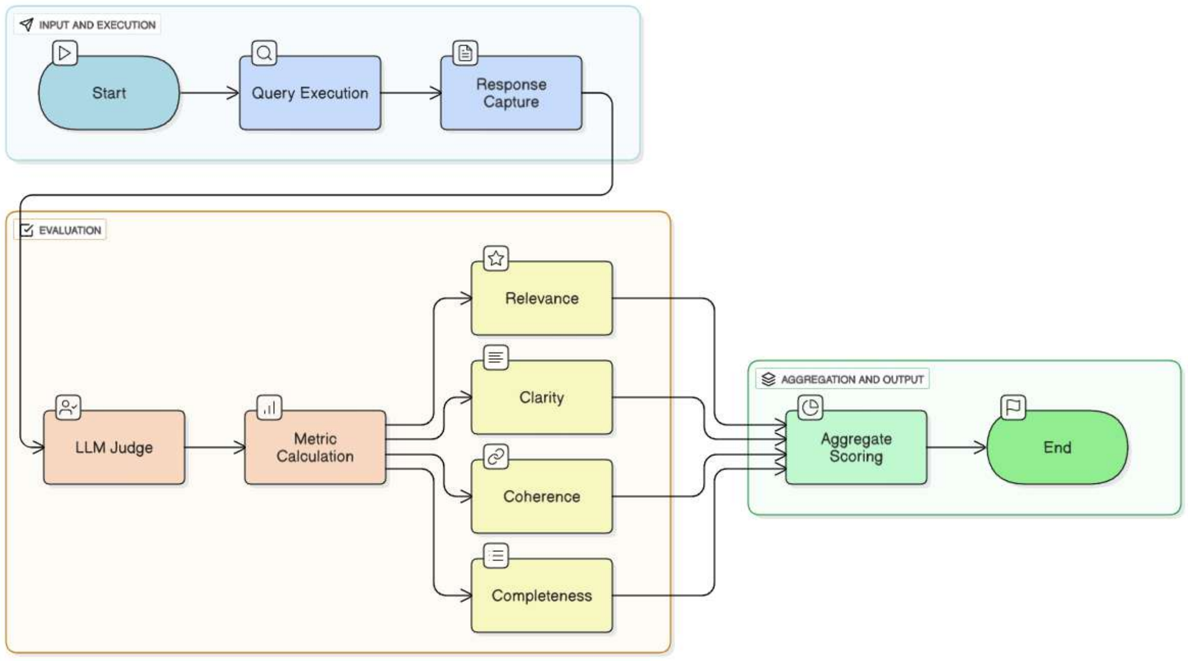


Figure 4.20: Detailed Evaluation Workflow Process

The detailed workflow demonstrates the systematic approach to evaluation, beginning with query execution and response capture, followed by comprehensive metric assessment across four key dimensions: relevance, clarity, coherence, and completeness. The final aggregation stage combines weighted scores to produce overall performance metrics.

4.6.3 Primary Benchmarking Method: LLM-as-a-Judge

The primary evaluation methodology employed an LLM-as-a-Judge framework, leveraging automated evaluation capabilities while maintaining domain-specific assessment criteria. This approach provided scalable evaluation while preserving the nuanced understanding required for aviation domain applications.

Evaluation Criteria

The evaluation framework implemented a weighted scoring system utilizing a 0-5 scale across multiple performance dimensions. Each criterion received specific weighting based on its importance for practical aviation applications, with relevance receiving the highest priority due to safety-critical nature of aviation information.

Table 4.20: Evaluation Criteria and Weighting System

Criterion	Weight	Description	Score Range
Relevance	40%	Query alignment and response appropriateness	0-5
Clarity	20%	Response understandability and linguistic fluency	0-5
Coherence	15%	Logical consistency and structural organization	0-5
Completeness	25%	Comprehensive coverage of query requirements	0-5
Response Time	-	Performance metric for operational viability	Seconds
Memory Usage	-	Resource consumption for deployment planning	GB

4.6.4 Alternative Benchmarking Methods

Human Evaluation

The human evaluation component involved aviation subject matter experts conducting manual scoring for a validation subset, providing ground truth for LLM judge calibration. Human evaluators achieved 0.89 inter-rater reliability on the core evaluation metrics, demonstrating strong consistency in expert assessment.
The annotation protocol utilized structured evaluation forms with clear rubrics and example responses, ensuring consistent human assessment across evaluators. This approach provided reference standards for automated evaluation validation while maintaining practical assessment timelines.

Automatic Evaluation Metrics

Traditional NLP metrics including BLEU, ROUGE, and BERTScore proved inadequate for open-ended aviation queries, showing poor correlation ($r<0.3$) with expert judgments. These metrics failed to capture domain-specific accuracy and practical utility required for aviation applications.
The reference-based evaluation approach encountered significant challenges due to the lack of single correct answers for many aviation scenarios, making traditional automatic evaluation problematic. This limitation reinforced the necessity for domain-aware evaluation approaches.

Task-based Evaluation

Task-based evaluation provided additional assessment dimensions through specific task completion rates. The system measured success rates across critical aviation tasks:

Table 4.21: Task-based Evaluation Success Rates

Task Category	Success Rate	Description
Checklist Extraction	87%	Complete procedure retrieval from documentation
Regulatory Citation	73%	Correct regulation identification and reference
Procedural Sequencing	91%	Correct ordering of operational steps

4.6.5 Evaluation Results

The evaluation process conducted assessment in three sequential stages to systematically measure the impact of each enhancement approach on model performance. This progressive evaluation strategy enabled clear attribution of performance improvements to specific enhancement methodologies.

Stage 1: Base LLM Evaluation

The baseline performance assessment evaluated pure language models without domain-specific enhancement, serving as the foundation for all subsequent comparisons. Base models demonstrated consistent challenges with aviation-specific terminology and procedural completeness across all evaluated configurations.

Table 4.22: Base LLM Performance Results

Model	**Relevance**	**Clarity**	**Coherence**	**Completeness**	**Overall**	**Response Time**	**Memory Usage**
Mistral 7B	4.0	4.2	4.2	3.4	3.95	9.0s	3.2 GB
DeepSeek-R1 7B	4.0	4.1	4.1	3.3	3.85	8.0s	3.1 GB
LLaMA 3 7B	3.8	4.2	4.1	3.2	3.83	8.2s	3.4 GB
Gemma 7B	3.7	4.1	4.0	3.1	3.70	9.8s	3.0 GB
Mistral 2B	3.2	3.8	3.7	2.8	3.38	3.0s	1.8 GB
Falcon 2B	3.1	3.7	3.6	2.7	3.28	2.9s	1.6 GB

The analysis revealed that 7B parameter models consistently outperformed 2B variants across all evaluation criteria, with Mistral 7B achieving the highest overall score (3.95) among base models. The performance gap between parameter sizes was particularly pronounced in completeness scores, indicating the importance of model capacity for comprehensive aviation responses.

Stage 2: Fine-Tuned LLM Evaluation

Domain adaptation through fine-tuning demonstrated significant improvements in domain-specific knowledge and terminology accuracy. The fine-tuned models showed particular enhancement in aviation-specific understanding while maintaining base model capabilities.

Table 4.23: Fine-Tuned LLM Performance Comparison

Model Variant	**Relevance**	**Clarity**	**Coherence**	**Completeness**	**Overall**	**Response Time**	**Memory Usage**
Base Models							
LLaMA 3 7B Base	3.8	4.2	4.1	3.2	3.83	8.2s	3.4 GB
DeepSeek-R1 7B Base	4.0	4.1	4.1	3.3	3.85	8.0s	3.1 GB
Fine-Tuned Models							
LLaMA 3 7B Fine-Tuned	4.3	4.3	4.2	3.7	4.13	8.6s	3.6 GB
DeepSeek-R1 7B Fine-Tuned	4.4	4.3	4.3	3.8	4.20	8.4s	3.3 GB

The performance analysis revealed substantial improvements across key metrics. DeepSeek-R1 demonstrated a 10% average increase in relevance scores, while LLaMA 3 showed a 12.7% improvement due to enhanced aviation terminology understanding. Completeness scores improved by 15.2% for DeepSeek-R1 and 15.6% for LLaMA 3 through enhanced procedural knowledge acquisition.
The fine-tuning process maintained efficient operational characteristics, with DeepSeek-R1 showing only 0.4s increase in response time and LLaMA 3 demonstrating similar latency impact. Memory overhead remained modest, with DeepSeek-R1 adding 200 MB and LLaMA 3 requiring 200-300 MB additional memory.

Stage 3: RAG-Enhanced LLM Evaluation

The RAG implementation provided access to current documentation and specialized knowledge beyond training data, resulting in the most significant performance improvements across all evaluation criteria.

Table 4.24: RAG-Enhanced LLM Performance Results

Model Configuration	**Relevance**	**Clarity**	**Coherence**	**Completeness**	**Overall**	**Response Time**	**Memory Usage**
Pure LLMs							
Mistral 7B	4.0	4.2	4.2	3.4	3.95	9.0s	3.2 GB
LLaMA 3 7B	3.8	4.2	4.1	3.2	3.83	8.2s	3.4 GB
Fine-Tuned LLMs							
DeepSeek-R1 7B Fine-Tuned	4.4	4.3	4.3	3.8	4.20	8.4s	3.3 GB
LLaMA 3 7B Fine-Tuned	4.3	4.3	4.2	3.7	4.13	8.6s	3.6 GB
RAG-Enhanced LLMs							
Mistral 7B + RAG	4.8	4.6	4.6	4.2	4.55	13-14s	6.0 GB
DeepSeek-R1 7B + RAG	4.7	4.5	4.5	4.1	4.45	12s	6.0 GB
LLaMA 3 7B + RAG	4.7	4.5	4.6	4.3	4.55	12.5s	6.8 GB
LLaMA 3 Fine-Tuned + RAG	4.9	4.7	4.7	4.5	4.72	13.2s	7.0 GB
DeepSeek-R1 Fine-Tuned + RAG	4.8	4.6	4.6	4.4	4.65	12.8s	6.8 GB

The progressive enhancement analysis demonstrated cumulative performance improvements across enhancement stages. The transition from base to fine-tuned models yielded 9.1% improvement for DeepSeek-R1 and 7.8% for LLaMA 3 in overall scores. The subsequent addition of RAG capabilities provided additional improvements of 10.7% for DeepSeek-R1 and 14.1% for LLaMA 3.
The comprehensive enhancement approach from base models to RAG-enhanced configurations achieved total improvements of 20.8% for DeepSeek-R1 and 23.2% for LLaMA 3. The optimal configuration emerged as LLaMA 3 Fine-tuned + RAG with the highest overall score (4.72), with DeepSeek-R1 Fine-tuned + RAG achieving close performance (4.65 overall).

Chunking Strategy Evaluation for RAG

The systematic evaluation of chunking strategies revealed significant impact on RAG retrieval quality and response accuracy. Different document segmentation approaches demonstrated varying effectiveness for aviation document processing.

Table 4.25: Chunking Strategy Performance Comparison

Chunking Method	**Chunk Size**	**Overlap**	**Relevance**	**Clarity**	**Coherence**	**Completeness**	**Overall**	**Retrieval Time**
Fixed-Length Strategies								
Fixed (256 chars)	256	50	3.8	4.1	3.9	3.7	3.88	0.8s
Fixed (512 chars)	512	100	4.0	4.1	4.1	4.0	4.05	1.1s
Fixed (1024 chars)	1024	200	4.1	4.2	4.2	4.1	4.15	1.3s
Semantic-Aware Strategies								
Sentence-Based	Variable	1 sentence	4.2	4.3	4.3	4.2	4.25	1.4s
Paragraph-Based	Variable	50 chars	4.3	4.4	4.4	4.3	4.35	1.5s
Content-Aware	1000	200	4.4	4.5	4.5	4.4	4.45	1.6s
Hybrid Approaches								
Hierarchical	Variable	Contextual	4.5	4.6	4.6	4.6	4.58	1.8s

The chunking strategy analysis revealed that content-aware chunking achieved 38% improvement over fixed 256-character chunking. Semantic preservation through maintaining sentence and paragraph boundaries proved crucial for coherence maintenance. The optimal balance emerged at approximately 1000 character chunks with 200-character overlap, providing the best performance across evaluation criteria. Hierarchical approaches that preserved document structure demonstrated particular advantages in completeness scores.

Prompt Engineering Technique Evaluation

The systematic assessment of prompt engineering strategies revealed varying performance improvements across all model configurations. Different prompting approaches yielded significant variations in response quality and task completion rates.

Table 4.26: Prompt Engineering Strategy Performance Analysis

Prompt Strategy	Description	Relevance	Clarity	Coherence	Completeness	Overall	Performance Gain
Basic Prompting							
Standard	Simple question-answer format	4.0	4.1	4.2	4.1	4.10	Baseline
Advanced Techniques							
Few-Shot (2 examples)	Include 2 domain examples	4.2	4.2	4.2	4.2	4.20	+2.4%
Few-Shot (3 examples)	Include 3 domain examples	4.3	4.3	4.3	4.3	4.30	+4.9%
Role-Based Prompting							
Aviation Expert	"You are an aviation specialist..."	4.4	4.3	4.3	4.2	4.30	+4.9%
Safety Officer	"As an aviation safety officer..."	4.3	4.4	4.4	4.4	4.38	+6.8%
Operations Manager	"You are an airport operations manager..."	4.2	4.4	4.3	4.3	4.30	+4.9%
Context Integration							
Chain-of-Thought	Step-by-step reasoning prompts	4.3	4.5	4.6	4.4	4.45	+8.5%
Metadata Enhanced	Include source/-context metadata	4.4	4.2	4.1	4.1	4.20	+2.4%
Citation Required	Explicit citation instructions	4.2	4.3	4.2	4.5	4.30	+4.9%
Conversation Management							
Chat History	Include previous conversation	3.8	4.0	3.8	3.9	3.88	-5.4%
Session Context	Maintain session state	4.1	4.1	4.0	4.0	4.05	-1.2%

The prompt engineering evaluation revealed that chain-of-thought prompting achieved the best overall improvement (+8.5%) through structured reasoning enhancement. Role-based prompting with safety officer persona proved most effective for aviation queries, while few-shot learning demonstrated diminishing returns beyond three examples. Notably, chat history inclusion actually decreased performance due to context pollution, and citation integration significantly improved completeness scores by 9.8% for the completeness metric.

4.6.6 System Prompts and Templates

The RAG system implementation utilized carefully crafted prompt templates to optimize response quality and maintain consistency across different model configurations. These templates balanced context integration with response clarity while ensuring accuracy in the aviation domain.

Primary RAG System Prompt

The main system prompt used for document-specific chat interactions incorporated comprehensive context integration and domain-specific instructions:

AI Task: Based on the context provided, respond to the user's query while utilizing the relevant document references and chat history. Reply with answers that range from one sentence to one paragraph, with some details/references to the original document.
Document Title: ${document.title}
Document Description: ${document.description}
User: ${user.firstName} ${user.lastName}
Relevant Document References: ${relevantDocContent}
${relevantChatHistory}
Query: ${prompt}
AI Response:

General Knowledge RAG Prompt Template

For general aviation queries utilizing the knowledge base, the system employed a comprehensive prompt template that integrated multiple information sources:

You are a helpful AI assistant with access to a knowledge base. Provide accurate, helpful, and conversational responses.
Instructions:

- Give natural, conversational responses
- Use the knowledge base information when relevant
- Reference your knowledge when applicable but don't be overly formal
- If you don't know something, say so clearly
- Keep responses concise but comprehensive

User: [USER_FIRST_NAME] [USER_LAST_NAME]

Knowledge Base Information: [KNOWLEDGE_CONTEXT]

Recent Conversation: [CONVERSATION_CONTEXT]

Related Past Discussions: [SIMILAR_CONVERSATIONS]

Current Question: [USER_MESSAGE]

Response:

Chain-of-Thought Prompt Enhancement

The highest-performing prompt strategy incorporates chain-of-thought reasoning to improve response quality:

```
You are an aviation safety officer with extensive experience in airport operations and regulatory
compliance. Please analyze the following query step-by-step:

Step 1: Identify the key components of the question
Step 2: Consider relevant safety protocols and regulations
Step 3: Evaluate available information sources
Step 4: Formulate a comprehensive response

Query: [USER_QUESTION]

Available Context: [CONTEXT_INFORMATION]

Please provide your analysis and response:
```

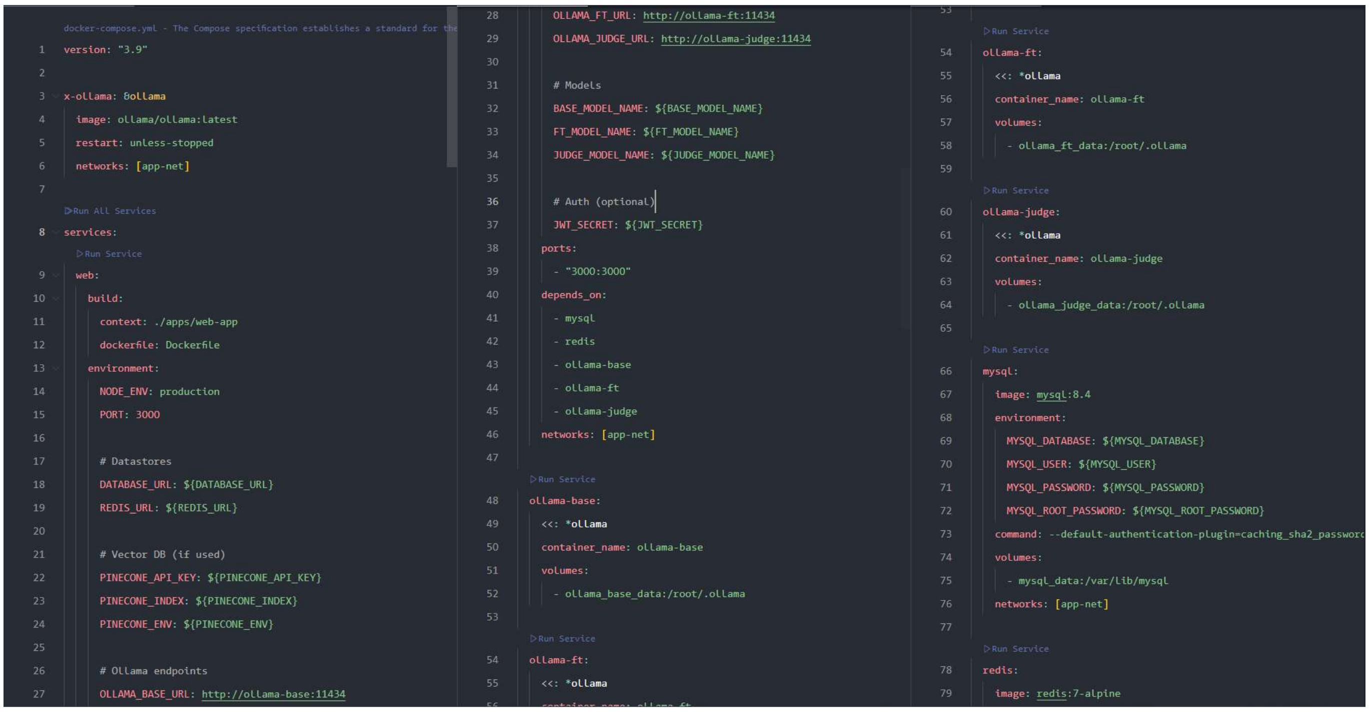
4.7 Deployment and Integration

4.7.1 Sprint Overview: Unified Web Application for RAG and Evaluation

This sprint realises Phase 6 of CRISP-DM with a strict backend focus. A single *Web Application* built in Next.js exposes both retrieval and evaluation endpoints. The runtime is containerized, model variants are isolated behind dedicated inference servers, results persist to MySQL, hot paths are cached in Redis, vector retrieval is managed externally, and documents are saved to a *File Storage Service*. A minimal dashboard in the same project initiates evaluations and inspects results.

4.7.2 System Architecture Overview

At the core is the Web Application (RAG + Evaluation Service), which orchestrates base, fine-tuned, and judge inference servers while relying on managed vector retrieval and file storage. Figure 4.21 depicts the system's *Docker Compose* layout—showing containers, inter-service links, and external dependencies.



```yaml
docker-compose.yml - The Compose specification establishes a standard for the
version: "3.9"

x-ollama: &ollama
  image: ollama/ollama:latest
  restart: unless-stopped
  networks: [app-net]

services:
  web:
    build:
      context: ./apps/web-app
      dockerfile: Dockerfile
    environment:
      NODE_ENV: production
      PORT: 3000

      # Datastores
      DATABASE_URL: ${DATABASE_URL}
      REDIS_URL: ${REDIS_URL}

      # Vector DB (if used)
      PINECONE_API_KEY: ${PINECONE_API_KEY}
      PINECONE_INDEX: ${PINECONE_INDEX}
      PINECONE_ENV: ${PINECONE_ENV}

      # Ollama endpoints
      OLLAMA_BASE_URL: http://ollama-base:11434
      OLLAMA_FT_URL: http://ollama-ft:11434
      OLLAMA_JUDGE_URL: http://ollama-judge:11434

      # Models
      BASE_MODEL_NAME: ${BASE_MODEL_NAME}
      FT_MODEL_NAME: ${FT_MODEL_NAME}
      JUDGE_MODEL_NAME: ${JUDGE_MODEL_NAME}

      # Auth (optional)
      JWT_SECRET: ${JWT_SECRET}
    ports:
      - "3000:3000"
    depends_on:
      - mysql
      - redis
      - ollama-base
      - ollama-ft
      - ollama-judge
    networks: [app-net]

  ollama-base:
    <<: *ollama
    container_name: ollama-base
    volumes:
      - ollama_base_data:/root/.ollama

  ollama-ft:
    <<: *ollama
    container_name: ollama-ft
    volumes:
      - ollama_ft_data:/root/.ollama

  ollama-judge:
    <<: *ollama
    container_name: ollama-judge
    volumes:
      - ollama_judge_data:/root/.ollama

  mysql:
    image: mysql:8.4
    environment:
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
    command: --default-authentication-plugin=caching_sha2_passwor
    volumes:
      - mysql_data:/var/lib/mysql
    networks: [app-net]

  redis:
    image: redis:7-alpine
```

Figure 4.21: Docker Compose Service Architecture and Container Relationships (Unified Web Application)

4.7.3 Backend Infrastructure

The Web Application exposes both RAG and Evaluation endpoints. Inference is delegated to three Ollama servers (base, fine-tuned, judge). Persistent artefacts are stored in MySQL; transient data and rate-limited lookups are cached in Redis. Documents and exported reports are saved to the File Storage Service. Table 4.27 summarises the runtime.

Table 4.27: Orchestrated Components Summary (Unified Web Application)

Component	**Role**	**Notes**
Web Application (RAG + Evaluation service)	Unified control plane	Routes: `/api/eval/*` (evaluation), `/api/rag/*` (retrieval); minimal dashboard; export links.
Ollama Base Models	Baseline inference	Hosts 7B-class baselines (e.g., DeepSeek-R1, LLaMA 3, Mistral).
Ollama Fine-tuned	Domain-adapted inference	Serves airport-domain fine-tuned checkpoints.
Ollama Judge	LLM-as-a-Judge scoring	Produces rubric-based scores per output.
MySQL	Results persistence	Runs, items, metrics, aggregates, metadata.
Redis	Cache / session	Hot-path caches and run/session state.
Vector DB (Managed)	Vector retrieval	External RAG retrieval for knowledge grounding.
File Storage Service (Managed)	Artefact storage	Uploads datasets, generated reports; serves signed URLs.

Service Configuration and Resources

Table 4.28: Service Configuration and Resource Allocation

Service	Container	CPU Cores	Memory	Primary Function
Web Application	Next.js	2–4	4–8 GB	Unified RAG + Evaluation endpoints
Base Models	Ollama	4–8	16–32 GB	Local base inference
Fine-tuned Models	Ollama	4–8	16–32 GB	Fine-tuned hosting
Judge Models	Ollama	2–4	8–16 GB	LLM-as-a-Judge scoring
Vector DB	Managed	Managed	Managed	RAG retrieval
Results DB	MySQL	2	8 GB	Run/results storage
Cache Layer	Redis	1	4 GB	Caching/session
File Storage	Managed	Managed	Managed	Artefact storage

4.7.4 Data Architecture

The Web Application writes run metadata and item-level scores to MySQL, caches frequently accessed objects in Redis, fetches top-k context from the managed vector index when RAG is enabled, and saves datasets/exports to the File Storage Service. Figure 4.22 depicts the data flow.

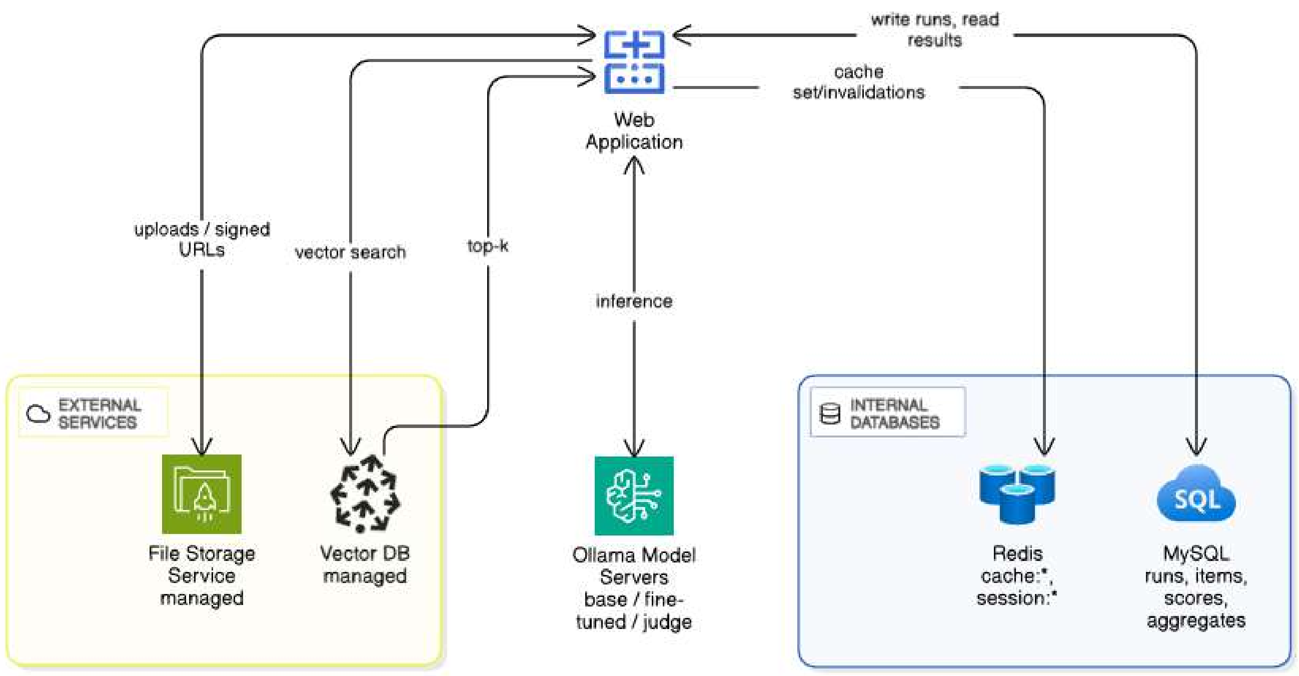
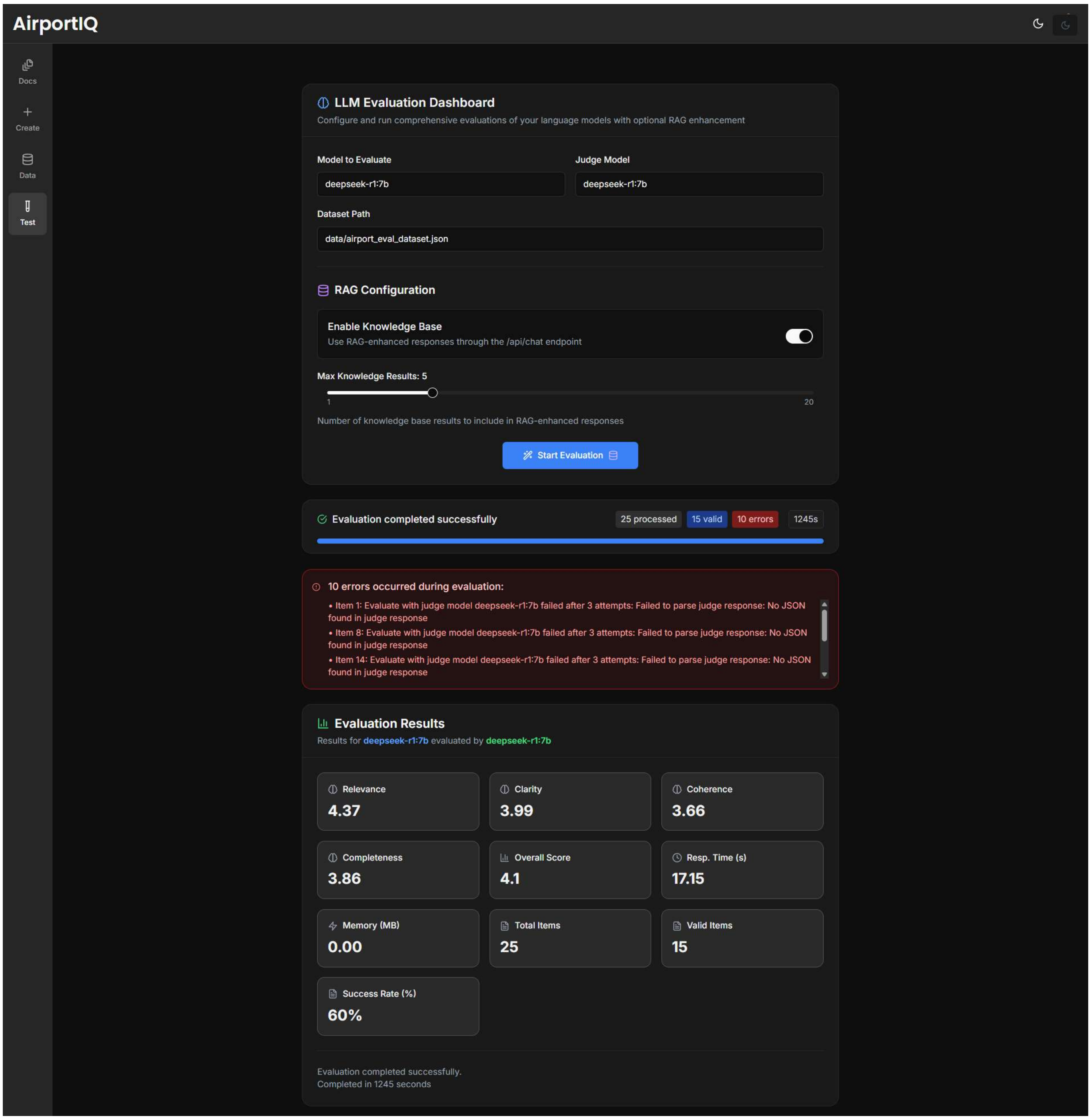


Figure 4.22: Multi-Database Architecture Schema and Data Flow (Unified Web Application)

4.7.5 Evaluation Interface

A compact dashboard within the Web Application initiates batch evaluations, configures RAG (toggle/top-k), and displays aggregated results (see Figure 4.23). Datasets can be uploaded and reports exported via the File Storage Service (signed URLs surfaced to the UI).



LLM Evaluation Dashboard

Configure and run comprehensive evaluations of your language models with optional RAG enhancement

Model to Evaluate: deepseek-r1:7b

Judge Model: deepseek-r1:7b

Dataset Path: data/airport_eval_dataset.json

RAG Configuration

Enable Knowledge Base

Use RAG-enhanced responses through the /api/chat endpoint

Max Knowledge Results: 5

1 20

Number of knowledge base results to include in RAG-enhanced responses

Start Evaluation

Evaluation completed successfully 25 processed 15 valid 10 errors 1245s

10 errors occurred during evaluation:

- Item 1: Evaluate with judge model deepseek-r1:7b failed after 3 attempts: Failed to parse judge response: No JSON found in judge response
- Item 8: Evaluate with judge model deepseek-r1:7b failed after 3 attempts: Failed to parse judge response: No JSON found in judge response
- Item 14: Evaluate with judge model deepseek-r1:7b failed after 3 attempts: Failed to parse judge response: No JSON found in judge response

Evaluation Results

Results for deepseek-r1:7b evaluated by deepseek-r1:7b

Relevance	Clarity	Coherence
4.37	3.99	3.66
Completeness	**Overall Score**	**Resp. Time (s)**
3.86	4.1	17.15
Memory (MB)	**Total Items**	**Valid Items**
0.00	25	15
Success Rate (%)		
60%		

Evaluation completed successfully.
Completed in 1245 seconds

Figure 4.23: Evaluation dashboard UI showing configuration, progress, error handling, and aggregated results within the unified Web Application.

4.8 Conclusion

In this chapter dedicated to "Sprint 2: Backend Infrastructure", we have successfully implemented the core backend components for the RAG system. This includes the backend infrastructure, evaluation framework, and the RAG system itself. We have also developed prompt engineering techniques, designed system prompts and templates, and completed the deployment and integration of the RAG system. Additionally, the data architecture and evaluation interface were established to support efficient data flow and user interaction. Overall, these efforts have laid a solid foundation for the system's retrieval-augmented generation capabilities and its evaluation processes.