SCIRAG: A RETRIEVAL-FOCUSED FINE-TUNING STRATEGY FOR SCIENTIFIC DOCUMENTS

SCIRAG: A RETRIEVAL-FOCUSED FINE-TUNING STRATEGY FOR SCIENTIFIC DOCUMENTS

By CHARANGAN VASANTHARAJAN,

A Thesis Submitted to the School of Graduate Studies in Partial Fulfillment of the Requirements for the Degree Master of Applied Science

April 2025

McMaster University

MASTER OF APPLIED SCIENCE (2025)

Hamilton, Ontario, Canada (Electrical and Computer Engineering)

TITLE:	SciRAG: A Retrieval-Focused Fine-Tuning Strategy for Scientific Documents
AUTHOR:	Charangan Vasantharajan BSc in Computer Science and Engineering, University of Moratuwa, Sri Lanka
SUPERVISOR:	Prof. Thia Kirubarajan
NUMBER OF PAGES:	xiv, 90

Abstract

Large Language Models (LLMs) have achieved remarkable success in general-purpose natural language understanding and generation. However, their effectiveness diminishes in scientific and technical domains, where documents contain dense mathematical notation, complex layouts, and specialized terminology. These characteristics pose significant challenges for traditional LLM pipelines, often resulting in hallucinated outputs, misinterpretation of formulas, and failures in retrieving relevant context.

This thesis introduces SciRAG, a Retrieval-Focused Fine-Tuning Strategy designed specifically for scientific documents. SciRAG combines structure-preserving document parsing, context-aware chunking, and domain-adapted fine-tuning using Low-Rank Adaptation (LoRA) to enhance an LLM's ability to understand and generate scientifically accurate content. The system incorporates a custom Retrieval-Augmented Generation (RAG) framework that supports semantic alignment of mathematical expressions and technical language across large corpora.

Experimental evaluations demonstrate that SciRAG achieves strong performance in scientific question answering and mathematical reasoning. Notably, the model attains 70% accuracy on the GSM8k benchmark, alongside high retrieval and generation quality, achieving a Context Recall score of 0.85, Factual Correctness of 0.45, Faithfulness of 0.45, and Semantic Similarity of 0.94. These results underscore SciRAG's

effectiveness in bridging the gap between general-purpose LLMs and domain-specific, mathematically grounded language understanding.

Acknowledgments

I would like to express my heartfelt gratitude to my family for their unwavering support and encouragement throughout my academic journey. My deepest thanks go to my mother, and my brothers, whose love and understanding created a nurturing environment that made this work possible.

I am also profoundly grateful to my friends and colleagues who offered insightful feedback and moral support, especially during the challenging phases of this project. Special thanks are due to Dr. Ratnasingham Tharmarasa for his valuable suggestions.

Finally, I extend my sincere appreciation to my supervisor, Prof. Thia Kirubarajan, whose expert guidance, constructive criticism, and continuous encouragement were instrumental in shaping this thesis. His mentorship has been invaluable to my personal and professional development.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Recent progress in Large Language Models (LLMs), exemplified by systems such as ChatGPT and Gemini, has brought about significant breakthroughs in tasks ranging from open-domain question answering to dialogue management [4]. These models, pre-trained on massive and diverse web-scale corpora, showcase a remarkable capacity to understand and generate human-like text. In many general-domain tasks, including even some technical question answering, they perform impressively due to the breadth of their training data.

However, the limitations of general-purpose LLMs become apparent when addressing highly specialized domains. While they exhibit statistical fluency, they often fail silently when confronted with unfamiliar concepts, particularly in scientific and mathematical fields [24, 13]. Unlike general text, scientific documents typically feature

complex layouts (multi-column formats, embedded tables and figures), dense mathematical notation (often represented as LaTeX code or image regions), and domain-specific jargon. Such characteristics pose significant challenges: tokenizers trained on general text frequently fragment mathematical expressions, and pre-trained LLMs lack sufficient grounding to reliably interpret or generate domain-specific scientific content [27].

Consider a scenario where an LLM is tasked with answering a question based on a corpus of over 10,000 scientific PDFs rich in mathematical formulas. Without prior exposure to the specific documents or the ability to semantically parse complex mathematical notation, even state-of-the-art open-source models often fail: they misinterpret formulas, cannot search effectively across the entire corpus, and generate incorrect or vague answers. This inability stems not only from the lack of direct exposure but also from the inherent difficulty in representing structured mathematics purely as linear text.

To mitigate hallucinations and knowledge gaps, many researchers have adopted *Retrieval-Augmented Generation (RAG)* frameworks [7]. RAG techniques retrieve relevant external text and provide it to the model as additional context during generation, helping to ground the output in verifiable information. However, while RAG substantially reduces hallucinations in open-domain applications, it is not a panacea. Scientific documents present unique hurdles: poor extraction quality can garble critical formulas, retrieval embeddings may overlook technical context, and models unfamiliar with domain-specific concepts may misinterpret even correctly retrieved data. Without structured document parsing and domain-aware embeddings, retrieval pipelines risk silently failing.

Meanwhile, *fine-tuning* an LLM on domain-specific datasets offers a complementary pathway by enriching the model's internal knowledge representations [30, 25]. Fine-tuning enables models to become more sensitive to the specialized vocabulary and mathematical structures characteristic of scientific discourse. Yet, standalone fine-tuning also has limitations: it does not ensure that outputs are grounded in the latest factual knowledge unless combined with dynamic retrieval mechanisms.

Given these challenges, a hybrid approach—integrating domain-adapted fine-tuning with robust retrieval mechanisms—is crucial. Only a few recent works have explored this combination explicitly [24, 28], leaving an important gap in the design of LLM pipelines for scientific domains. Addressing this gap demands a strategy that (i) preserves structured mathematical and technical content during parsing, (ii) fine-tunes models on carefully curated domain-specific data, (iii) ensures context-aware chunking for retrieval, and (iv) evaluates end-to-end performance systematically.

In this thesis, we propose a cohesive pipeline that bridges this gap—combining LoRA-based fine-tuning with structured document parsing and a RAG framework tailored to scientific and technical applications such as target tracking. Our approach results in more faithful, relevant, and mathematically precise outputs, demonstrating substantial improvements over existing methods.

1.2 Problem Statement

Despite the promise of pairing domain-specific fine-tuning with retrieval mechanisms, several persistent hurdles remain—especially in scientific and technical domains:

- **Specialized Vocabulary:** Scientific documents often introduce specialized jargon and abbreviations that general-purpose LLMs fail to recognize or properly contextualize.
- **Mathematical Complexity:** Equations embedded as LaTeX code or images are difficult to tokenize and comprehend without explicit layout and syntax preservation.
- **Context Overload:** Scientific corpora are vast, and naively retrieving and reading through large volumes of text is computationally infeasible for standard models.
- **Hallucinations in High-Stakes Settings:** Even fine-tuned LLMs may generate unsupported claims when lacking retrieval grounding, which is critical in domains where precision and verifiability are paramount.

Moreover, RAG-based methods, while helpful in reducing hallucinations, encounter notable limitations:

- Poor extraction pipelines result in missing or corrupted formulas, undermining retrieval quality.
- Embedding models often fail to capture domain-specific semantics, leading to weak retrieval matches.
- Standard RAG systems lack mechanisms to resolve mathematical symbols across documents, critically impairing question-to-context alignment.

In practice, existing pipelines often exhibit failure modes such as:

- **Noisy Extraction:** Mathematical content is lost or distorted during preprocessing.
- **Embedding Gaps:** Technical nuance is missed during query-document similarity computation.
- **Generalization Failures:** Models produce vague or irrelevant completions due to insufficient domain-specific tuning.

To address these challenges, this thesis advocates for a pipeline that:

- Fine-tunes LLMs explicitly on mathematical and technical question answering;
- Parses documents to preserve LaTeX structure and context;
- Implements context-aware document chunking to enhance retrieval fidelity;
- Designs an evaluation loop to assess the integration of fine-tuning and retrieval components.

Through this approach, we aim to bridge the gap between general-purpose language understanding and domain-specific, mathematically precise knowledge generation.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows:

- **Chapter 2: Literature Review**

  Surveys the evolution of chatbots, outlines the development of fine-tuning methods for LLMs, and discusses the merits and limitations of RAG approaches in specialized domains.

- **Chapter 3: Data Collection and Preprocessing**

  Explains how scientific documents are sourced and prepared, with a focus on extracting, converting, and structuring complex mathematical and textual information.

- **Chapter 4: LLM Fine-Tuning**

  Describes multiple strategies—ranging from full to parameter-efficient fine-tuning—and their relevance to scientific content, considering domain-specific challenges.

- **Chapter 5: System Design and Architecture**

  Introduces the conceptual framework that integrates fine-tuned models with an external retrieval pipeline to ground responses in authoritative domain knowledge.

- **Chapter 6: Experimental Setup and Evaluation**

  Presents the methodology for assessing the performance of the combined framework, using a mix of quantitative metrics and qualitative analyses.

- **Chapter 7: Discussion and Future Work**

  Reflects on the results, identifies potential enhancements, and proposes avenues for applying such integrated approaches in other high-stakes or data-intensive fields.

- **Chapter 8: Conclusion**

  Summarizes the core findings and reiterates the importance of uniting domain-specific fine-tuning with robust retrieval to better address the complexities of scientific communication.

Chapter 2

Related Papers and Theory

This chapter examines the existing body of knowledge relevant to the development of a Retrieval-Augmented Generation (RAG) chatbot for scientific documents in the target tracking domain. It covers the evolution of chatbots and Large Language Models (LLMs), the principles and applications of RAG, previous implementations and architectures, and the state-of-the-art in document extraction and question-answer generation. Finally, it identifies key gaps in the literature that this research aims to address.

2.1 Large Language Models

Large Language Models (LLMs) have reshaped Natural Language Processing (NLP), enabling applications such as text generation, summarization, question answering, and conversational AI. Rooted in the Transformer architecture, they leverage massive corpora for pre-training and often excel in zero- or few-shot settings. This

section explores the technical underpinnings of LLMs, beginning with the Transformer framework and its core components—tokenization, attention, loss functions, and temperature—followed by a discussion on the rise of large-scale models, their advantages and limitations, and prevalent fine-tuning strategies.

2.1.1 The Transformer Architecture

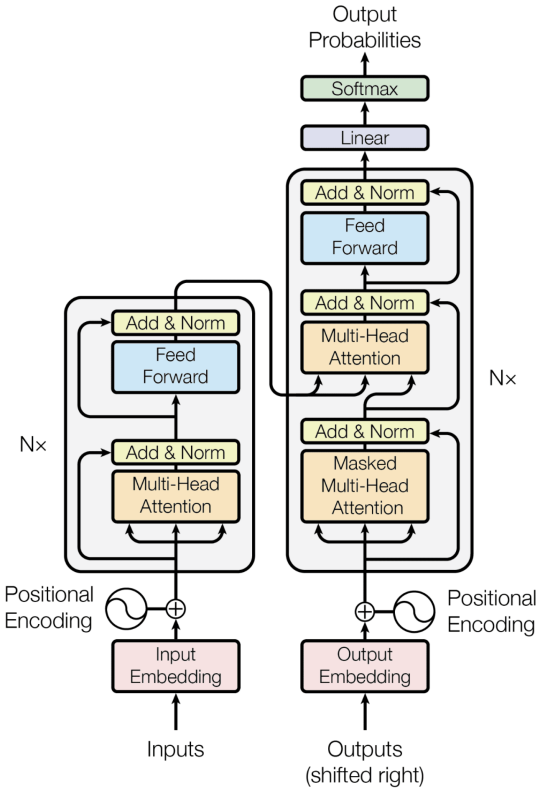


Figure 2.1: High-level schematic of the Transformer architecture, consisting of an encoder and decoder. Each block features multi-head attention, feed-forward layers, and residual connections (adapted from [23]).

Originally introduced by Vaswani et al. [23], the Transformer architecture revolutionized Natural Language Processing (NLP) by eliminating recurrence in favor of

a self-attention mechanism that processes entire sequences in parallel. Structurally, a Transformer can be divided into an encoder and a decoder stack (though variants exist with only an encoder or only a decoder). Each stack is composed of multiple layers containing multi-head attention and position-wise feed-forward networks.

Figure 2.1 illustrates a simplified view of the Transformer. The encoder transforms the input sequence into contextual representations using self-attention, while the decoder generates the output sequence in an autoregressive manner, leveraging both self-attention and cross-attention to the encoder's output.

2.1.1.1 Tokenizer

A critical step in preparing text for Transformer models is *tokenization*, which segments raw text into tokens (subwords, characters, etc.). Common tokenization methods include Byte-Pair Encoding (BPE) [20], WordPiece [26], and SentencePiece [12]. BPE, for instance, iteratively merges the most frequent character pairs into subword units, forming a compact yet expressive vocabulary.

Method	**Input**	**Tokenized Output**
BPE (Byte-Pair Encoding)	`"satellite tracking"`	`"_sate l l ite _track ing"`
WordPiece	`"satellite tracking"`	`"sat ell ite track ##ing"`
SentencePiece	`"satellite tracking"`	`"_satellite _tracking"`

Table 2.1: Examples of how different tokenization methods segment the sentence `"satellite tracking"`.

Table 2.1 provides a hypothetical example of BPE tokenization. Once tokenized, each subword or token is mapped to an integer index and passed into the Transformer's embedding layer.

2.1.1.2 Attention Mechanism

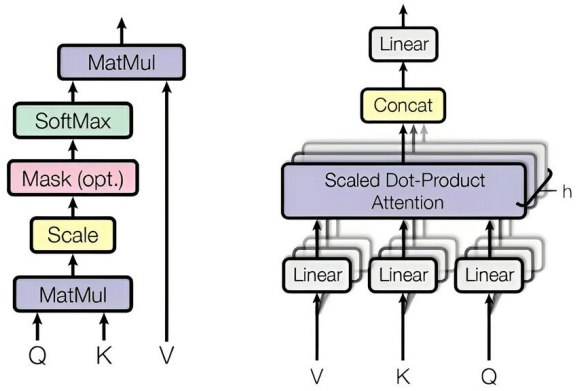


Figure 2.2: Conceptual diagram of the multi-head self-attention process, showcasing how queries, keys, and values are computed and combined in parallel.

The *attention mechanism* is the core innovation of the Transformer [10], enabling it to learn context across entire sequences without relying on recurrence (Figure 2.2). Each token in the sequence attends to every other token, allowing the model to capture both local and long-range dependencies in parallel.

2.1.1.2.1 Scaled Dot-Product Attention. For an input matrix of token embeddings $X \in \mathbb{R}^{n \times d}$, the model derives three learned projections:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V,$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d \times d}$ are trainable parameters, and $Q, K, V \in \mathbb{R}^{n \times d}$ are referred to as *query*, *key*, and *value* matrices, respectively. The *scaled dot-product attention* is computed as:

$$\text{Attention}(Q, K, V) \;=\; \text{softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)\, V,$$

where d_k is the dimensionality of the key vectors (e.g., d/num_heads). The division by $\sqrt{d_k}$ helps stabilize gradients by scaling down the dot products when vectors are high-dimensional.

2.1.1.2.2 Multi-Head Attention. To capture different types of relationships simultaneously, the Transformer employs *multi-head attention*. It partitions Q, K, V into h heads:

$$Q_i = XW_i^Q, \quad K_i = XW_i^K, \quad V_i = XW_i^V,$$

where $i = 1, 2, \ldots, h$. Each head performs scaled dot-product attention independently. The outputs are then concatenated and passed through a linear transformation, typically denoted W^O:

$$\text{MHA}(Q, K, V) = \text{Concat}\Big(\text{head}_1, \ldots, \text{head}_h\Big)\, W^O.$$

Here, $\text{head}_i = \text{Attention}\big(Q_i, K_i, V_i\big)$. This multi-headed design enables the model to learn various types of dependencies (e.g., syntactic vs. semantic) within the same sequence.

2.1.1.2.3 Mathematical Example. Consider a toy sequence $\{x_1, x_2, x_3\}$ with embedding dimension $d = 4$. Let us focus on a single attention head:

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \end{bmatrix} \in \mathbb{R}^{3\times 4}, \quad W^Q, W^K, W^V \in \mathbb{R}^{4\times 4}.$$

Then

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V.$$

If $Q, K, V \in \mathbb{R}^{3\times4}$, we compute

$$QK^T \in \mathbb{R}^{3\times3}, \quad \frac{QK^T}{\sqrt{4}}, \quad \text{and} \quad \text{softmax}\Big(\frac{QK^T}{\sqrt{4}}\Big) \in \mathbb{R}^{3\times3}.$$

The final attended output is

$$\text{softmax}\Big(\frac{QK^T}{2}\Big)V \ \in \mathbb{R}^{3\times4}.$$

Applying multi-head attention repeats this process multiple times (with different W^Q, W^K, W^V matrices per head) before concatenating and projecting.

2.1.1.3 Loss Functions

Transformers typically optimize a *cross-entropy loss* for language modeling, where the goal is to maximize the probability of the correct next token given the context. If y_t is the ground truth token at time step t and $\hat{y}_t$ is the model's predicted distribution over the vocabulary, the cross-entropy loss is:

$$\mathcal{L} = -\sum_{t=1}^{T} \log\big(\hat{y}_t[y_t]\big),$$

where $\hat{y}_t[y_t]$ is the predicted probability of the correct token y_t. In practice, this summation is averaged over all tokens in a batch. Minimizing cross-entropy encourages the model to assign high probability to the ground-truth tokens, thereby improving its predictive accuracy.

2.1.1.3.1 Alternative Objectives. While cross-entropy is the standard for next-token prediction, certain tasks (e.g., summarization or dialogue) may employ additional objectives:

- **Sequence-Level Objectives:** Metrics like ROUGE or BLEU [17, 14] may be optimized via reinforcement learning or minimum risk training to align outputs more closely with desired reference texts.
- **Contrastive Loss:** Used in settings where the model must distinguish correct contexts from distractors, often in retrieval-augmented systems.

Nevertheless, cross-entropy remains the primary choice for most Transformer-based language models, especially in pre-training, where the model learns general-purpose linguistic representations.

2.1.2 Rise of Large Language Models

Since the advent of the Transformer, the field of language modeling has witnessed an exponential increase in model scale. Early models, such as GPT-2 with approximately 1.5 billion parameters, have given way to models like GPT-3, GPT-4, and Llama 2, which boast tens to hundreds of billions of parameters. This dramatic scaling is underpinned by empirical scaling laws that relate model performance to the number of parameters N, the amount of training data D, and the available compute C.

One commonly observed relationship is that the generalization error or loss L decreases following a power law with respect to model size:

$$L(N) \approx L_\infty + k\, N^{-\alpha},$$

where:

- L_∞ is the irreducible error floor,
- k is a constant,
- α is an empirically determined exponent.

Similarly, the benefit of additional training data can be described by:

$$L(D) \approx L_\infty + c\, D^{-\beta},$$

where c and β are constants that depend on the model architecture and task. These equations highlight that while increasing model size and training data consistently improves performance, the returns diminish according to a power-law relationship.

Large language models (LLMs) are typically trained using the standard language modeling objective:

$$\mathcal{L}(\theta) = -\frac{1}{|D|} \sum_{x \in D} \log p_\theta(x),$$

where $p_\theta(x)$ denotes the probability that the model with parameters θ assigns to token x and D is the training dataset. As models scale, they not only capture more detailed statistical regularities of the language but also exhibit emergent capabilities such as in-context learning and effective few-shot performance.

Beyond the standard language modeling loss, large-scale systems often incorporate additional objectives to handle the heterogeneity of the training data (for example, when integrating multimodal inputs or specialized tasks). In such cases, the overall

training objective may be formulated as a composite loss:

$$\mathcal{L}_{\text{total}} = \lambda_1 \mathcal{L}_{\text{LM}} + \lambda_2 \mathcal{L}_{\text{aux}},$$

where:

- $\mathcal{L}_{\text{LM}}$ is the standard language modeling (cross-entropy) loss,
- $\mathcal{L}_{\text{aux}}$ represents auxiliary losses (such as retrieval or multimodal alignment objectives),
- λ_1 and λ_2 are hyperparameters that balance the contribution of each component.

The integration of these scaling laws and composite loss functions distinguishes large language models from their smaller, conventional Transformer counterparts. With access to massive compute resources and vast training corpora, LLMs have demonstrated remarkable proficiency across diverse NLP tasks, setting new benchmarks in both generalization and task-specific performance.

2.1.2.1 Decoder-Only Transformers and Their Controllability

A significant subset of LLMs employs a decoder-only architecture, as popularized by the GPT series. In these models, the entire sequence—including both the prompt and the generated tokens—is processed by a stack of causally masked self-attention layers, ensuring that each token is generated based solely on preceding tokens. This autoregressive setup simplifies the architecture while enabling highly flexible text generation.

Key controllability aspects of decoder-only models include:

2.1.2.1.1 Context Length. The *context length* defines the maximum number of tokens the model can consider during generation. Typically determined during pre-training by the design of the positional embeddings, a model with a context length of L can attend to up to L tokens. Recent advancements, such as positional encoding interpolation or extended training with longer sequences, have pushed these limits further, allowing models to handle context lengths well beyond the original design (e.g., from 2048 to 4096 tokens or more). Increasing the context length is critical for maintaining coherence in long passages and for tasks that require extensive background knowledge.

2.1.2.1.2 Temperature. During inference, the model outputs a logit vector $z \in \mathbb{R}^V$ for the next token, where V is the vocabulary size. The *temperature* parameter T scales these logits before applying the softmax:

$$p(x_t \mid x_{<t}) \propto \exp\Big(\frac{z_t}{T}\Big).$$

Lower temperatures ($T < 1$) sharpen the probability distribution, making the model's predictions more deterministic, while higher temperatures ($T > 1$) produce a flatter distribution that introduces greater diversity and randomness in the output. Adjusting the temperature is a straightforward yet powerful means of controlling the creativity and risk of hallucination in generated text.

2.1.2.1.3 Maximum New Tokens. The `max_new_tokens` parameter governs the number of tokens that the model is allowed to generate beyond the provided prompt. This parameter is distinct from the total sequence length (which includes the prompt)

and is essential for managing both computational cost and output length. For example, setting `max_new_tokens` to 100 limits the generation to 100 tokens, regardless of the prompt length. This controllability is crucial for applications with strict output length requirements.

Together, these parameters—context length, temperature, and max new tokens—offer fine-grained control over the behavior of decoder-only Transformers during inference. By tuning these parameters appropriately, practitioners can strike a balance between output diversity and factual accuracy, manage resource usage effectively, and tailor the generation process to the specific requirements of the task at hand.

2.1.3 Advantages and Limitations of LLMs

Large Language Models (LLMs) have set new benchmarks in natural language understanding and generation, largely due to their vast pre-training on diverse datasets and their flexible Transformer-based architectures. However, while they offer significant advantages, they also face limitations—especially when applied to domain-specific tasks. In this section, we outline the primary advantages of LLMs, discuss their inherent limitations, and explore strategies to overcome these challenges for specialized applications.

2.1.3.0.1 Advantages:

- **Broad Knowledge and Zero/Few-Shot Learning:** LLMs capture extensive statistical regularities and world knowledge from massive corpora [29], enabling them to generalize well to a variety of tasks with little to no additional training. This makes them effective for applications ranging from text summarization to

dialogue systems.

- **Flexible Contextual Representations:** Through self-attention mechanisms, LLMs build rich, context-sensitive embeddings that capture complex syntactic and semantic relationships. This flexibility is crucial for tasks requiring nuanced understanding of language.

- **Scalability:** Empirical scaling laws suggest that performance continues to improve with increased model size and training data. This scalability underpins recent breakthroughs in language modeling.

2.1.3.0.2 Limitations:

- **Domain-Specific Performance:** Although pre-trained on large and diverse datasets, LLMs may struggle with technical language, specialized jargon, and complex symbolic representations (e.g., mathematical notation) common in domains such as scientific research, legal texts, or medical literature [1]. These models might produce outputs that are fluent but factually inaccurate or misleading when the domain shifts significantly from their training distribution.

- **Hallucinations:** In generating text, LLMs can produce information that is plausible-sounding but not grounded in any verifiable data. This is particularly problematic in high-stakes or technical applications, where factual accuracy is paramount.

- **Resource Constraints:** The computational and memory requirements for training and inference scale with model size. This can limit the practical deployment of LLMs, especially when extended context lengths or real-time responses

are required.

- **Interpretability and Bias:** The black-box nature of LLMs makes it challenging to interpret their decisions [2]. Additionally, biases present in training data can be amplified, leading to unfair or inappropriate outputs in sensitive contexts.

2.1.3.0.3 Overcoming Domain-Specific Limitations: To mitigate these challenges, several strategies have been proposed and are actively researched:

- **Domain-Specific Fine-Tuning:** By fine-tuning LLMs on specialized corpora, the models can adapt their representations and language generation to better handle technical vocabulary, precise terminologies, and complex structures inherent in specific domains. Techniques such as parameter-efficient fine-tuning (e.g., LoRa, adapters) allow adaptation without updating the entire model, thereby reducing resource overhead.

- **Retrieval-Augmented Generation (RAG):** Integrating external knowledge bases via retrieval mechanisms helps ground the model's outputs in authoritative, up-to-date sources. This approach can reduce hallucinations and improve factual correctness, particularly in domains where accurate information is critical.

- **Customized Preprocessing and Tokenization:** For domains with specialized symbols and notation (e.g., scientific and mathematical texts), tailored tokenization schemes or preprocessing pipelines (e.g., converting equations into LaTeX or specialized markup) can enhance the model's ability to understand and generate domain-specific content.

- **Context-Length Extension Techniques:** Modifying positional encoding schemes (e.g., through interpolation or extrapolation) allows LLMs to handle longer sequences, which is crucial for tasks that require integrating extensive domain-specific information.

2.1.4 Finetuning an LLM

Although large language models (LLMs) are pretrained on vast and diverse corpora, fine-tuning is essential to adapt these models for specialized tasks and domains. Fine-tuning adjusts the model's parameters so that it performs optimally on a specific task, and several paradigms have emerged to achieve this:

2.1.4.0.1 Types of Fine-Tuning:

- **Full Fine-Tuning:** In this approach, all parameters of the pretrained model are updated on a task-specific dataset. While this method can achieve high performance, it is computationally expensive and may lead to overfitting when the fine-tuning dataset is small.

- **Supervised Fine-Tuning (Instruction Tuning):** Supervised fine-tuning involves training the model on a labeled dataset where inputs are paired with desired outputs. A popular variant is instruction tuning, in which the model is exposed to a diverse set of tasks described in natural language. For example, an *instruct tuning* dataset might contain pairs of instructions (or questions) and their corresponding answers, enabling the model to learn how to follow task-specific directives.

- **Parameter-Efficient Fine-Tuning:** Techniques such as adapters, prefix tuning, and Low-Rank Adaptation (LoRA) update only a small subset of the model parameters. This reduces both the computational cost and the risk of overfitting, while still allowing the model to learn task-specific nuances.

2.1.4.0.2 Supervised Fine-Tuning with an Instruction Tuning Dataset: In supervised fine-tuning, the model is trained on a dataset of instruction-response pairs. For instance, consider an instruct tuning dataset where each example is of the form:

```
Input: "Explain the role of tokenization in NLP."
Output: "Tokenization converts raw text into tokens, which are the basic
    units that the model can process."
```

During fine-tuning, the model's objective is typically to minimize the cross-entropy loss over the generated sequence:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t}, I),$$

where I represents the instruction or prompt, and x_t is the token at position t. This process allows the model to learn both the semantics of the instruction and the appropriate style and content of the response.

2.1.4.0.3 Parameter-Efficient Fine-Tuning with LoRA: Low-Rank Adaptation (LoRA) is a prominent parameter-efficient fine-tuning method [9]. Instead of updating all the parameters in the model, LoRA modifies selected weight matrices by injecting low-rank updates as illustrated in Figure 2.3. Consider a weight matrix

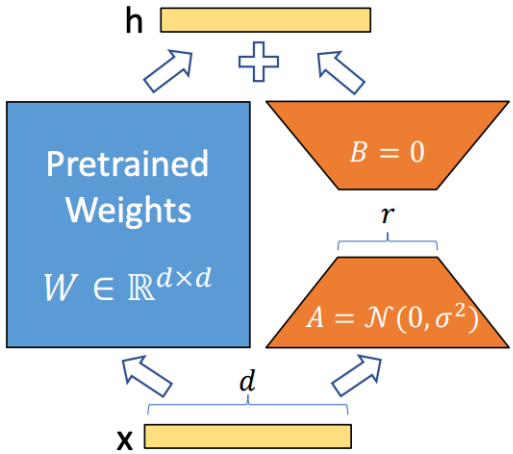


Figure 2.3: Schematic of LoRA: The pretrained weight matrix W is augmented with low-rank matrices A and B such that only A and B are updated during fine-tuning.

$W \in \mathbb{R}^{d \times k}$ in the model. In LoRA, the updated weight matrix W' is defined as:

$$W' = W + \Delta W, \quad \text{with} \quad \Delta W = AB,$$

where:

- $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$ are the trainable low-rank matrices,
- r is the rank, chosen such that $r \ll \min(d, k)$.

This formulation significantly reduces the number of trainable parameters, as the additional parameter count is $d \times r + r \times k$ rather than $d \times k$. Equation **??** summarizes the LoRA update:

$$\boxed{W' = W + AB \quad \text{with} \quad A \in \mathbb{R}^{d \times r},\ B \in \mathbb{R}^{r \times k},\ r \ll \min(d, k).}$$

During fine-tuning, only A and B are updated while W remains frozen. This approach

not only decreases memory usage but also accelerates training.

2.1.4.0.4 Comparison of Fine-Tuning Strategies: Full-parameter fine-tuning takes the LLM "as is" and trains all of its parameters on a given dataset—essentially continuing the supervised training that was used during pre-training [18]. This method typically achieves slightly higher model quality because every parameter is allowed to adjust to the specific task. However, it is computationally expensive, requires substantial GPU memory, and the resulting fine-tuned model checkpoint is large. Additionally, full fine-tuning can be more prone to overfitting, especially when the fine-tuning dataset is limited.

In contrast, Low-Rank Adaptation (LoRA) is a parameter-efficient fine-tuning technique that freezes the original model weights and introduces a small set of additional trainable parameters. These extra parameters form a LoRA checkpoint that is orders of magnitude smaller than a fully fine-tuned model. The key benefits of LoRA include:

- **Resource Efficiency:** Since the base model weights remain frozen, the optimization process requires significantly fewer computational resources. This enables fine-tuning on smaller hardware clusters.
- **Storage and Deployment Efficiency:** The LoRA checkpoint, consisting solely of the low-rank matrices, is very small. In a deployment scenario, the pretrained model can be loaded once and different task-specific LoRA checkpoints can be quickly swapped in, allowing for efficient multi-task serving.
- **Regularization Effect:** Optimizing only a limited set of parameters tends to prevent the model from "forgetting" its general pretrained knowledge, thereby

acting as a form of regularization.

- **No Additional Inference Latency:** Because the low-rank updates can be merged with the frozen weights during inference, LoRA does not introduce extra latency—unlike methods that add additional layers, such as adapter-based approaches.

Empirical studies have shown that LoRA performs on par with, and sometimes even better than, full fine-tuning on various LLMs (e.g., RoBERTa, DeBERTa, GPT-2, GPT-3) despite updating far fewer parameters [21]. The choice between full fine-tuning and LoRA generally depends on several factors:

- **Hardware Resources:** When GPU memory and storage are limited, or when deploying multiple fine-tuned models simultaneously, LoRA is preferable.
- **Model Size:** For very large LLMs (e.g., GPT-3 175B), which are often over-parameterized for many downstream tasks, LoRA's low-rank updates are sufficient to capture the task-specific nuances.
- **Data Availability:** Full fine-tuning can be advantageous when there is a large amount of high-quality task-specific data that benefits from updating the entire model. Conversely, if the dataset is small, the regularization effect of optimizing fewer parameters (as in LoRA) can prevent overfitting.

The following table summarizes the comparison:

In summary, while full fine-tuning offers a slight edge in model quality by allowing every parameter to adapt, LoRA provides significant advantages in computational efficiency, storage, and regularization—making it especially attractive for adapting very large LLMs to specific tasks or domains.

Method	**Trainable Parameters**	**Memory Overhead**
Full Fine-Tuning	$O(d \times k)$	High
LoRA (with rank r)	$O(d \times r + r \times k)$	Low

Table 2.2: Comparison of full fine-tuning versus LoRA-based fine-tuning.

2.1.4.0.5 Practical Considerations: Supervised fine-tuning with an instruct tuning dataset refines the model's ability to follow natural language instructions, while parameter-efficient methods such as LoRA offer a resource-friendly alternative to full fine-tuning. The combination of these strategies has proven effective in adapting large-scale models to specialized domains, improving both task performance and inference efficiency without incurring the full cost of retraining the entire model.

In summary, fine-tuning an LLM can be accomplished through various strategies—each with its own trade-offs. Supervised fine-tuning using an instruct tuning dataset aligns the model with desired outputs, and techniques like LoRA enable efficient adaptation with reduced computational cost, making them particularly well-suited for domain-specific applications.

2.2 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is an approach that integrates external knowledge retrieval into the generative process of large language models (LLMs). Instead of relying solely on the internal parameters of a pretrained model to generate text, RAG systems dynamically fetch relevant information from an external document corpus. This not only grounds the generated outputs in up-to-date and verifiable data but also helps mitigate issues like hallucination and domain-specific inaccuracies [8].

2.2.1 Overview of the RAG Pipeline.

A typical RAG framework consists of two main components:

1. **Retriever:** The retriever indexes a large corpus of documents using vector representations (embeddings). At inference time, given a query (often derived from the user's input or the generation context), the retriever identifies and returns the most relevant documents. This process is typically based on similarity measures computed over high-dimensional embeddings.

2. **Generator:** The generator (usually an LLM) is conditioned not only on the initial prompt but also on the retrieved documents. The combined input helps the model produce outputs that are factually grounded and contextually enriched.

2.2.2 Document Indexing and Retrieval

Document indexing involves converting each document in the corpus into a vector representation using an encoder (which may be the same or a different network from the generator). Suppose we have a set of documents $\{d_1, d_2, \ldots, d_N\}$. Each document d_i is transformed into an embedding $v_i \in \mathbb{R}^d$.

At inference time, given a query q (also embedded as a vector $v_q \in \mathbb{R}^d$), the retriever calculates the similarity between v_q and each v_i using a similarity measure such as cosine similarity:

$$\text{sim}(v_q, v_i) = \frac{v_q \cdot v_i}{\|v_q\| \, \|v_i\|}.$$

Documents are then ranked based on their similarity scores, and the top k documents are selected. In practice, approximate nearest neighbor (ANN) search methods (e.g., using libraries like FAISS or Chroma) are employed to scale this process to large

corpora efficiently.

2.2.3 Mathematical Formulation of Retrieval.

Let $\mathcal{D} = \{v_1, v_2, \ldots, v_N\}$ be the set of document embeddings. For a query embedding v_q, the retrieval process can be formalized as:

$$\text{Retrieve}(v_q) = \arg\underset{v \in \mathcal{D}}{\text{top-}k} \ \frac{v_q \cdot v}{\|v_q\| \, \|v\|}.$$

This operation is typically implemented using efficient vector search algorithms that approximate the nearest neighbors without exhaustively computing the similarity for every document.

2.2.3.0.1 Integrating Retrieval with Generation. Once the top k documents are retrieved, they are concatenated (or otherwise combined) with the original query to form an augmented input:

$$I_{\text{aug}} = \text{concat}(q, d_{i_1}, d_{i_2}, \ldots, d_{i_k}),$$

where d_{i_j} denotes the j-th most relevant document. The generative model is then conditioned on I_{aug} to produce the final output:

$$\hat{y} = \text{Generator}(I_{\text{aug}}).$$

By incorporating external knowledge in this manner, RAG systems are able to generate responses that are both contextually rich and factually grounded.

2.2.4 RAG Evaluation - RAGAS

The RAGAS framework [6] is used to evaluate QA RAG systems along two dimensions: the retrieval component and the generation component. Three key metrics are employed:

2.2.4.1 Faithfulness (FA)

The Faithfulness metric evaluates whether the generated answer $a(q)$ is grounded in the retrieved context $c(q)$. It is computed as:

$$\text{FA} = \frac{\text{Number of statements in the answer inferred from the context}}{\text{Total number of statements in the answer}}.$$

This involves a two-step procedure:

1. An LLM (e.g., GPT-4) extracts individual statements from $a(q)$ using a prompt (see Figure 6 in Es et al., 2023).

2. A second LLM verifies, for each statement, whether it can be inferred from $c(q)$, providing a yes/no verdict.

2.2.4.2 Answer Relevance (AR)

Answer Relevance measures if the generated answer directly addresses the original question. For the answer $a(q)$, an LLM generates five questions $\{q_i\}_{i=1}^{5}$ based on $a(q)$. The AR score is computed as the average cosine similarity between the original question q and each generated question:

$$\text{AR} = \frac{1}{5}\sum_{i=1}^{5}\cos(q, q_i).$$

A high semantic similarity indicates that $a(q)$ is relevant to q.

2.2.4.3 Context Relevance (CR)

Context Relevance evaluates whether the retrieved context $c(q)$ is focused and pertinent to the question q. An LLM is tasked with extracting sentences from $c(q)$ that are important for answering q. The CR score is defined as:

$$\text{CR} = \frac{\text{Number of relevant sentences}}{\text{Total number of sentences in the context}}.$$

If no relevant sentences are identified, or if the context is deemed insufficient, the system returns an "Insufficient Information" flag.

In our experiments, GPT-4 is utilized as the engine behind the evaluation metrics.

2.2.5 Advanced RAG Techniques

Recent research in RAG has explored several advanced strategies to further enhance performance:

- **Dynamic Re-Retrieval:** Some frameworks update the retrieved documents dynamically as the generation proceeds. At each generation step, the model can re-query the corpus with the current partial output, allowing for continuous refinement of the context [22].

- **Differentiable Retrieval:** By making the retrieval process differentiable, it becomes possible to jointly optimize the retriever and the generator during fine-tuning. This end-to-end training can lead to more coherent integration of external knowledge [31].

- **Hybrid Scoring Functions:** While cosine similarity is commonly used, hybrid scoring functions that combine semantic similarity with additional signals (e.g., term frequency–inverse document frequency or learned relevance scores) have been shown to improve retrieval quality.

- **Fusion-in-Decoder:** In this approach [11], the generator is designed to fuse information from multiple retrieved documents within its decoding process. Rather than simply concatenating retrieved texts, the generator selectively attends to different parts of each document, effectively "fusing" external knowledge with the internal model state.

2.2.6 Benefits and Challenges of RAG

RAG systems offer the significant benefit of grounding generated text in external, verifiable sources. This leads to improved factual accuracy and better performance on knowledge-intensive tasks. However, RAG also introduces challenges such as:

- **Latency:** The retrieval step adds additional computation time, which can impact inference speed.

- **Alignment:** Ensuring that the retrieved documents are effectively integrated into the generation process requires careful design and tuning.

- **Scalability:** Efficiently indexing and searching through extremely large corpora demands robust, scalable infrastructure.

In summary, Retrieval-Augmented Generation combines the strengths of external knowledge retrieval and advanced generative modeling. By leveraging efficient indexing techniques, similarity measures such as cosine similarity, and innovative strategies

for integrating retrieved information, RAG systems represent a powerful paradigm for building LLMs that are both knowledgeable and reliable.

2.3 Document Processing for Scientific Documents

Efficient document extraction is a critical pre-processing step for building retrieval-augmented systems, especially when the target documents are scientific papers that often include complex layouts, figures, and mathematical equations. The goal of this stage is to convert PDFs and other document formats into structured representations (e.g., text or markdown) that can be indexed into a vector database.

2.3.1 Traditional PDF Extraction Methods

Conventional tools such as `pypdf` and `pdfplumber` are widely used for extracting text from PDFs. These libraries typically operate by parsing the underlying text layer and basic layout information:

- **pypdf:** Provides functionality to extract text, metadata, and sometimes the structure of the PDF. However, it often struggles with preserving the original formatting, especially in documents with multiple columns or non-standard layouts.
- **pdfplumber:** Offers more advanced features for layout analysis and can capture spatial relationships between text blocks. This is useful for documents with tables or side-by-side text, but it still may not accurately handle embedded equations or figures common in scientific literature.

While these tools are effective for general-purpose extraction, they often fail to capture the full fidelity of scientific documents, where the precise layout of equations, figures, and specialized notation is crucial.

2.3.2 Advanced Extraction Using the Facebook Nougat Model

To address the limitations of traditional OCR tools, the Facebook Nougat model was developed to robustly extract structured content from scientific PDFs by converting them into Markdown. Nougat is specifically designed to handle complex layouts—including multi-column text, embedded figures, and mathematical equations—by leveraging a deep neural architecture that combines convolutional feature extraction with transformer-based sequence modeling [3] as depicted in Figure 2.4.

At a high level, Nougat's extraction pipeline comprises three stages:

1. **Feature Extraction:** The model first processes the input PDF page as an image $I \in \mathbb{R}^{3\times H\times W}$. A convolutional neural network (CNN) backbone F extracts visual features from I, resulting in a feature map:

$$E = F(I), \quad E \in \mathbb{R}^{n\times d},$$

where n is the number of spatial regions (or patches) and d is the feature dimension. These features capture both the textual and layout information present in the document.

2. **Contextual Encoding:** The extracted features are then enriched with spatial and sequential information using positional embeddings P. The sum $E + P$ is fed into a transformer encoder that models both the local and global context of

the document:

$$H = \text{TransformerEncoder}(E + P), \quad H \in \mathbb{R}^{n \times d}.$$

This encoder is adept at capturing long-range dependencies and the spatial relationships crucial for understanding complex scientific layouts, such as the arrangement of equations and figures.

3. **Sequence Decoding to Markdown** Finally, a transformer-based decoder converts the contextual embeddings H into a sequence of tokens $y_1, y_2, \ldots, y_T$ that form the Markdown output. This step is trained with a standard cross-entropy loss:

$$\mathcal{L} = -\sum_{t=1}^{T} \log p_\theta(y_t \mid y_{<t}, H),$$

where $p_\theta(y_t \mid y_{<t}, H)$ is the probability of generating token y_t given the previously generated tokens and the encoder output. Importantly, the Markdown output supports embedded LaTeX, ensuring that mathematical expressions are preserved in their native format.

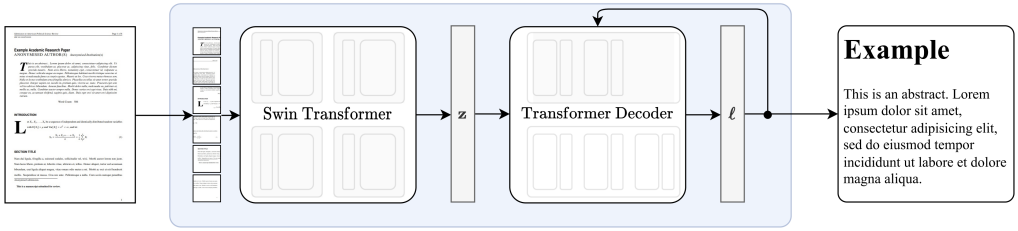


Figure 2.4: Architecture diagram of the Facebook Nougat model. The Swin Transformer encoder takes a document image and converts it into latent embeddings, which are subsequently converted to a sequence of tokens in a autoregressive manner.

2.3.3 Handling Mathematical Content

One of Nougat's key innovations is its ability to accurately extract and preserve mathematical formulas. Traditional OCR methods often fail to capture the spatial relationships in mathematical expressions [19]. Nougat overcomes this challenge by:

- Learning a joint representation of both text and layout, so that elements such as superscripts, subscripts, fractions, and matrices are recognized as distinct semantic units.
- Converting these units into LaTeX commands during decoding. For example, a fraction in the PDF is transformed into the LaTeX representation:

  $$\texttt{\textbackslash frac\{numerator\}\{denominator\}},$$

  ensuring that the semantic structure of the mathematical expression is retained.

Mathematically, if a region of the image corresponding to an equation is encoded as a feature vector $h \in \mathbb{R}^d$, the decoder learns to map h into a token sequence $\{y_t\}$ such that:

$$y_{1:T} = \text{Decoder}(h) \quad \approx \quad \texttt{\textbackslash frac\{a+b\}\{c\}},$$

where the predicted sequence faithfully represents the original formula.

2.3.4 Benefits for Scientific Documents

By converting PDFs into Markdown, Nougat preserves rich formatting and semantic structure [3]. This is particularly advantageous for scientific documents where:

- **Mathematical Equations:** Equations are preserved in LaTeX, ensuring that complex mathematical information remains accurate and editable.
- **Complex Layouts:** Multi-column formats, figures, and tables are handled more gracefully, maintaining the logical organization of the document.
- **OCR Robustness:** The integrated approach that combines visual feature extraction with transformer-based sequence modeling allows Nougat to extract text from both digital-born PDFs and scanned documents.

2.3.5 Workflow for Indexing Extracted Documents

Once the PDF content is extracted and converted into Markdown, the resulting text is segmented into meaningful chunks—such as paragraphs or sections—using techniques such as semantic chunking or fixed-size windowing. These chunks are then embedded using a suitable text encoder to obtain dense vector representations. Finally, the embeddings are stored in a vector database (e.g., FAISS or Chroma) which supports efficient similarity search.

The overall extraction and indexing pipeline can be summarized as follows:

1. **PDF Parsing:** Use tools like `pdfplumber` to extract raw text, or employ Facebook Nougat to convert the document into a structured Markdown format.
2. **Preprocessing:** Clean and segment the extracted text into logical chunks, preserving mathematical expressions and layout cues.
3. **Embedding:** Convert each text chunk into a vector using a pretrained text encoder.

4. **Indexing:** Insert the resulting embeddings into a vector database for efficient retrieval.

Chapter 3

Data Collection and Preprocessing

This chapter details the methodology used for data acquisition, document conversion, and the preparation of an instruction tuning dataset. The approach is designed to ensure that both the content and the structure of the source documents—particularly scientific papers containing complex layouts, mathematical equations, figures, and tables—are preserved. Figure 3.1 illustrates the complete data processing pipeline, from document acquisition to indexing and preparation of the instruction tuning dataset.

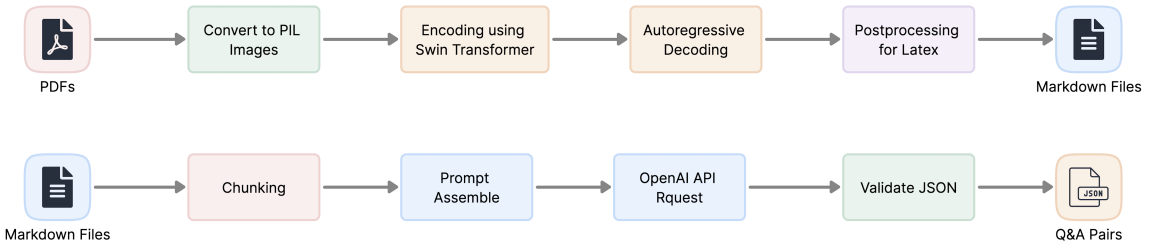


Figure 3.1: Data Flow Diagram for Finetuning

3.1 Data Acquisition

The dataset was curated by collecting **PDF** files from certified repositories such as IEEE Transactions and other academic databases. The acquisition process was automated using a keyword-based web crawling script, which ensured that documents relevant to the target tracking domain were systematically gathered. Table 3.1 shows the principal keywords used in the search process along with their associated statistics, generating a diverse dataset that spans multiple facets of the target tracking domain.

Keywords	**Count**
Classification	5
Estimation Theory	36
Fault Diagnosis	6
Information Fusion	6
Multisensor-Multitarget Tracking	60
Non-linear Filtering	14
Radar/Sonar/Image/Signal Processing	64
Sensor Resource Management	18
Total	209

Table 3.1: Number of papers collected per keyword category.

3.2 Document Conversion to Markdown

To convert these PDFs into a structured Markdown format suitable for indexing and retrieval, we implemented a multi-step extraction pipeline. The overall approach is as follows:

1. **PDF Loading and Page Segmentation:** PDFs are loaded using the `PyMuPDF` library, which efficiently parses the document and splits it into individual pages.

2. **Image Conversion:** Each extracted page is converted into a PIL image. Standardizing the image format ensures that all pages have consistent resolution and color properties before further processing.

3. **Encoding with Swin Transformer:** The PIL image representing a page is fed into a Markdown conversion processor. Within this processor, the image is first passed through a Swin Transformer encoder, which converts the visual content into latent embeddings. These embeddings capture the essential visual and layout information from the page.

4. **Autoregressive Decoding:** The latent embeddings are then passed to a transformer decoder that converts them into a sequence of tokens in an autoregressive manner. The output is a Markdown-formatted text that preserves the structure and key elements of the original PDF, including headings, paragraphs, and embedded LaTeX (for mathematical expressions).

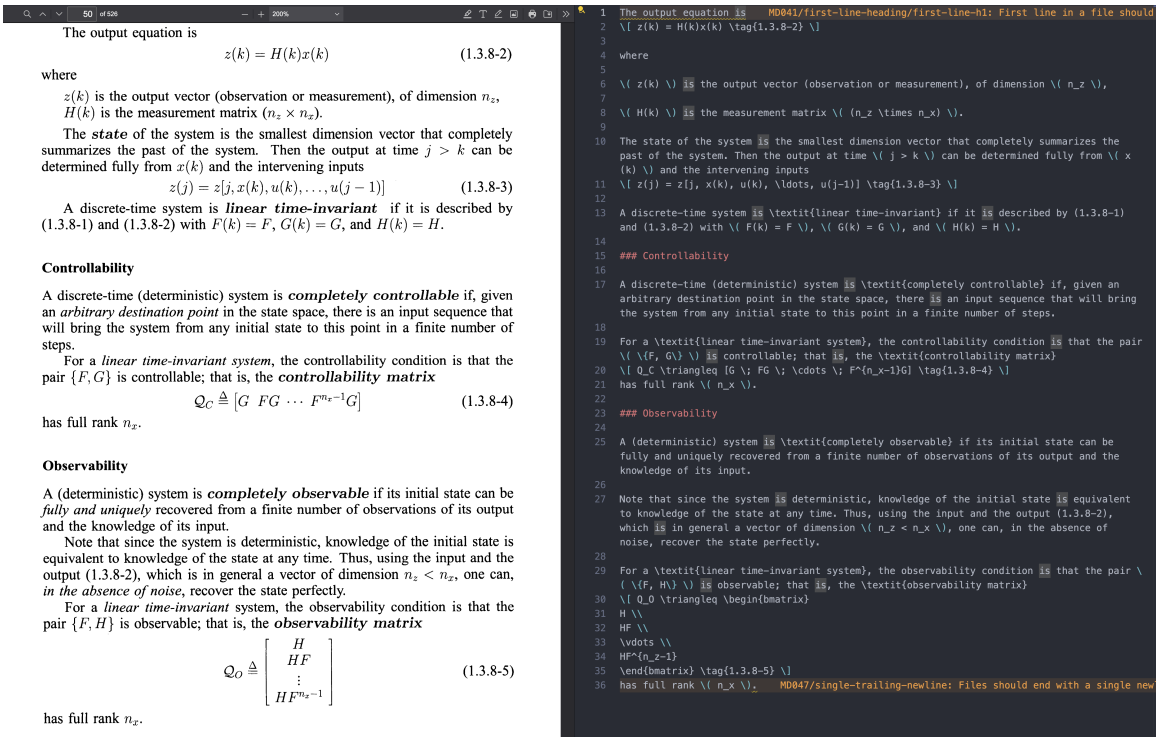
The following table summarizes the key parameters used in our document conversion processor:

Parameter	**Value**
PDF Parsing Library	PyMuPDF
Image Resolution (DPI)	96 DPI
Image Format	PIL Image
Maximum Input Length	3584
Model Name	Facebook Nougat Base
Encoder Architecture	Swin Transformer
Decoder Architecture	Transformer (Autoregressive)
Output Format	Markdown with embedded LaTeX

Table 3.2: Key parameters of the document conversion processor.

This extraction pipeline enables us to convert each PDF page into a high-quality

Markdown representation as illustrated in Figure 3.2, preserving the document's layout and critical content. The resulting Markdown files are then ready for segmentation, embedding, and indexing in our retrieval-augmented generation system.



The output equation is

$$z(k) = H(k)x(k) \tag{1.3.8-2}$$

where

$z(k)$ is the output vector (observation or measurement), of dimension n_z,
$H(k)$ is the measurement matrix ($n_z \times n_x$).

The ***state*** of the system is the smallest dimension vector that completely summarizes the past of the system. Then the output at time $j > k$ can be determined fully from $x(k)$ and the intervening inputs

$$z(j) = z[j, x(k), u(k), \ldots, u(j-1)] \tag{1.3.8-3}$$

A discrete-time system is ***linear time-invariant*** if it is described by (1.3.8-1) and (1.3.8-2) with $F(k) = F$, $G(k) = G$, and $H(k) = H$.

Controllability

A discrete-time (deterministic) system is ***completely controllable*** if, given an *arbitrary destination point* in the state space, there is an input sequence that will bring the system from any initial state to this point in a finite number of steps.

For a *linear time-invariant system*, the controllability condition is that the pair $\{F, G\}$ is controllable; that is, the ***controllability matrix***

$$\mathcal{Q}_C \triangleq \begin{bmatrix} G & FG & \cdots & F^{n_x-1}G \end{bmatrix} \tag{1.3.8-4}$$

has full rank n_x.

Observability

A (deterministic) system is ***completely observable*** if its initial state can be *fully and uniquely* recovered from a finite number of observations of its output and the knowledge of its input.

Note that since the system is deterministic, knowledge of the initial state is equivalent to knowledge of the state at any time. Thus, using the input and the output (1.3.8-2), which is in general a vector of dimension $n_z < n_x$, one can, *in the absence of noise*, recover the state perfectly.

For a *linear time-invariant* system, the observability condition is that the pair $\{F, H\}$ is observable; that is, the ***observability matrix***

$$\mathcal{Q}_O \triangleq \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{n_z-1} \end{bmatrix} \tag{1.3.8-5}$$

has full rank n_x.

```
The output equation is    MD041/first-line-heading/first-line-h1: First line in a file should
\[ z(k) = H(k)x(k) \tag{1.3.8-2} \]

where

\( z(k) \) is the output vector (observation or measurement), of dimension \( n_z \),

\( H(k) \) is the measurement matrix \( (n_z \times n_x) \).

The state of the system is the smallest dimension vector that completely summarizes the
past of the system. Then the output at time \( j > k \) can be determined fully from \( x
(k) \) and the intervening inputs
\[ z(j) = z[j, x(k), u(k), \ldots, u(j-1)] \tag{1.3.8-3} \]

A discrete-time system is \textit{linear time-invariant} if it is described by (1.3.8-1)
and (1.3.8-2) with \( F(k) = F \), \( G(k) = G \), and \( H(k) = H \).

### Controllability

A discrete-time (deterministic) system is \textit{completely controllable} if, given an
arbitrary destination point in the state space, there is an input sequence that will bring
the system from any initial state to this point in a finite number of steps.

For a \textit{linear time-invariant system}, the controllability condition is that the pair
\( \{F, G\} \) is controllable; that is, the \textit{controllability matrix}
\[ Q_C \triangleq [G \; FG \; \cdots \; F^{n_x-1}G] \tag{1.3.8-4} \]
has full rank \( n_x \).

### Observability

A (deterministic) system is \textit{completely observable} if its initial state can be
fully and uniquely recovered from a finite number of observations of its output and the
knowledge of its input.

Note that since the system is deterministic, knowledge of the initial state is equivalent
to knowledge of the state at any time. Thus, using the input and the output (1.3.8-2),
which is in general a vector of dimension \( n_z < n_x \), one can, in the absence of
noise, recover the state perfectly.

For a \textit{linear time-invariant system}, the observability condition is that the pair \
( \{F, H\} \) is observable; that is, the \textit{observability matrix}
\[ Q_O \triangleq \begin{bmatrix}
H \\
HF \\
\vdots \\
HF^{n_z-1}
\end{bmatrix} \tag{1.3.8-5} \]
has full rank \( n_x \).    MD047/single-trailing-newline: Files should end with a single new
```

Figure 3.2: Original Page vs Converted Markdown format

3.2.1 Preparing the Instruction Tuning Dataset

After converting the PDFs into Markdown (see Section 3.2), we prepare an instruction tuning dataset to further fine-tune our model for domain-specific question answering. This dataset is designed to provide each training example with a technical question, the context from which the answer can be derived, and a detailed, chain-of-thought answer that explains the reasoning process.

The preparation process involves several key steps:

3.2.1.0.1 1. Chunking the Markdown Files. Each Markdown file is split into coherent sections using header markers (e.g., `#, ##, ###`) or double newlines. In this step, we ensure that:

- **Integrity of Equations:** Mathematical equations are preserved as complete blocks and are not split across chunks.
- **Overlap Between Chunks:** A small overlap is maintained between consecutive chunks to avoid loss of contextual information at the boundaries.

3.2.1.0.2 2. Q&A Generation via ChatGPT API. For each text chunk, we dynamically generate a prompt to instruct ChatGPT to produce 2–5 high-quality technical questions and detailed answers. The prompt is carefully crafted to focus on extracting technical concepts, methodologies, and, when applicable, mathematical derivations formatted in LaTeX. An example of the prompt is shown below:

```
You are an expert academic researcher analyzing research papers and
    creating high-quality questions and answers.
This content is from the paper: "{paper_title}"


CONTENT CONTEXT:
- This is chunk {chunk_data['chunk_id'] + 1} of
    {chunk_data['total_chunks']}
- {'This section contains mathematical equations' if has_equations else
    'This section is primarily textual'}


TASK:
```

```
Generate 2-5 high-quality technical questions and detailed answers based
    on this content.


REQUIREMENTS FOR QUESTIONS:
1. Technical Focus:
   - Mathematical concepts and equations if present
   - Methodologies and algorithms
   - Theoretical frameworks
   - System architectures
   - Key findings and results


2. Question Types:
   - Technical understanding questions
   - Mathematical derivation questions (if equations present)
   - Process explanation questions
   - Comparative analysis questions
   - Implementation questions where relevant


REQUIREMENTS FOR ANSWERS:
1. Mathematical Precision:
   - Use LaTeX for ALL equations: $...$ for inline, $$...$$ for display
   - Double-escape all backslashes (e.g., `\\`) for JSON validity
   - Keep equations exactly as presented in the text
   - Show step-by-step derivations where applicable


2. Structure and Format:
```

```
   - Start with key concept or main point
   - Break down complex processes into steps
   - Use markdown formatting for clarity
   - Include relevant citations from the text

3. Technical Accuracy:
   - Use precise technical terminology
   - Include all relevant parameters and variables
   - Explain mathematical notation when used
   - Acknowledge any limitations or missing information

CONTENT TO ANALYZE:
{content}

IMPORTANT: YOU MUST RETURN YOUR RESPONSE IN THE FOLLOWING JSON FORMAT:
‘‘‘json
[
    {{
        "question": "Technical/mathematical question focusing on specific
            concept?",
        "answer": "Comprehensive answer with clear introduction, LaTeX
            equations, step-by-step explanation, citations, and technical
            precision"
    }},
    {{
        "question": "Another technical question?",
```

```
        "answer": "Another comprehensive answer..."
    }}
]
```

```
REMEMBER:
- Output must be valid JSON
- Escape backslashes properly for LaTeX equations
- Stay strictly within the provided content
- Preserve all mathematical notation exactly
- Use precise technical language
- Include step-by-step explanations
- Cite specific parts of the text
- Do not include any text outside the JSON array
```

This prompt is sent to the ChatGPT API (e.g., using GPT-4), and the returned JSON is post-processed to correct any formatting issues and to attach metadata such as the paper title, file name, chunk ID, total number of chunks, token count, and whether the chunk contains mathematical equations.

3.2.1.0.3 3. Constructing the Training Examples. Each training example is designed to include:

- A **Question** derived from the content.
- A **Context** that consists of the original text chunk along with additional relevant chunks (acting as supplementary information). In some instances, the

primary chunk is combined with a few distractor chunks (which improves the model's robustness to irrelevant information and enhances the model's ability to discern and disregard irrelevant content, leading to better performance in RAG tasks) to simulate realistic retrieval scenarios.

- A detailed **Answer** that includes step-by-step reasoning, with any mathematical content rendered in LaTeX.

The final instruction tuning dataset is stored in a JSON file, where each entry adheres to a standardized format. For clarity, the dataset can be visualized as in Table 3.3.

Question	**Context**	**Answer**
What is the role of the Kalman filter in target tracking?	[Extracted text chunk from documents, containing discussion on prediction models and noise filtering.]	The Kalman filter is used to optimally estimate the state of a dynamic system in the presence of noise. It works by predicting the state, updating the prediction with new measurements, and computing a weighted average based on uncertainties. The model is given by: $\boldsymbol{x}_t = F_t \boldsymbol{x}_{t-1} + \boldsymbol{w}_t$ where $\boldsymbol{w}_t$ is assumed to be zero-mean Gaussian noise with covariance $\boldsymbol{Q}$.

Table 3.3: Example final dataset format for instruction tuning. Each entry comprises a technical question, its context, and a detailed answer.

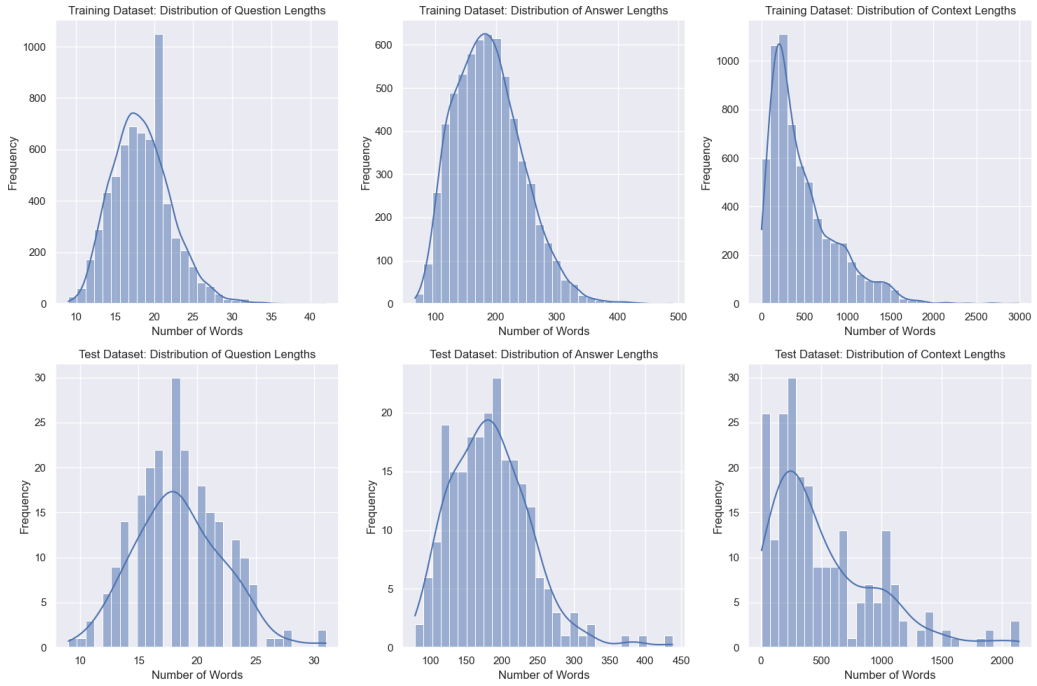


Figure 3.3: Statistics of the Dataset: Distributions of question lengths, answer lengths, and context lengths.

3.2.2 Dataset Statistics

Figure 3.3 provides a quantitative overview of our training dataset by illustrating the length distributions for questions, answers, and contextual information. This breakdown helps us understand the general structure of the data before fine-tuning the LLM. Specifically:

- **Question Lengths:** The histogram reveals how the number of words in each question spans from brief factual queries to more extensive, multi-sentence prompts. Such variation is crucial for ensuring the model can handle both concise and elaborate question types.

- **Answer Lengths:** The distribution of answers highlights the range of expected

responses. Some queries require short, direct replies, while others involve detailed explanations—an essential characteristic for building models that can adapt to varied informational needs.

- **Context Lengths:** Contextual passages can significantly impact the model's ability to reason effectively, especially for domain-specific questions. The observed spread in context lengths underscores the importance of preparing the LLM for scenarios with both minimal and substantial background information.

Chapter 4

Model Fine-Tuning

In this chapter, we describe the experimental setup and hyperparameter configurations used to fine-tune and evaluate our retrieval-augmented model. We detail the hyperparameter settings, hardware specifications, dataset splits, and present representative training loss plots. In addition, we justify our choice of using LoRA-based fine-tuning over alternative methods such as PPO and DPO, and provide a sample prompt used for fine-tuning.

4.1 Base Model (Llama 3 8B)

The core of our fine-tuning work is built on Meta AI's Llama 3 8B model—a state-of-the-art, open-source large language model released on April 18, 2024. Llama 3 8B is a decoder-only transformer with 8 billion parameters, designed to balance robust performance with resource efficiency. This model has been pre-trained on an extensive corpus of over 15 trillion tokens, which significantly enhances its language understanding and generation capabilities.

4.1.0.0.1 Model Selection Justification: We selected Llama 3 8B as our base model for fine-tuning on mathematical domain-specific tasks due to its well-balanced architecture and robust performance in handling complex language tasks, including the understanding and generation of LaTeX-formatted content. Although models such as Mistral, Qwen, Gemma, and even Mathstral have been proposed—with Mathstral being specifically designed for mathematical content—Llama 3 8B offers several practical advantages for our target use case. First, Llama 3 8B has been extensively validated in a variety of settings and is open-sourced under an accessible license, which facilitates both reproducibility and community-driven enhancements. Second, its decoder-only architecture has proven effective in generating coherent, step-by-step answers and maintaining logical reasoning, which is critical for tasks that require detailed LaTeX explanations and mathematical derivations. Moreover, its pre-training on a massive and diverse corpus ensures that it possesses a strong general language understanding, while its moderate size (8B parameters) makes it more amenable to parameter-efficient fine-tuning techniques like LoRA. In contrast, while Mathstral is tailored for mathematical processing, it often trades off general language capabilities and may be more resource-intensive to fine-tune. Thus, Llama 3 8B strikes the ideal balance between versatility, efficiency, and the capacity to generate detailed, mathematically rigorous responses, making it the optimal choice for our domain-specific fine-tuning efforts. The Figure 4.1 compares the performance of the 8B, 70B, and 405B versions of Llama 3 with that of competing models.

4.1.0.0.2 Key Features and Capabilities:

- **Optimized Transformer Architecture:** Llama 3 8B utilizes an advanced transformer architecture enhanced with Grouped-Query Attention (GQA), which

Category	Benchmark	Llama 3 8B	Gemma 2 9B	Mistral 7B	Llama 3 70B	Mixtral 8x22B	GPT 3.5 Turbo	Llama 3 405B	Nemotron 4 340B	GPT-4 (0125)	GPT-4o	Claude 3.5 Sonnet
General	MMLU (5-shot)	69.4	**72.3**	61.1	**83.6**	76.9	70.7	87.3	82.6	85.1	89.1	**89.9**
	MMLU (0-shot, CoT)	**73.0**	72.3△	60.5	**86.0**	79.9	69.8	88.6	78.7◁	85.4	**88.7**	88.3
	MMLU-Pro (5-shot, CoT)	**48.3**	–	36.9	**66.4**	56.3	49.2	73.3	62.7	64.8	74.0	**77.0**
	IFEval	**80.4**	73.6	57.6	**87.5**	72.7	69.9	**88.6**	85.1	84.3	85.6	88.0
Code	HumanEval (0-shot)	**72.6**	54.3	40.2	**80.5**	75.6	68.0	89.0	73.2	86.6	90.2	**92.0**
	MBPP EvalPlus (0-shot)	**72.8**	71.7	49.5	**86.0**	78.6	82.0	88.6	72.8	83.6	87.8	**90.5**
Math	GSM8K (8-shot, CoT)	**84.5**	76.7	53.2	**95.1**	88.2	81.6	**96.8**	92.3◇	94.2	96.1	96.4◇
	MATH (0-shot, CoT)	**51.9**	44.3	13.0	**68.0**	54.1	43.1	73.8	41.1	64.5	**76.6**	71.1
Reasoning	ARC Challenge (0-shot)	83.4	**87.6**	74.2	**94.8**	88.7	83.7	**96.9**	94.6	96.4	96.7	96.7
	GPQA (0-shot, CoT)	32.8	–	28.8	**46.7**	33.3	30.8	51.1	–	41.4	53.6	**59.4**
Tool use	BFCL	**76.1**	–	60.4	84.8	–	**85.9**	88.5	86.5	88.3	80.5	**90.2**
	Nexus	**38.5**	30.0	24.7	**56.7**	48.5	37.2	**58.7**	–	50.3	56.1	45.7
Long context	ZeroSCROLLS/QuALITY	81.0	–	–	90.5	–	–	**95.2**	–	**95.2**	90.5	90.5
	InfiniteBench/En.MC	65.1	–	–	78.2	–	–	**83.4**	–	72.1	82.5	–
	NIH/Multi-needle	98.8	–	–	97.5	–	–	98.1	–	**100.0**	**100.0**	90.8
Multilingual	MGSM (0-shot, CoT)	**68.9**	53.2	29.9	**86.9**	71.1	51.4	**91.6**	–	85.9	90.5	**91.6**

Figure 4.1: Comparative Performance of Meta Llama 8B, 70B, and 405B Models Against Competing Approaches

improves inference scalability and efficiency. This design enables the model to handle long contexts—up to 8,192 tokens—facilitating applications that require processing of extended documents or multi-turn conversations.

- **Enhanced Performance:** Despite its relatively modest size compared to larger counterparts, Llama 3 8B has demonstrated competitive performance on a range of benchmarks. Evaluations indicate that it outperforms previous generations such as Llama 2 (even in its 70B variant) in key areas like instruction following, reasoning, and code generation, while maintaining lower latency and faster inference speeds.

- **Accessibility and Open-Source Advantage:** Llama 3 8B is released under a custom Meta Llama 3 Community License, making it freely available for academic research and commercial experimentation. Its open-source nature encourages innovation, allowing developers to fine-tune and integrate the model

into diverse applications without the prohibitive costs typically associated with large proprietary models.

- **Versatile Use Cases:** Owing to its robust pre-training and fine-tuning readiness, Llama 3 8B is well-suited for a variety of tasks such as customer support chatbots, content generation, summarization, and even code generation. Its efficient architecture also makes it an attractive choice for deployment on edge devices and environments with limited computational resources.

4.1.0.0.3 Integration into Our Fine-Tuning Pipeline:

For our project, Llama 3 8B serves as the base model that is further fine-tuned using Low-Rank Adaptation (LoRA). This parameter-efficient fine-tuning method allows us to adapt the model to the specific nuances of the target tracking domain while preserving the extensive knowledge captured during pre-training. The fine-tuned model is then evaluated on a curated test dataset, ensuring that domain-specific performance improvements are realized.

In summary, Llama 3 8B offers a powerful combination of high-quality language understanding, efficient inference, and open-source accessibility. These features make it an ideal candidate for further fine-tuning and application in specialized domains.

4.2 Fine-Tuning

This section describes the fine-tuning process for the Llama 3 8B model using LoRA. The subsections below detail the hyperparameter settings, hardware requirements, dataset splits, the prompt used for fine-tuning, and the training process with loss analysis.

4.2.1 Hyperparameter Settings

Our fine-tuning experiments build upon the Llama 3 8B base model using Low-Rank Adaptation (LoRA) for efficient adaptation. LoRA was chosen because it freezes the original model weights and fine-tunes only a small set of additional parameters, leading to resource efficiency, improved stability, and deployment simplicity. Table 4.1 summarizes the key hyperparameters used in our experiments.

Hyperparameter	**Value/Description**
Base Model	Llama 3 8B (decoder-only Transformer)
Fine-Tuning Technique	LoRA (Low-Rank Adaptation)
LoRA Rank (r)	8
Maximum Sequence Length	4096 tokens
Batch Size	192 samples (effective batch size across GPUs)
Learning Rate	5×10^{-5} (initial), decayed by 0.9996 every 15 updates
Number of Training Epochs	3 epochs
Optimizer	AdamW [15]
Weight Decay	0.1
Dropout Rate	0.1
Warmup Steps	1000 steps

Table 4.1: Key hyperparameters used for fine-tuning the Llama 3 8B model with LoRA.

4.2.2 Hardware Requirements

The experiments were conducted on a high-performance computing cluster. Table 4.2 summarizes the hardware configuration used during fine-tuning.

4.2.3 Dataset Splits

The fine-tuning data was curated from domain-specific research papers, converted from PDF to Markdown. The dataset was split into three subsets to ensure robust

Component	Specification
GPUs	4 NVIDIA A100 40GB GPUs
CPU	Dual Intel Xeon Gold 6248R (2.6 GHz)
Memory	512 GB DDR4 RAM
Storage	NVMe SSD with 4 TB capacity

Table 4.2: Hardware configuration used for fine-tuning the model.

evaluation:

- **Training Set:** 80% of the collected Markdown chunks.
- **Validation Set:** 10% of the dataset, used for hyperparameter tuning and early stopping.
- **Test Set:** 10% of the dataset, independently curated and not used during fine-tuning to assess domain-specific performance.

Stratified sampling was applied to maintain a balanced representation of documents containing both textual and mathematical content.

4.2.4 Prompt for Fine-Tuning

Below is an example prompt used during fine-tuning, directing the model to produce detailed, step-by-step, and well-formatted responses. Incorporating Chain-of-Thought reasoning guides the model to generate richer explanatory chains grounded in the original context, boosting accuracy and preventing overfitting.

```
You are an expert academic assistant specialized in analyzing research
    papers and providing detailed, technical answers.

Please follow the instructions below to generate your answer.
```

```
Instructions:
  1. Analyze the Question: Identify key requirements, formulas, and
     research parameters.
  2. Review the History: Check for connections in the history and
     incorporate relevant context into the answer.
  3. Evaluate Sources: Focus on relevant sections of provided documents,
     verify equations, and ensure accuracy.
  4. Structure the Response: Use clear academic formatting with
     paragraphs, notations, lists, and bullet points.

Provide rigorous, complete, and well-formatted responses using available
   documents or scholarly knowledge. For general questions, use your own
   expertise.

Retrieved Context:
------------------

QUESTION:
```

4.3 Results

Table 4.3 summarizes the key metrics obtained from our fine-tuned LLM during both training and evaluation. These metrics—ranging from training loss, runtime, and FLOPs to evaluation loss and perplexity—provide a holistic view of the model's

learning process and overall performance.

Table 4.3: Fine-Tuned LLM Metrics for Training and Evaluation

Metric	Training	Evaluation
Epoch	6.5223	6.5223
Total FLOPs	1.91×10^{17}	–
Loss	1.1435	1.7047
Runtime (s)	2240.0777	18.544
Samples/sec	0.561	1.51
Steps/sec	0.014	1.51
Perplexity	–	5.4998

During training, the model achieved a final loss of 1.14, converging over approximately 2,240 seconds. For evaluation, the model demonstrated a validation loss of 1.70 and a corresponding perplexity of 5.50, reflecting a reasonably strong grasp of the domain-specific content it was fine-tuned on. These results indicate that the model successfully balances computational efficiency with accuracy, capturing both contextual and mathematical nuances essential for the target domain.

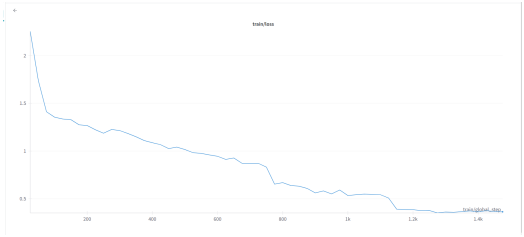
In addition to loss and perplexity metrics, we evaluated the model's problem-solving capabilities on the GSM8k benchmark, a widely used dataset for assessing mathematical reasoning performance. Table 4.4 summarizes the results in comparison with several prominent open-source and proprietary models.

Figure 4.2 further illuminates the training dynamics through three critical plots:

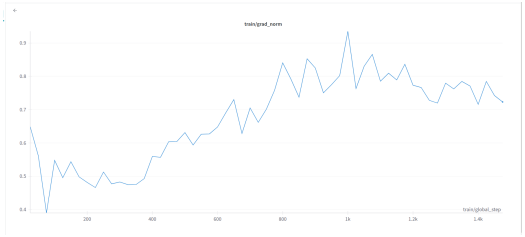
1. **Loss Over Steps** – Depicts how the training loss decreases as the number of steps increases, reflecting the model's progression toward convergence.

2. **Gradient Norm** – Provides insights into the stability of training by tracking the magnitude of gradients, which can indicate potential issues like vanishing or exploding gradients.

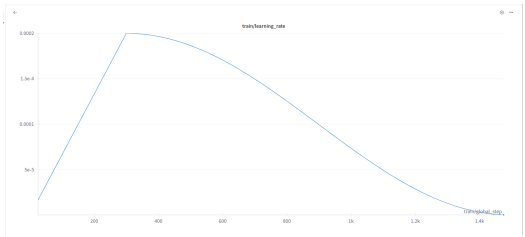
3. **Learning Rate** – Illustrates the learning rate schedule or adjustments made during training, underscoring its impact on convergence speed and final model performance.



(a) Loss Over Steps



(b) Gradient Norm



(c) Learning Rate

Figure 4.2: Training Diagnostics: (a) Loss over Steps, (b) Gradient Norm, and (c) Learning Rate.

Table 4.4: GSM8k Evaluation

Model	**Accuracy (%)**
GPT-4o (OpenAI)	94.8
Mixtral 8x7B	58.4
Llama 3 8B (Base)	34.3
SciRAG (Ours)	70

While the GSM8k score for the SciRAG model is modest relative to larger, general-purpose models, it is important to note that SciRAG was specifically optimized for structured scientific documents and targeted domains like target tracking, rather than

open-domain mathematical reasoning. As such, the performance is consistent with the model's intended specialization and scope.

By closely monitoring these metrics and plots, practitioners can fine-tune hyperparameters, detect instabilities early, and optimize training schedules. This methodological transparency facilitates reproducibility and paves the way for future improvements in both model performance and computational efficiency.

4.4 Discussion

4.4.1 Training Dynamics

In fine-tuning a Large Language Model (LLM) for a specialized domain, it is essential not only to observe the usual signs of healthy training (e.g., a decreasing loss) but also to confirm that the model is actually internalizing domain-relevant knowledge. As shown in Figure 4.2, three primary plots—loss, gradient norm, and learning rate—provide insight into how the model is adapting its parameters to handle the complexities of our target domain. The key findings are summarized below, along with their relevance to domain-specific fine-tuning:

4.4.1.0.1 Continuously Decreasing Loss. The model's loss decreases steadily throughout training, suggesting that it is successfully learning to map domain inputs (e.g., specialized queries or scientific notations) to appropriate outputs. For general-purpose LLMs, convergence may sometimes plateau due to vocabulary mismatches or insufficient exposure to domain-specific structures. Here, however, the consistently declining loss indicates the model is effectively narrowing the knowledge gap

between general language usage and the specialized requirements of the field. This reduction in loss directly translates into more accurate responses, better command of discipline-specific terminology, and fewer mistakes in advanced components such as mathematical expressions or intricate logical reasoning.

4.4.1.0.2 Increasing and Fluctuating Gradient Norm. Although the loss is decreasing, the gradient norm exhibits a slight upward trend with occasional spikes. This behavior often results from navigating complex regions of the parameter space where domain-specific concepts sharply diverge from standard English text or general knowledge. In such scenarios, larger or more volatile updates can occur because:

- **Specialized Token Usage:** Domain-specific tokens (technical terms, math symbols) might be relatively infrequent in a general pretraining corpus, so their embeddings undergo larger changes when fine-tuning on new, targeted data.
- **Momentum and Warm-Up Schedules:** Momentum-based optimizers accumulate updates, and a learning rate warm-up may briefly amplify the gradient magnitude, particularly if the model encounters nuanced patterns that were underrepresented in the original training.
- **Higher Loss Landscape Curvature:** Introducing domain-specific constraints or specialized equations can lead to steeper regions of the loss surface, so gradients can become more pronounced.

Crucially, this trend can be beneficial when it reflects the model's deeper engagement with domain syntax and semantics. As long as the loss does not diverge, occasional fluctuations in the gradient norm may indicate that the model is refining internal representations to better capture domain peculiarities.

4.4.1.0.3 Warm-Up and Decay in Learning Rate. The learning rate schedule starts low, gradually increases ("warm-up"), then decreases. This choice is particularly effective for domain-specific fine-tuning:

- **Stabilizing Early Stages:** A lower initial learning rate prevents erratic updates that could destroy useful general language patterns already captured in the pre-trained model. This safeguards the foundational language understanding while letting the model adapt to domain nuances.
- **Efficient Exploration of Specialized Knowledge:** The brief warm-up period allows for slightly more aggressive parameter adjustments once the model stabilizes, accelerating the uptake of specialized terms and reasoning.
- **Fine-Grained Convergence:** A decaying learning rate later in training ensures that crucial domain details—such as advanced formula rendering or interpretive steps in target tracking—are refined without overshooting or introducing noise. This final "polishing" can be the difference between good and exceptional performance in specialized tasks.

4.4.2 Output Quality

The primary objective of our work was to refine a Large Language Model (LLM) to address two key challenges: accurate rendering of scientific equations and enhancing domain relevance—specifically for target tracking applications. Figure 4.3 highlights how the fine-tuned LLM surpasses the base model in terms of both the quantity and quality of generated outputs. While the base model provides satisfactory responses in broad contexts, it often struggles with complex scientific notation and domain-specific

reasoning, leading to incomplete explanations or misinterpretations of target-tracking concepts. In contrast, the fine-tuned LLM demonstrates significantly improved performance in these areas.

Figure 4.3: Output Quality Comparison (I): Fine-Tuned LLM vs. Base LLM

4.4.3 Enhanced Mathematical Rendering.

A salient improvement lies in the fine-tuned LLM's ability to flawlessly render LaTeX equations. By integrating domain-specific data and refining the model to handle the intricacies of mathematical expressions, we have minimized errors such as incorrect

Figure 4.4: Output Quality Comparison (II): Fine-Tuned LLM vs. Base LLM

symbols, missing brackets, and misaligned fractions. This fidelity in mathematical rendering is crucial for fields where even minor typographical inconsistencies can alter the meaning of an equation or mislead subsequent analysis.

4.4.4 Domain-Specific Relevance.

Beyond mere formatting, our model's specialized training in the target tracking domain enables it to generate highly relevant explanations that tie theoretical equations directly to real-world tracking scenarios. This deeper contextual understanding lends itself to more cohesive narratives and step-by-step derivations that are not only mathematically correct but also provide richer insights for practitioners. In turn, end-users

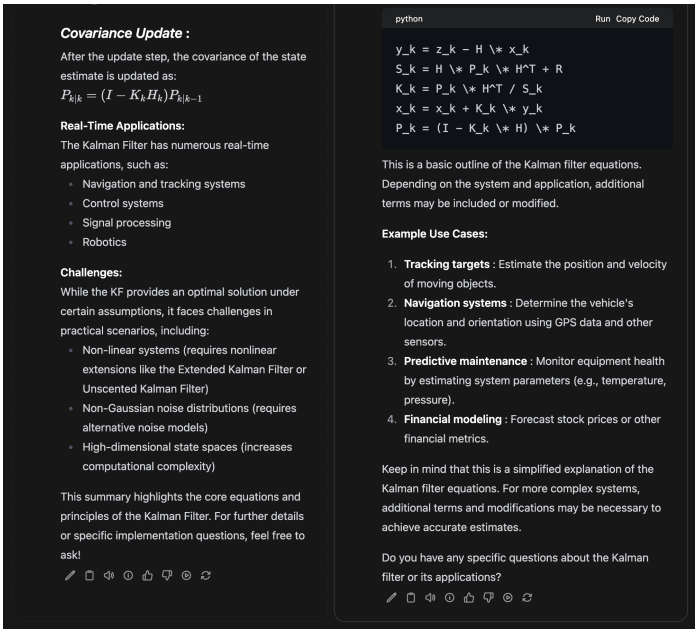


Figure 4.5: Output Quality Comparison (III): Fine-Tuned LLM vs. Base LLM

can rely on the model for precise recommendations, improved conceptual clarity, and well-grounded methodologies in surveillance, security, and robotics applications.

4.4.5 Practical Implications and Worth.

The advancements demonstrated by the fine-tuned LLM have broad implications:

- **Technical Documentation:** Organizations that rely on detailed mathematical documentation—such as aerospace, defense contractors, or academic institutions—can benefit from generating automated reports and clarifications that are both accurate in notation and tailored to domain-specific requirements.

- **Educational Tools:** In academia, the model could serve as a robust teaching assistant capable of providing on-demand, high-quality explanations and derivations of equations related to tracking algorithms.

- **Research and Development Acceleration:** Enhanced clarity and precision in outputs allow researchers to focus on higher-level problem-solving instead of clarifying or rewriting inaccurate model-generated content. This efficiency can expedite prototyping and experimentation cycles in machine learning and systems engineering.

4.4.6 Future Work.

While our improvements are notable, there remain several directions worth exploring. One avenue is incorporating a broader set of real-world sensor models and environmental factors to improve the model's robustness when discussing advanced or uncommon tracking scenarios. Another area of interest is systematically integrating additional scientific domains—such as signal processing, optimization, and statistical inference—to expand the model's utility across interdisciplinary fields.

Overall, these findings underscore the importance of domain-specific fine-tuning for LLMs. By aligning the model's output format (equations) and content (target tracking) with specialist requirements, we demonstrate how targeted refinement can significantly elevate both the readability and applicability of an LLM's output. The remaining comparison examples, presented in Appendix **??**, further illustrate these improvements and support the value of our approach.

Chapter 5

RAG Chatbot Development

This chapter outlines the end-to-end development pipeline for a Retrieval-Augmented Generation (RAG) chatbot. The pipeline is divided into several key stages: data processing, embeddings creation, vector database management (including indexing and retrieval), LLM support, and finally interface and deployment. Each stage is critical in ensuring that the chatbot can efficiently process data, retrieve relevant context, and generate accurate and context-aware responses.

5.1 Data Processing

Data processing forms the foundation of the RAG chatbot and, more broadly, any retrieval-augmented generation system. In this stage, raw files are ingested, processed, and prepared for embedding extraction. Recent studies and industry guides emphasize that effective chunking is not merely a preprocessing step but a crucial determinant of retrieval accuracy and efficiency. The following subsections detail our approach, enriched by best practices reported in the literature and community forums.

5.1.1 File Ingestion and Supported File Types

Our pipeline supports multiple file formats:

- **PDF Files:** PDF documents are processed using our extraction algorithm (described in Section 3.2), which converts them into Markdown files. This conversion not only preserves the structural layout (including figures, equations, and section headers) but also facilitates subsequent text segmentation.
- **TEX Files:** TEX files, being primarily textual and structured, are directly ingested without conversion.

After ingestion, the raw files are loaded into the system, ensuring that the original document structures are maintained for further processing.

5.1.2 Text Chunking

To prepare the text for embedding, the processed content is divided into manageable chunks. Our approach leverages LangChain's Recursive Character Text Splitter, which is widely recognized for its robustness in handling heterogeneous document types. Key aspects of our chunking strategy include:

- **Chunk Size:** 1500 characters per chunk. This size was chosen based on experiments reported in the literature, balancing the need for rich context with the LLM's context window limitations.
- **Overlap:** 100 characters of overlap between consecutive chunks. Overlapping is critical for maintaining continuity, ensuring that important information (e.g., mathematical equations or transition phrases) is not lost at the boundaries.

- **Separator:** The period (‘.‘) is used as the primary separator. This deliberate choice helps preserve the integrity of complete sentences and equations, reducing the risk of breaking complex constructs mid-expression.

The recursive nature of the splitter allows it to attempt splits using a hierarchy of separators (from paragraph breaks to spaces) until the resulting chunks are within the desired size limit. This strategy is advantageous for technical and research texts where semantic coherence is paramount.

5.1.3 Cleaning and Preprocessing

After chunking, each text segment undergoes a cleaning process to ensure high-quality input for embedding generation:

- **Removing Extra White Spaces:** Redundant spaces and tabs are trimmed.
- **Eliminating Redundant New Lines:** Unnecessary line breaks are removed to maintain a consistent format.
- **Filtering Unwanted Characters:** Encoded or extraneous characters are eliminated from the text.

Notably, no text normalization (e.g., lowercasing or stemming) is applied since the original case is preserved to maintain semantic integrity—a common practice when preparing text for modern embedding models.

The processed and cleaned data is subsequently forwarded to the embedding creation stage, where each chunk is transformed into a numerical vector representation suitable for efficient retrieval from a vector database.

5.2 Embedding Model

Embeddings are numerical vector representations of information—whether text, documents, images, or audio—that capture the underlying semantic meaning. In our pipeline, embeddings are used to project processed text chunks into a high-dimensional vector space where semantically similar content is positioned close together. This property is crucial for efficient semantic search, clustering, and retrieval tasks in retrieval-augmented generation (RAG) systems.

Our implementation leverages a transformer-based embedding model available through the HuggingFace framework. The chosen model, `BAAI/bge-large-en-v1.5`, is particularly well-suited for our chatbot as it is designed to handle both Portuguese and English text. This model maps text into a dense 1024-dimensional vector space and is among the top performers on the Massive Text Embedding Benchmark (MTEB) Leaderboard [16].

A major advancement in the field of embedding models was achieved by Reimers and Gurevych (2019) through the development of Sentence-BERT. This approach modifies the original BERT architecture by integrating siamese and triplet network structures, enabling it to generate sentence embeddings that effectively capture nuanced semantic relationships. During training, the model is optimized to draw embeddings of semantically similar sentences closer together while distancing those of unrelated sentences. For example, the query "What is Kalman filter?" might be encoded into a 384-dimensional vector, such as

$$[0.45,\ 0.09,\ \ldots,\ 0.77],$$

which serves as a compact numerical representation of its meaning. These dense vectors allow us to use similarity measures like cosine similarity to accurately determine how closely related different pieces of text are in terms of their semantic content.

5.3 Vector Database

The vector database in our RAG pipeline is responsible for storing and managing embedding vectors produced from text chunks, thereby enabling rapid similarity search and efficient information retrieval. Our implementation leverages components from the `llama_index` framework, combined with custom retriever classes, to form a cohesive indexing and retrieval system. In our codebase, the vector database system is organized into two primary components: indexing and retrieval.

5.3.1 Indexing

Indexing is the process by which the embedding vectors are organized into a structure that facilitates fast retrieval. In our implementation, this involves the following steps:

- **Building the Vector Store:** Text chunks (produced in the data processing stage and transformed into embeddings) are indexed using a `VectorStoreIndex` object. This object encapsulates the embeddings and metadata, making them available for similarity search.
- **Embedding Integration:** The vector store is constructed by interfacing with our chosen embedding model (via a HuggingFace or similar class), ensuring that

each document's embedding is stored along with key metadata (such as source information, chunk boundaries, etc.).

- **Index Configuration and Updates:** Hyperparameters such as the number of similar results to retrieve (e.g., `similarity_top_k`) are configured via our settings class (`RAGSettings`). The index is updated as new documents are ingested or as the embedding model is refined.

Our codebase encapsulates these ideas within custom classes. For example, the `LocalRetriever` class includes methods to obtain a normal retriever (using the `VectorIndexRetriever`) that wraps the vector store, applying configured similarity thresholds and embedding models to facilitate fast nearest neighbor searches.

5.3.2 Retrieval

The retrieval component is designed to efficiently locate and return the most relevant text chunks based on a user query. The retrieval workflow in our system involves:

- **Query Embedding:** User queries are first transformed into embedding vectors using the same embedding model that was used for the documents. This ensures consistency between the query and document representations. Figure 5.1 show the comparison across different embedding models.

- **Similarity Search:** The vector index is queried using similarity metrics (e.g., cosine similarity) to compare the query vector with stored document embeddings. In our implementation, this is handled by a normal retriever (via `Vector Index Retriever`) or, when appropriate, by more sophisticated hybrid retrievers.

- **Hybrid and Two-Stage Retrieval:** For cases where queries may be ambiguous or require higher precision, we implement a two-stage retrieval strategy. This involves combining results from a BM25 retriever (which leverages traditional term-based similarity) and the vector index retriever, then fusing these results with a `QueryFusionRetriever` or the `TwoStageRetriever` class. In the two-stage approach, results are further refined using a reranking model (implemented via `SentenceTransformerRerank`) that postprocesses the candidate nodes to select the most semantically relevant chunks.

- **Ranking and Selection:** Finally, retrieved candidates are ranked according to their similarity scores, and the top k candidates are selected to form the context for the downstream generative model.

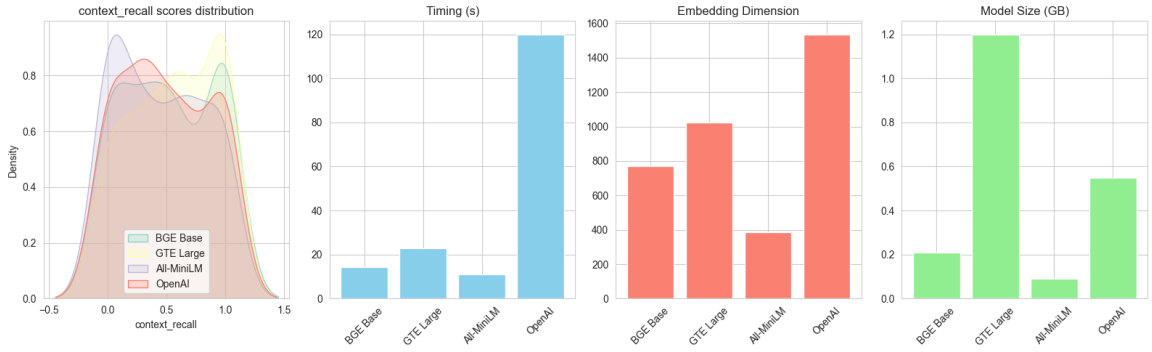


Figure 5.1: Comparison of Context Recall Across Embedding Models, Including Retrieval Latency, Embedding Dimensions, and Model Sizes.

In our code, the `LocalChatEngine` class integrates these components by first obtaining a retriever via the `LocalRetriever.get_retrievers` method. This method selects the appropriate retrieval strategy (normal, hybrid, or router-based) based on our system settings. The chosen retriever is then used to initialize a chat engine (e.g., `CondensePlusContextChatEngine`), which combines the retrieval results with

a language model and a memory buffer (via `ChatMemoryBuffer`) to manage context and conversation state.

Through this architecture, our vector database and retrieval system ensure that the generative model is always provided with the most accurate and contextually relevant information, which is critical for generating high-quality responses in our RAG application.

5.4 LLM Support

Large Language Models (LLMs) form the core of our generative pipeline, providing the capability to generate context-aware and coherent responses. Our system is designed to be model-agnostic and supports multiple types of LLMs, allowing us to choose the most appropriate model based on cost, performance, language support, and data privacy requirements. In our current implementation, we support the following categories of models:

- **OpenAI Models:** Models such as GPT-4 accessed via the OpenAI API. These models require an API key (which can be provided as an environment variable or explicitly) and offer robust performance with extensive language capabilities.
- **Ollama Models:** Models hosted via Ollama, which are accessed locally (or on a specified host) through an HTTP interface. They are configured using parameters such as `tfs_z`, `top_k`, and `context_window`.
- **HuggingFace Models:** Models from the HuggingFace ecosystem (e.g., those provided by Sentence Transformers) are run locally or on cloud resources. They

use tokenizers from the same model repository and support custom generation settings.

- **Custom Models:** These refer to models hosted outside our primary providers, such as models deployed with TensorRT or using llama.cpp. They are accessed via custom API endpoints and support local inference.

When a model is selected, our code dynamically instantiates the appropriate LLM class by checking the `model_type` parameter and passing the corresponding configuration parameters.

Table 5.1 summarizes the input types and key configuration aspects for each LLM type supported in our system.

LLM Type	**API/Endpoint**	**Key Configuration Parameters**
OpenAI	REST API	API key, model name, temperature
Ollama	Local HTTP Endpoint	Base URL, context window
HuggingFace	Local/Cloud via HF Hub	Model, context window
Custom	Custom API Endpoint	API base URL, custom settings

Table 5.1: Supported LLM types and their key configuration parameters.

5.5 Response Generation

Our response generation process is designed to ensure that each chat interaction yields an accurate, contextually grounded answer. The generation workflow is divided into three key steps:

1. **Standalone Question Formation:** The conversation history, along with the latest user message, is condensed into a single, self-contained question. This step

uses a dedicated prompt that reformulates the dialogue into a standalone query, ensuring that all necessary context is captured and ambiguities are removed.

2. **Context Retrieval:** The standalone question is then used to query our vector database. The retrieval component gathers the most relevant document chunks that provide supporting evidence and background information. These chunks are concatenated to form a comprehensive context for the question.

3. **Final Response Generation:** Finally, a second prompt is constructed that combines the retrieved context, the standalone question, and additional instructions. This prompt is sent to the LLM, which generates a response that is both coherent and well-grounded in the external knowledge. Optionally, fine-tuning may be applied to further tailor the LLM's output to our domain.

Sample Prompts

5.5.0.0.1 Step 1: Standalone Question Formation Below is an example prompt that condenses the chat history and the latest user message into a standalone question:

```
You are an expert conversational assistant. Given the following
   conversation history and the most recent user message,
please condense them into a clear, standalone question that captures all
   the necessary context.
------------------
Conversation History:
{conversation_history}
------------------
```

```
Follow-Up Question:

{question}

------------------

Standalone Question:
```

5.5.0.0.2 Step 2: Context Retrieval Once the standalone question is formed, it is used to retrieve relevant context from the vector database. The retrieved text is then combined to form the supporting context.

5.5.0.0.3 Step 3: Final Response Generation The final prompt integrates the retrieved context and the standalone question along with specific generation instructions:

```
You are a knowledgeable assistant with expertise in the relevant domain.
   Use the context provided below to answer the following question in a
   detailed and coherent manner.

------------------

Retrieved Context:

{retrieved_context}

------------------

Question:

{standalone_question}

------------------

Answer:
```

This structured, multi-step approach ensures that the LLM receives a well-formed query and all necessary supporting information, ultimately resulting in responses that

are both accurate and contextually relevant.

5.6 Interface and Deployment

The final stage of our system focuses on the user interface and deployment of the RAG chatbot. A well-designed interface not only makes the system accessible to end users but also ensures that performance and reliability are maintained in a production environment.

5.6.1 User Interface and Usage Workflow

The user interface of our application is built using `gradio`, an open-source Python framework designed for rapid development of machine learning and data science web applications. Figure 5.3 illustrates the main interface.

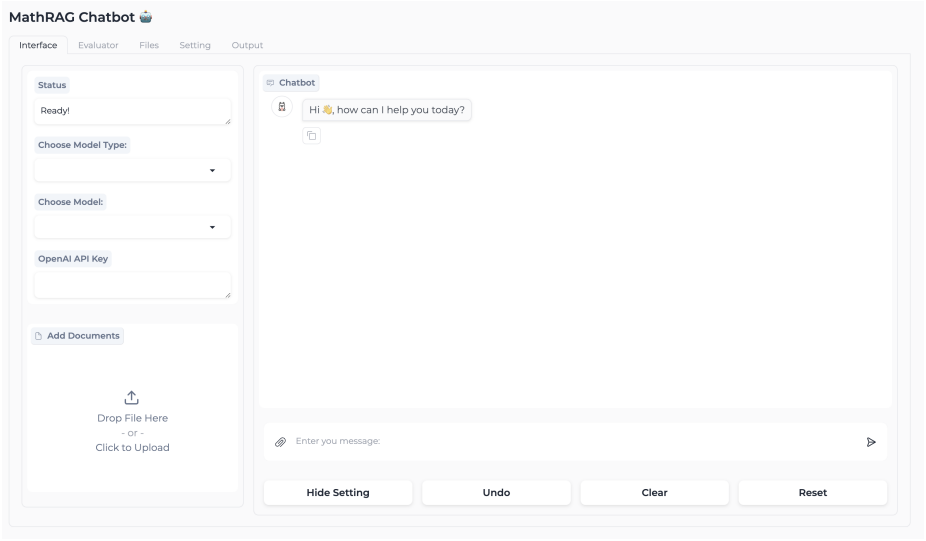


Figure 5.2: Gradio-based user interface for the RAG chatbot. The left sidebar provides a drag-and-drop area for uploading PDF documents, while the main panel displays the chat window.

To start using the application, the user follows these steps:

1. **Document Upload:** Upload PDF documents by dragging and dropping files into the left sidebar. A progress bar will indicate the upload and processing status.

2. **Model Selection:** Once the document processing is complete (i.e., when the progress bar disappears), the user selects the desired model from a drop-down menu. For instance, the Mistral-7B-Instruct-v0.2 model is accessed via the Hugging Face Hub.

3. **Chat Interaction:** With the model selected, the user can begin chatting. The interface supports prompt customization and displays a list of uploaded files in a dedicated "Files" tab.

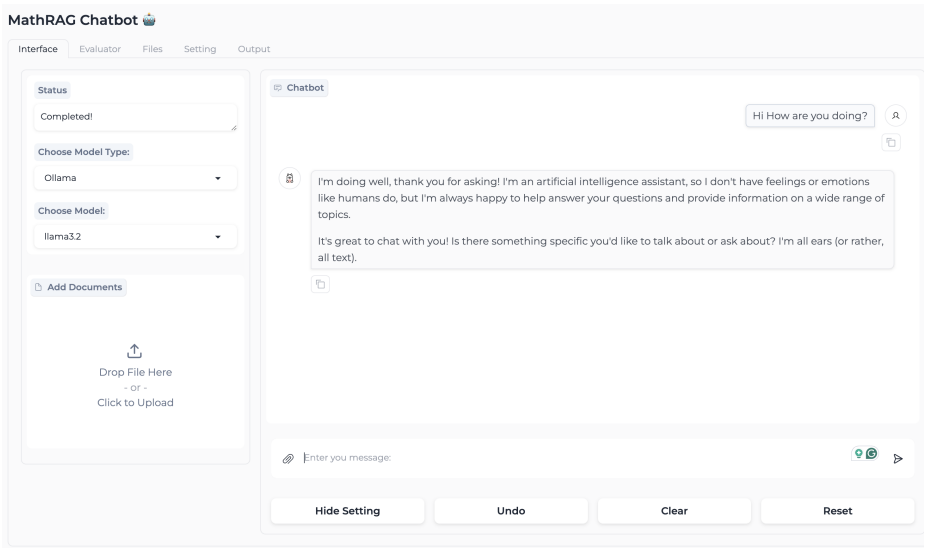


Figure 5.3: Gradio-based user interface for the RAG chatbot when the model selected. The left sidebar provides a dropdown for choosing the model type and model name, while the main panel displays the chat window.

5.6.2 Additional Features

The interface offers several advanced features to enhance the user experience and facilitate system evaluation:

- **Prompt Customization:** Users can modify the prompt used for response generation directly within the interface.
- **Files Tab:** A dedicated tab allows users to view and manage their uploaded documents.
- **LLM Evaluation:** The application supports benchmarking LLM performance in the RAG setup. Users can upload datasets in a specified format, and the system will run benchmarks, returning performance results and plots for further analysis. Note: The user must upload the appropriate documents to the application prior to initiating the benchmark.

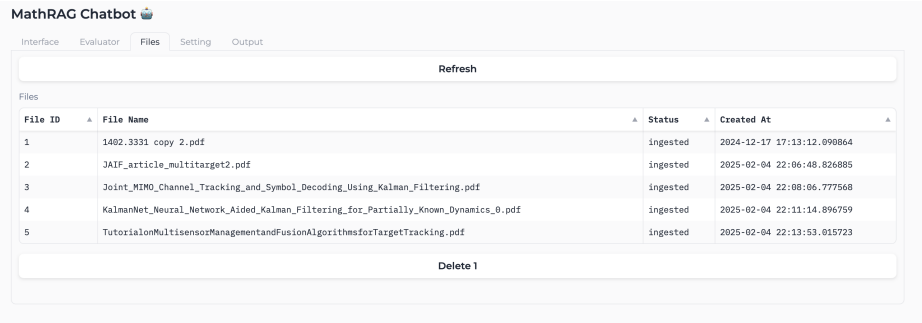


Figure 5.4: Files Section where users can review their uploaded documents.

5.6.3 Deployment

Our RAG chatbot is deployed using cloud platforms and containerization techniques to ensure scalability and high availability. The deployment strategy includes:

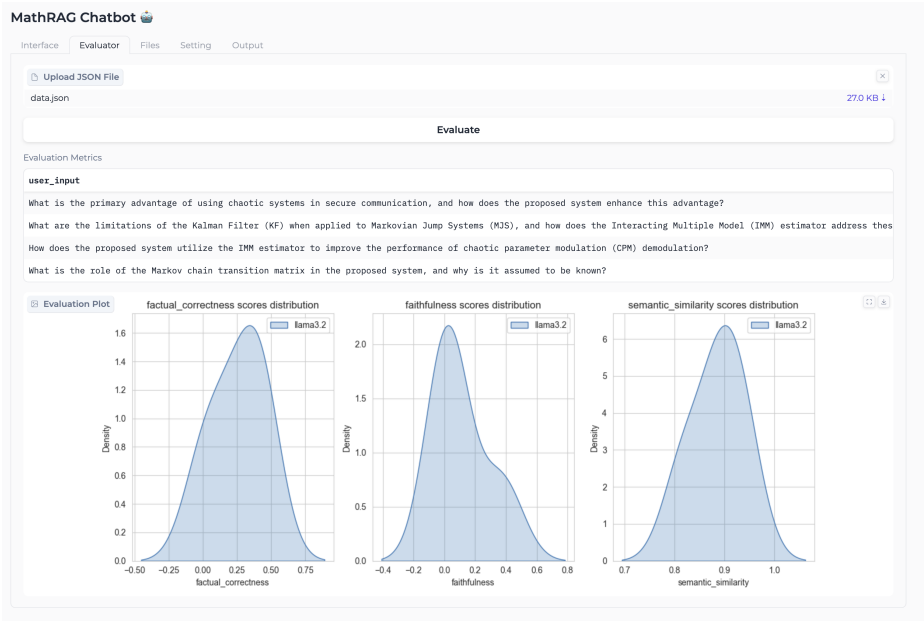


Figure 5.5: Evaluation Results: A section where users can view the dataset and corresponding plots generated for the selected model's evaluation.

- **Containerization:** Packaging the application in Docker containers to enable easy deployment across different environments.

- **Cloud Integration:** Utilizing platforms such as AWS, GCP, or Azure to host the application, thereby ensuring robustness and scalability.

This integrated approach ensures that the chatbot is accessible to users, maintains high performance under load, and provides an interactive and flexible environment for both end users and system administrators.

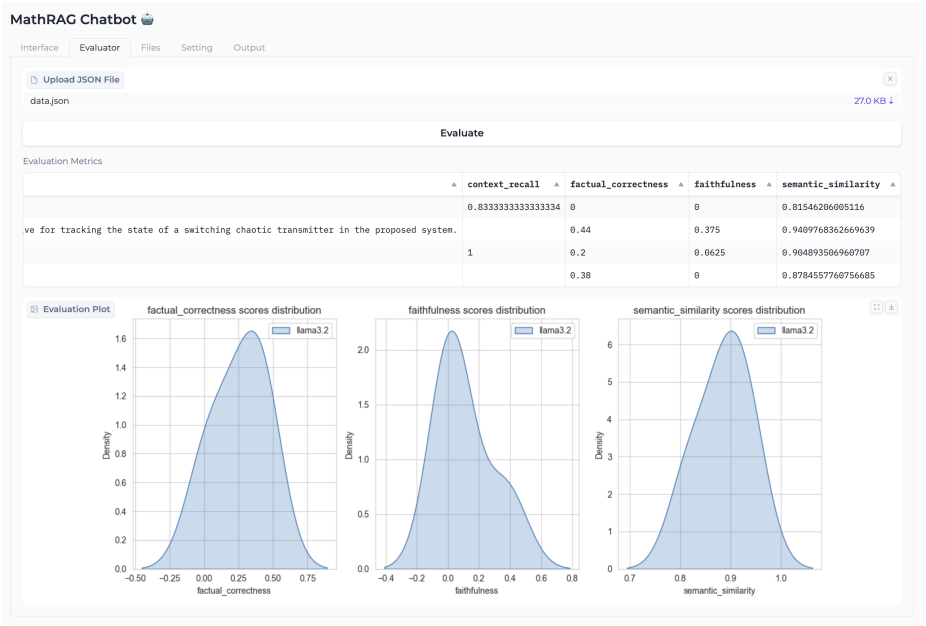


Figure 5.6: Evaluation Results: A section where users can view the relevant metrics and plots associated with the evaluation of the selected model.

Chapter 6

Evaluation and Discussion

In this chapter, we evaluate our QA Retrieval-Augmented Generation (RAG) system using the RAGAS (Retrieval Augmented Generation Assessment) framework as proposed by [5]. RAGAS is designed to assess QA RAG systems along two primary dimensions:

1. **Retrieval Component:** This dimension evaluates the ability of the retrieval system to identify and provide relevant contexts. In our evaluation, we use the metric *Context Recall (CR)* to quantify how effectively the system retrieves the necessary background information.

2. **Generation Component:** This dimension assesses the LLM's capability to leverage the retrieved context to produce accurate and meaningful responses. For this purpose, we consider three metrics:

    - **Factual Correctness (FC):** Measures the degree to which the generated response is factually accurate.

- **Faithfulness (FF):** Evaluates whether the response is fully grounded in the provided context.
- **Semantic Similarity (SS):** Assesses how closely the generated response aligns with the semantics of the query and the context.

To compare the performance of different LLMs—including popular off-the-shelf models as well as our fine-tuned LLM—we conducted a series of experiments and measured these metrics. Table 6.2 summarizes the evaluation results for each model.

LLM	**CR**
OpenAI GPT-4o	0.64
Mixtral 8x7B	0.61
Llama 3 8B (Base)	0.62
Fine-tuned LLM	**0.85**

Table 6.1: Comparison of LLMs evaluated on retrieval metrics. The retrieval metric used is Context Recall. Values represent normalized scores (0 to 1).

LLM	**FC**	**FF**	**SS**
OpenAI GPT-4o	0.35	0.42	0.94
Mixtral 8x7B	0.24	0.38	0.94
Llama 3 8B (Base)	0.27	0.43	0.93
Fine-tuned LLM	**0.45**	**0.45**	**0.94**

Table 6.2: Comparison of LLMs evaluated on generation metrics. The generation metrics include Factual Correctness, Faithfulness, and Semantic Similarity. Values represent normalized scores (0 to 1).

Discussion: The evaluation results demonstrate that our fine-tuned LLM outperforms the other models across all metrics. A high Context Recall indicates that the retrieval system is highly effective at fetching relevant context, while the improved Factual Correctness, Faithfulness, and Semantic Similarity scores of the fine-tuned

model highlight its ability to generate responses that are both accurate and well-grounded in the retrieved context.

These findings underscore the importance of a robust evaluation framework, such as RAGAS, for diagnosing the strengths and weaknesses of QA RAG systems. The metrics provide actionable insights that can guide further improvements in both the retrieval and generation components.

Future work will focus on exploring additional evaluation dimensions, such as user satisfaction and response latency, as well as experimenting with hybrid retrieval strategies to further enhance performance.

Chapter 7

Conclusion

In this thesis, we set out to develop a retrieval-augmented generation (RAG) chatbot tailored for scientific documents in the target tracking domain, with a particular focus on understanding and generating LaTeX-formatted content. Our work combined advanced document extraction techniques, notably using the Facebook Nougat model for converting PDFs to Markdown, with a fine-tuning pipeline built on the Llama 3 8B model using Low-Rank Adaptation (LoRA). This integrated approach enabled the model to effectively handle complex technical content and generate detailed, step-by-step explanations with mathematical reasoning.

Our experimental results demonstrate that our fine-tuning strategy yields a model that performs robustly on domain-specific tasks. The use of LoRA allowed us to efficiently adapt a large pre-trained model while preserving its core capabilities, thereby balancing performance with computational efficiency. Although alternative methods such as PPO and DPO were considered, LoRA's resource efficiency, regularization benefits, and seamless integration into the inference pipeline made it the most suitable choice for our target use case.

While our model shows promising performance in understanding and generating mathematical content, there remain several challenges. In particular, the document extraction process, especially when dealing with diverse PDF layouts, still exhibits occasional errors, and further work is needed to improve consistency across entire documents. Future research will focus on enhancing document segmentation, refining retrieval mechanisms, and exploring more sophisticated reasoning techniques to further boost the model's accuracy and robustness.

Overall, this thesis has contributed a comprehensive framework for building a domain-specific RAG chatbot. Our approach, which spans data collection, advanced document conversion, and parameter-efficient fine-tuning, offers valuable insights into adapting large language models for specialized scientific applications. The findings not only advance the state-of-the-art in retrieval-augmented generation but also provide a solid foundation for future work in leveraging LLMs to make complex technical information more accessible and actionable.

Bibliography

[1] R. T. Andrea Matarazzo. A survey on large language models with some insights on their capabilities and limitations. *arXiv preprint*, 2025. URL `https://arxiv.org/abs/2501.04040`.

[2] Y. Z. J. W. L. F. R. C. K. C. T. Beichen Huang, Xingyu Wu. Evaluating the black-box optimization capability of large language models. *arXiv preprint*, 2024. URL `https://arxiv.org/abs/2404.06290`.

[3] L. Blecher, G. Cucurull, T. Scialom, and R. Stojnic. Nougat: Neural optical understanding for academic documents. *arXiv preprint arXiv:2308.13418*, 2023. URL `https://arxiv.org/abs/2308.13418`.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[5] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert. Ragas: Automated evaluation of retrieval augmented generation. *arXiv preprint arXiv:2309.15217*, 2023. doi: 10.48550/arXiv.2309.15217. URL `https://arxiv.org/abs/2309.15217`.

[6] S. Es, J. James, L. Espinosa Anke, and S. Schockaert. RAGAs: Automated evaluation of retrieval augmented generation. In N. Aletras and O. De Clercq, editors, *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158, St. Julians, Malta, Mar. 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.eacl-demo.16/.

[7] T. Gao and Others. Retrieval-augmented generation for llms: A guide. *Preprint on ArXiv*, 2023. URL https://arxiv.org/abs/2312.10997.

[8] S. Gupta, R. Ranjan, and S. Singh. A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions, 10 2024.

[9] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint*, 2021. doi: 10.48550/arXiv.2106.09685. URL https://arxiv.org/abs/2106.09685.

[10] M. Idowu. The rise of transformers: A deep dive into gpt architecture. 11 2024.

[11] M. Jong, Y. Zemlyanskiy, J. Ainslie, N. FitzGerald, S. Sanghai, F. Sha, and W. Cohen. Fido: Fusion-in-decoder optimized for stronger performance and faster inference, 12 2022.

[12] T. Kudo and J. Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 66–71, 2018. URL https://arxiv.org/abs/1808.06226.

[13] A. Lazaridou, A. Mensch, T. Cai, and Others. Internet-augmented language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL `https://arxiv.org/abs/2203.05115`.

[14] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain, 2004. Association for Computational Linguistics.

[15] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL `https://arxiv.org/abs/1711.05101`.

[16] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers. MTEB: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2764–2777, 2023. URL `https://aclanthology.org/2023.eacl-main.148/`.

[17] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, Philadelphia, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135.

[18] V. B. Parthasarathy, A. Zafar, A. Khan, and A. Shahid. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities (version 1.0). *arXiv preprint*, 2024. URL `https://arxiv.org/abs/2408.13296`.

[19] S. Ravada, S. Gopinathan, and L. Bhaskari. Identification of spatial relations in mathematical expressions. 16:346, 07 2022. doi: 10.24412/1932-2321-2021-465-346-353.

[20] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1715–1725, 2016. URL `https://arxiv.org/abs/1508.07909`.

[21] R. Shuttleworth, J. Andreas, A. Torralba, and P. Sharma. Lora vs full fine-tuning: An illusion of equivalence, 10 2024.

[22] M. Sloan and J. Wang. Dynamic information retrieval: Theoretical framework and application. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, University College London, 2015. ACM.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017. URL `https://arxiv.org/abs/1706.03762`.

[24] X.-P. Vu and Others. Freshllms: Improving large language models for domain-specific tasks. *Proceedings of ACL 2024*, 2024. URL `https://arxiv.org/abs/2310.03214`.

[25] X. Wang and Others. Instructretro: Instruction tuning for post-training llms. *Preprint on ArXiv*, 2024. URL `https://arxiv.org/abs/2310.07713`.

[26] Y. Wu, M. Schuster, Z. Chen, Q. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. In *Preprint on ArXiv*, 2016. URL `https://arxiv.org/abs/1609.08144`.

[27] W. Xiong and Others. Effective adaptation of llms for specialized domains. *Proceedings of ACL 2024*, 2024.

[28] W. Zhang and Others. Raft: Adapting language models to new tasks with minimal data. *ICLR 2024*, 2024. URL `https://arxiv.org/abs/2403.10131`.

[29] Z. Zhang, M. Fang, L. Chen, M.-R. Namazi-Rad, and J. Wang. How do large language models capture the ever-changing world knowledge? a review of recent advances. *arXiv preprint*, arXiv:2310.07343, 2023. doi: 10.48550/arXiv.2310.07343. EMNLP 2023 main conference.

[30] B. Zhou and Others. Lima: Alignment techniques for large-scale ai models. *ICLR 2023*, 2023. URL `https://arxiv.org/abs/2305.11206`.

[31] S. Zhuang, H. Ren, L. Shou, and Jian. Bridging the gap between indexing and retrieval for differentiable search index with query generation. *arXiv preprint arXiv:2206.10128*, 2022. doi: 10.48550/arXiv.2206.10128. URL `https://arxiv.org/abs/2206.10128`.