National College of Ireland

Evaluating Trade-offs Between Performance and Resource Utilization in Retrieval-Augmented Generation and Finetuning for Domain-Specific Language Models

MSc Research Project
MSc in Artificial Intelligence

Aryan Ramani
Student ID: 23235438

School of Computing
National College of Ireland

Supervisor: Abdul Shahid

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aryan Ramani
Student ID:	23235438
Programme:	MSc in Artificial Intelligence
Year:	2025
Module:	MSc Research Project
Supervisor:	Abdul Shahid
Submission Due Date:	11/08/2025
Project Title:	Evaluating Trade-offs Between Performance and Resource Utilization in Retrieval-Augmented Generation and Finetuning for Domain-Specific Language Models
Word Count:	6960
Page Count:	23

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	10th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Evaluating Trade-offs Between Performance and Resource Utilization in Retrieval-Augmented Generation and Finetuning for Domain-Specific Language Models

Aryan Ramani
23235438

Abstract

This report investigates the trade-offs between Retrieval Augmented Generation (RAG) and fine-tuned Language Models for a domain-specific question answering task. A high-performing large language model (LLM) delivers practical value only when it can be deployed within the constraints of the target environment. We focus on NASA technical reports to generate a dataset, a perfect fit for jargon-rich domain. First, we extract the content of the PDFs and generate corresponding questions. Secondly, we conduct experiments using simple RAG architecture and LLMs finetuned exclusively on our dataset. Lastly, we evaluate both the approaches based on the standard performance metrics as well as the computation resources consumed to produce the associated results. The performance analysis aspect of our research highlights the strength and weaknesses of each method in handling domain queries. Whereas resource monitoring ensure that the techniques are efficient and scalable enough to be utilized in real-life practices given the resource constraint environments. Based on the results, we recommend the more effect approach while providing practical guidelines for choosing retrieval-based and finetuning-based strategies, balancing performance with the available resources.

1 Introduction

Large Language Models (LLMs) have shown great capabilities across a wide range of natural language processing tasks including text generation, question answering and summarization. These models are trained on huge amount of general-textual data available through sources like internet, which enables them to understand and generate human-like language. However, when applied to specialized technical domains such as science, LLMs often face limitations due to lack of domain knowledge leading to inaccurate generations and fictitious information. To address this issue, two approaches have been widely used by practitioners: Retrieval-Augmented Generation (RAG) and Fine-Tuning.

RAG enhances the quality and accuracy of generation by providing relevant information from external sources like PDFs. This method allows the model to support its generation by using domain specific knowledge. It also helps the LLM to be aware about latest information, as LLMs suffer from a knowledge cutoff, meaning it has been trained

on data up-to a specific point in time. On the other hand, Fine-tuning involves a re-training of an existing LLM on a specific dataset to tune or adapt its parameters to incorporate knowledge from that specific domain. This approach enables the model to learn the domain specific knowledge at the retraining stage rather than inference stage, unlike RAG. The two are increasingly being adopted by specialized domain like law, science and medicine. With factual knowledge support, their use improves the decision making in knowledge-intensive work environments.

Both of these approaches, also have computational costs associated with them. RAG uses an external document store as well as a vector database that store the knowledge base and their vector embeddings respectively. RAG also requires prepend the information retrieved to the input, resulting in a longer context sequence, hence more text processing. Fine-tuning doesn't require to maintain an external source, however, the approach itself is resource-intensive as it requires updation of its parameters while understanding the specific corpus. This allows the model to retain knowledge in itself, eliminating the need to append knowledge into input, hence processing small context sequences.

This raises the question: **"Which of the two techniques - Retrieval Augmented Generation or Fine-Tuning - outperforms the other with respect to the quality and accuracy of generation as well as the computational resources required for a specific domain?"**

The key contributions of this research are:

1. Construction of a custom domain-specific dataset.
2. Evaluation of RAG and Fine-Tuning techniques on technical content.
3. Analysis of trade-offs between performance, accuracy and computation resources for both approaches.

We chose aerospace engineering as the experimental domain and formulated a custom dataset using publicly distributable NASA Technical Reports. We chose this domain as there is a growing demand for understanding complex technical literature. Aerospace engineering reports posses a unique challenge by being under-explored domain.

First, this report provides an overview of the foundational literature and existing research related to Large Language Models, Retrieval-Augmented Generation, and fine-tuning techniques along with any prior experiments in the field of astronomy or aerospace engineering. Later, the methodology for creation of dataset from NASA Technical Reports including the preprocessing and text extraction, along with the implementation details of both RAG and fine-tuning approaches are discussed. It also provides an overview of evaluation metrics used. Furthermore, the design decisions for the setup of experiments are discussed like choosing the LLMs, Retriever for RAG and Fine-tuning technique along the hardware specification to carry out the experiments. After the design, the actual implementation for both approaches are documented including the specific tools and technologies used to develop the dataset as well as the actual experiments and evaluation. Penultimately, the actual experiments along with the detailed analysis of the results are interpreted. The comparison and performance of RAG and fine-tuning is also discussed here, followed by a discussing the advantages and limitations of each. Lastly, we conclude the report by summarizing the key findings if the research, and outline any potential future directions for improvements in the research area.

In summary, this report investigates the trade offs between two prominent techniques - RAG and fine-tuning - when applied to Aerospace Engineering by evaluating their performance, accuracy and computation resources used. The following sections highlights the existing work, experimental setup and analysis of the research findings.

2 Related Work

This section consists of a in depth and critical analysis of the existing literature in the area of Large Language Models along with Retrieval Augmented Generation and Fine-tuning. Furthermore, we address the current state-of-the-art and the gaps in the prior research.

2.1 Overview and Evolution of LLMs

Natural Language Processing has improved significantly in the recent times with the introduction of Transformer-based architecture. Introduced by Vaswani et al. (2023), the Transformer architecture rely on self-attention using scaled dot product between query, keys and values to process sequential data to enable parallelizable training process, unlike traditional recurrent models. Following the transformers model, two new models-BERT (Bidirectional Encoder Representation from Transformers) and GPT (Generative Pre-trained Transformer) emerged. BERT (Devlin et al.; 2019) introduced deep bidirectional embeddings, which improved performance on tasks like sequence classification, named entity recognition, and question answering. In contrast, GPT (Radford et al.; 2019) used unidirectional encoding from the transformer decoder architecture for auto-regressive language model, excelling in tasks like text generation. Later, GPT series saw addition of models like GPT-3 (Brown et al.; 2020), scaling the number of parameters to 175B. It demonstrated excellent zero-shot and few-shot learning capabilities on tasks it was not trained on, revealing the importance of model scaling for generalization. In parallel, models like T5 (Text-to-Text Transfer Transformer) (Raffel et al.; 2023) and BART (Bidirectional and Auto-Regressive Transformer) (Lewis et al.; 2019) incorporated generative capabilities with discriminative under a common sequence-to-sequence framework to expand the existing utilities of LLMs across tasks like summarization, translation and sequence classification.

Recently, open-source initiatives like Meta's Llama (Touvron et al.; 2023) have allowed researchers and practitioners to access efficient LLMs, offering competitive performance with significantly less number of parameters. Llama models have demonstrated that fine-tuned models can perform significantly better than their large counterparts when trained on high-quality dataset and longer context lengths.

2.2 Retrieval-Augmented Generation (RAG)

While LLMs exhibit strong generative capabilities as discussed in Section 2.1, they are limited to the knowledge they are trained on. This results in generate factually false responses on niche or recent topics, especially in domains high dependent on factual knowledge. Huang et al. (2025) reasons that hallucinations are caused by the limitations of the training data of the LLM along with its misinformation and biases as neural networks tend to memorize training data. To counter this limitation, **Retrieval-Augmented**

Generation (RAG) was introduced by combining the LLM with an external document source consisting factual knowledge (Lewis et al.; 2021).

RAG Architecture retrieve the most relevant documents from an external knowledge based on the input query and condition the model's generation on the retrieved text(s). This allows the model to access the domain-specific information during inference stage, improving factual accuracy and reducing hallucinations. RAG uses a embedding model at the retrieval step that converts text to dense embeddings using models like BERT to perform similarity search (Karpukhin et al.; 2020). A Vector Store is also required to store the embeddings of domain-specific knowledge base (e.g FAISS (Douze et al.; 2024)).

RAG enables LLM to scale their knowledge without retraining. However, it introduces a new dependency on the retrieval quality, based on the embedding model and similarity metric. Poorly retrieved contexts can degrade the performance, making the entire system highly sensitive to retrieval pipeline.

2.3 Fine-Tuning

Fine-tuning is a widely used technique for LLMs for task-specific objectives. While LLMs demonstrate strong performance, they often underperform in specialized domain due to vocabulary mismatch, unfamiliar terms or task-specific nuances (Gururangan et al.; 2020). Moreover, Parthasarathy et al. (2024) provides a detailed importance of fine-tuning for domain specific tasks by adjusting to the nuances, vocabulary and jargon of the specific domain.

Finetuning comprises of further training of the model on a smaller dataset relevant to the target domain. The structure of dataset depends on the specific type. An unsupervised finetuning would require plain textual data that helps the model to understand the domain-specific language. On the contrary, for classification or summarization tasks would required a labeled or a well structured dataset. Moreover, instruction based fine-tuning requires instructions relevant to the task along with dataset. Model retraining allows the model to adjust its weights or parameters to better capture the task-specific or domain-specific semantics, lead to an improved accuracy (Lee et al.; 2019; Beltagy et al.; 2019).

Finetuning requires a significant amount of computational resource, especially for models with billions of parameters. It also risks the loss general capabilities learned by the model during pre-training phase. To mitigate these issues, parameter efficient techniques like Adapter Layers and LoRA have gained attraction (Houlsby et al.; 2019; Hu et al.; 2021)

2.4 LLMs in Aerospace Engineering Domain

LLMs have been experimented and adopted in domains like healthcare and law, excelling in complicated tasks like Question-Answering (QA), Summarization, and document understanding. However, their application in aerospace engineering technical reports, especially NASA technical reports, have been unexplored. The field of aerospace engineering is rich in jargon, structured documents, domain specific language which posses an interesting challenge to LLMs trained on general purpose dataset.

Recent work like, SciBERT (Beltagy et al.; 2019), has explored domain adaptation of LLMs through fine-tuning on general purpose scientific and engineering texts, but lacked training on Datasets specific to NASA technical reports. SciBERT is also a decoder only

architecture, lacking capabilities of text generation. AstroQA (Jie Li; 2025) introduced a dataset, mostly consisting of Multiple Choice Question and Short Answers, but it focus on general astronomy rather than specific technical details.

3 Methodology

This section contains a detailed description of the proposed methodology that will be utilized to conduct the research. From creation of dataset to the LLM experiments, we provide a comprehensive overview of each step in the pipeline. Each component is designed to ensure the study is able to address the research question.

3.1 Creation of Dataset

The fundamental step to perform domain-specific evaluation on RAG and Finetuning is to formulate a domain-specific dataset. In order to prepare a high quality, a multi-step data extraction and pre-processing approach was adopted. The main goal of the pipeline was to extract structured and meaningful text from documents, while neglecting the irrelevant and noisy content like list of contents, header and footer. The following sections discuss the methodology used to create the dataset.

3.1.1 Document Selection

The document corpus consisted of six publicly available NASA technical reports covering diverse aspects of space systems engineering, including thermal analysis, fracture control, programmable logic devices, and systems modeling.

- **NASA System Engineering Handbook** — Revision 2
- **Fracture Control Implementation Handbook for Spaceflight Hardware** — Revision, 15/12/2023
- **NASA Thermal Analysis Handbook** — Revision, 25/09/2023
- **Programmable Logic Devices (PLD) Handbook** — Revision, 18/06/2025
- **Engineering Elegant Systems: Theory of Systems Engineering** — Revision, 06/2020
- **NASA Systems Modeling Handbook for Systems Engineering** — Revision, 12/03/2025

Each document provides a unique challenge for the LLM, consisting of technical details of a sub-domain of the aerospace engineering domain.

3.1.2 Block Level Extraction

The first step was to extract block level text rather than page level to ensure localized text from relevant sections is captured, avoiding unrelated sections like headers and footers. By focusing on blocks, the technique helps in retaining the context and structure within each block, which is essential for downstream tasks that depend on coherent textual sections.

3.1.3 Section Wise Grouping

Once individual blocks were extracted, they were grouped together based on the section they belonged to. For example, all blocks of text under section 2.1 were grouped together until the next section header i.e. 2.2 was encountered. This ensured that the grouped text maintained the logical flow and hierarchy of the original document. Section-wise grouping helps maintain the semantic meaning of the section, which is particularly beneficial for models that need structured contextual input.

3.1.4 Semantic Chunking

After grouping section together, semantic chunking was performed as a safeguard, so that the sections are divided into semantically meaningful units. The chunker compares surrounding block using semantic similarity by converting them into vector embeddings.

3.1.5 Filter Criteria

All text consists of words less than 50 words or more than 300 words to filter out paragraphs that are too long or short for a decent evaluation. Moreover, words consisting of less than 7 words per line, often representing labels, table text or incomplete text fragments were also left out.

3.1.6 Question Generation

After extraction all the relevant text from PDFs, an high quality LLM was used to generation question for the answers. Each text was passed as a single answer, for which LLM generation a single question. This was done so that one question only had one support answer to remove ambiguity during evaluation.

The dataset curation methodology has various advantages. It ensures that text consisting of similar semantic meaning are grouped together, while filtering out unnecessary blocks. Together these techniques produces a well suitable dataset for inference and training tasks for domain-specific experiments.

However, this methodology has some limitations. Mathematical expressions and equations are excluded as they are often treated as separate blocks, limiting the dataset to text only blocks. Moreover, bullet points or indexed text are not handled as effective as paragraph style text, resulting in an information cutoff. Lastly, the use of static heuristics like word count thresholds may lead to the unintentional exclusion of certain valuable content.

3.2 Retrieval-Augmented Generation

This section discusses about the methodology of first of the two experiments explored in this study. The primary objective of this experiment is to evaluate the performance of RAG systems on domain specific dataset curated in Section 3.1. The responses generated by a RAG system is based on information retrieved from a fixed corpus.

The same document corpus served as the knowledge base for the retriever. These documents were semantically chunked into a minimum chunk size of 100 words/tokens to maintain some sort of semantic coherence during retrieval. For document retrieval, a dense vector similarity search retrieval strategy was used, with a sentence embedding

model serving to generate vector embeddings (Karpukhin et al.; 2020). A Vector Store was utilized to store documents and their embeddings for scalable and efficient retrieval out of a big corpus. The top 2 similar chunks were selected, determined by cosine similarity between question and the chunks. This technique ensures that the LLM receives factual information to assist in answer generation.

To maintain the consistency of across the experiment, a common prompting template was used to reduce computational redundancy. An example template for the prompt is:

RAG Prompt Template

```
[Instruction]

Context:
[Retrieved Document 1]
[Retrieved Document 2]

Question:
[User Question]

Answer:
```

Prompts were generated all at once to be used as the standardized input for 3 LLMs that were used for the experiment. All prompts were saved in the dataset to reduce experimentation overhead and unnecessary computation to retrieve documents again and recreate prompt.

The RAG method provided a flexible mechanism for appending domain knowledge without updating model weights. By limiting LLM responses in external documents, this approach seeks to reduce hallucinations and improve domain expertise, particularly when models are not explicitly fine-tuned on the target domain.

3.3 Fine-Tuning

The second experiment consist of fine-tuning the same LLMs used to develop the RAG system. This approach firsts fine-tunes the LLM on domain specific dataset using raw extracted texts from PDFs and then uses the questions generated for performance evaluation.

The raw text consists of all the blocks of text from PDFs mentioned in Section 3.1 instead we do not filter out information other than very small block of text with less than 50 words, as the most small chunks for entire dataset was 50 words. To mitigate the issue of catastrophic forgetting, a condition where model forgets its general purpose instruction following capabilities, we interleaved 4000 examples from Alpaca dataset (Taori et al.; 2023) to ensure that model retains its previous capabilities while learning new domain-specific knowledge.

Low-Rank Adaption (LoRA) (Hu et al.; 2021) was employed for parameter-efficient fine-tuning. It injects trainable rank-decomposed weight matrices into each transformers block, reducing the number of trainable parameters. This approach allowed us to perform fine-tuning in resource-constrained environments by avoiding to update the full base model weights.

Post fine-tuning, same question asked to RAG system was asked to the LLMs without context to let it generate answers from new domain-specific information gained.

3.4 Resource Monitoring

This study also aims to provide comprehensive insights into trade-offs between the performance of systems with the computational resources required. Achieving higher performance is always the primary goal of the research experiments, it is also crucial consider the associated costs in terms of memory and time consumption. For instance, a 10% improvement in performance may seem rational at first, but if it utilizes three times the resources along with slower inference, it raises the question, "is the method practically feasible?". We monitor the entire RAG pipeline, from database creation to answer generation and fine-tuning stage along with the answer generation phase.

Ultimately, the aim is not only to build the most powerful system, but also to be aware about resource constraints in production environments to make best informed decisions.

4 Design Specification

The two underlying techniques that we use - RAG and Finetuning - are adopted in a very simplistic way. The goal is to determine what advantages a technique posses, while using the resources they require rather than building a novel approach or designing a complex system.

4.1 Retrieval Augmented Generation

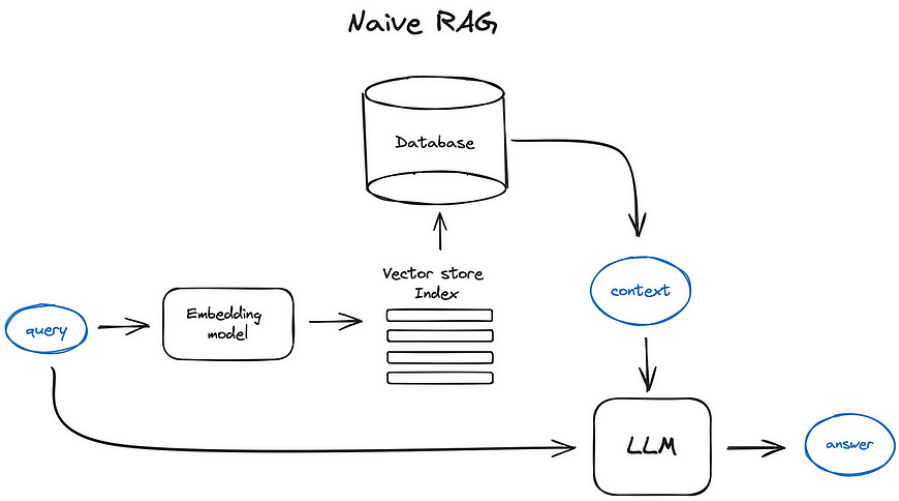

Figure 1: Architecture for RAG

Figure 1[1] shows the architecture we use for RAG experiments. It is simple architecture which combines a retriever using Vector Database containing the domain documents, with the LLM to generate answers.

4.2 Finetuning

Figure 2[2] shows the architecture we use for fine-tuning experiments. It is a straightforward process where a pre-trained model is adapted to Question Answering task using NASA

[1]Source: https://ai.gopubby.com/advanced-rag-techniques-unlocking-the-next-level-040c205b95bc
[2]Source: https://medium.com/@avikumart_/evaluation-of-fine-tuned-llm-using-monsterapi-a67a7714a65b

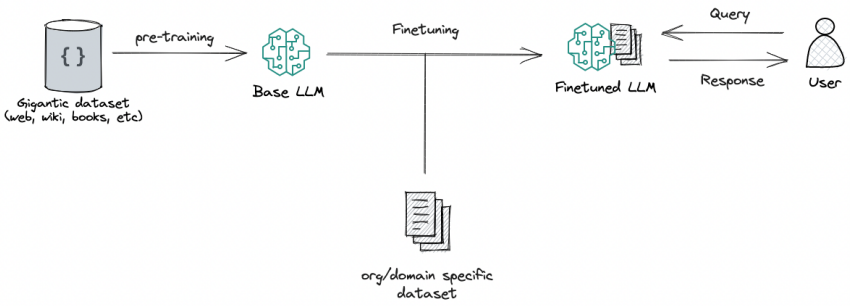

Figure 2: Architecture for Finetuning

Technical Reports Dataset (discussed in section 3.1), allowing the model to specialize while retaining its general language understanding. Specific details on finetuning are discussed later on in section 5.3.2.

5 Implementation

This section outlines the practical steps taken to develop a system for the proposed methodology. It details the technical execution of each phase, including data processing, model setup, RAG architecture, finetuning procedures, and evaluation. The main emphasis is placed on the tools, libraries, and frameworks used, along with relevant implementation decisions that shaped the system's performance.

5.1 Dataset Creation

Dataset creations consist of two steps: extraction of actual PDF text and generation of question using the extracted text. Both approaches are discussed below in detail.

5.1.1 Answer Extraction

To implement the domain-specific knowledge extraction described in section 3.1, `fitz` library was used in Python for extracting PDF content. This library provides a simple and powerful interface for block-level text extraction.

Each document was opened using fitz and read page by page. Instead of extracting the whole page, block level extraction was implemented using `page.get_text("blocks")`, which returns a list blocks within a specified bounding box to ensure headers and footers were ignored. Later, addition filtering is done to ignore blocks containing elements other than text, like figures.

Section headers were detected using simple regular expressions that identified patterns like 1.1, 2.2.3, 5 etc.. Subsequent blocks were combined together until next section was discovered.

Last pre-processing, Semantic Chunking was performed using Langchain framework[3] to divide or combine the text into meaningful sections. Sentence Embeddings were generated for each of the section using a pretrained sentence transformers model. Later, these sections were merged or split based on the cosine similarity between adjacent blocks. This

[3]https://www.langchain.com/

approach made sure each chunk had a complete and meaningful context, and prevented abrupt incompletion of text. Lastly, chunks with less than 50 words, more than 300 words or average words per line less than 7 were discarded.

5.1.2 Question Generation

OpenAI's GPT 4o-mini (OpenAI; 2024) was used to generate questions. Using the following prompt:

> # Identity
>
> You are a question answer generator for Aerospace Engineering. Your task is to frame a question that is descriptive enough but not too long for the give text. Please do not generate the question only, and no extra text.
>
> # Instructions
>
> * Only create question for the text given. If it refers to another section, do not include that in question.
>
> * However, if it only refers to a section frame a question like, "Where does this particular information lie?"

5.1.3 Inter-Annotator Agreement

To assess the quality of the dataset curated, two annotators independently rated each pair of question-answer on a scale from 0 (no relevance) to 5 (highest relevance). Inter-annotator agreement was measured using the Cohen's Kappa coefficient, which account for agreement occurring by chance. The Cohen's Kappa score was 0.7789, indicating a substantial agreement among both parties according to the interpretation of kappa values (Landis and Koch; 1977).

The final output `.csv` file consisted of two columns: "question" and "answer". This ensure easily accessible and ready to use dataset to be used for experiment later on.

5.2 Retrieval-Augmented Generation

The approach for RAG experiments consist of implementation in 2 different steps- creation of prompt using retriever and LLMs experiments.

5.2.1 Retriever

The Retriever pipeline was implemented in Python 3.9 on a macOS system (MacBook Air M1, macOS Sequoia 15.4.1). It was developed using the `LangChain` framework, which acted as a central interface for integration between embedding models, vector storage and query-based document retrieval.

Table 1: System Specifications for CPU experiments

Component	**Specification**
Processor	Apple M1 (8-core CPU)
GPU	Integrated 7-core GPU
RAM	8 GB
Storage	256 GB SSD
Operating System	macOS Sequoia 15.4.1
Python Version	Python 3.9
Architecture	ARM64 (Apple Silicon)

The embedding model was incorporated using Langchain's abstraction over huggingface[4] and sentence-transformers[5] library using the **all-MiniLM-L6-v2** model (Reimers and Gurevych; 2019; Wang et al.; 2020). This model was used to generate dense semantic embeddings for both the input questions and the document chunks. The six NASA documents, mentioned in Section 3.1 were first split into semantically similar text of minimum 100 words chunk size using **SemanticChunker**. The generated chunks were stored in ChromaDB[6] Vector Database using Cosine Similarity as the metric for retrieval. Although the database was persisted for backup purposes, the actual retrieved contexts were stored alongside the corresponding question-answer pairs for direct use in LLM evaluation, reducing need for repetitive retrieval. Retrieval was performed with a top-k strategy with $k = 2$, meaning the top 2 most relevant documents will serve as the support document for the LLM and appended to the input prompt.

These generated prompts were stored in a CSV file along with the original question and answer pairs to ensure consistency across multiple LLM evaluations. No custom chunking or embedding scripts were required, as all components were handled using LangChain's built-in modules with default settings or minor changes in settings.

5.2.2 LLM Answer Generation

All LLM inference experiments were conducted on RunPod.io[7], using the container image `runpod/pytorch:2.7.0-py3.10-cuda11.8.0-devel-ubuntu22.04`. This container was pre-configured with PyTorch 2.7, Python 3.10, CUDA 11.8, and Ubuntu 22.04, providing a stable environment for running GPU-accelerated PyTorch applications. The cloud instance was equipped with a 48GB NVIDIA A40 GPU and 9 virtual CPU cores, providing resources capable of loading decent-scale language models.

[4]https://huggingface.co/
[5]https://sbert.net/
[6]https://www.trychroma.com/
[7]https://www.runpod.io/

Table 2: RunPod Inference Environment Specifications

Component	**Specification**
Operating System	Ubuntu 22.04
Python Version	Python 3.10
PyTorch Version	PyTorch 2.7
CUDA Version	CUDA 11.8
GPU	NVIDIA A40 (48 GB VRAM)
CPU	9 vCPUs
RAM	50 GB

Table 3 lists the LLMs that were used to conduct the experiments. All models were downloaded locally using the huggingface API. The "text-generation" pipeline was employed for efficient parallel inference with a batch size of 16, to ensure maximum GPU utilization. Prompt inputs where provided to the model using a **KeyDataset** class object to feed the model only required information from the already generated CSV file.

Table 3: LLMs Used for RAG Inference

Model Name	**Parameters**	**Quantized**
google/gemma-3-4b-it	4B	No
meta-llama/Llama-3.1-8B-Instruct	8B	No
mistralai/Mistral-Small-24B-Instruct-2501	24B	8-bit

Each model was configured with same generation arguments:

1. generating at most **256 tokens**, chosen deliberately as the closest power of two below the dataset's maximum total length of 300 tokens.

2. do_sample=False, ensured deterministic outputs, for a fair comparison across all the models.

This configuration limits deterministic and concise responses while also controlling the output length. Quantization of the Mistral model to 8-bit enabled it to fit within the available GPU memory and creating a diversified pool of LLMs used in the experiment. All models were evaluated using identical prompts, ensuring fairness and standardization in output comparison.

5.3 Fine-Tuning

The Finetuning approach consist of 3 steps, unlike RAG which consisted of 2 steps. First, we curate a mix dataset discussed in section 3.3. Then, LLM fine tuning is performed. Lastly, questions were asked for answer generation to perform evaluation.

5.3.1 Dataset Setup

We used `datasets` library, developed by huggingface, to curate the dataset as well as save it on cloud for fine-tuning purposes. A ratio of 63:37 was curated, consisting of 6831 sample from domain specific dataset and 4000 general purpose instruction Alpaca dataset. The instruction dataset was converted into prompt template before saving into text

column. The dataset is publically available at: https://huggingface.co/datasets/notaryanramani/NASA-alpaca-dataset.

5.3.2 LLM Fine-tuning

The fine-tuning pipeline was implemented using the Huggingface Ecosystem, using libraries like transformers, datasets, peft, accelerate, and bitsandbytes. We also used `Unsloth`[8], an optimization toolkit for fast and resource efficient LLM fine-tuning.

The LLMs used for the experiments are same as LLMs mentioned earlier (see Table 3). However, instead of 8-bit quantization for **mistralai/Mistral-Small-24B-Instruct-2501**, we perform 4-bit quantization with LoRA adapters due to resource limitations on our hardware (see Section 5.2.2).

Table 4: Training Hyperparameters used in SFTConfig

Parameter	Value
per_device_train_batch_size	8
gradient_accumulation_steps	32
warmup_steps	5
num_train_epochs	3
learning_rate	2e-5
logging_steps	20
optim	"adamw_8bit"
weight_decay	0.01
lr_scheduler_type	"linear"

LoRA was applied to both attention as well as MLP layers of the transformer architecture, following Unsloth's recommended configuration for instruction-based as well as domain-knowledge tuning. The rank and alpha value was set to 8. Dropout rate was selected as 0, meaning no dropout was performed for LoRA layers. This configuration balances adaptability and efficiency, making it suitable for fine-tuning large models even on modest hardware setups.

Training Arguments mentioned in Table 4 were same across all models. It uses 8-bit AdamW, linear LR scheduling, and gradient accumulation for efficient fine-tuning, optimizing memory usage.

5.3.3 LLM Answer Generation

The same setup, as discussed in Section 5.2.2, was used to generate answer from the fine-tuned models. However, unlike RAG where we input context along with question, outputs were recorded upon generation from asking only questions, as expected for fine-tuned models.

5.4 Resource Monitoring

To monitor the resource utilization and execution time of the two experiments conducted in the study, we utilized Python's inbuilt libraries along with PyTorch's CUDA utilities. Python's inbuilt module help us to record the starting and ending points for various

[8] https://unsloth.ai/

phases in the implementation. PyTorch's CUDA utilities monitors the GPU memory usage and ensures that results are not impacted by an asynchronous GPU operations. This implementation provides a lightweight process which can be integrated into the source code itself, eliminating any use of external tools like MLFlow. This led to a minimal disruption in the workflow of the experiments for a reliable performance monitoring.

For the RAG experiments, we focus on recording the execution of entire pipeline, from document retrieval to answer generation. To have a common ground between the models and techniques, we use huggingface library and ChromaDB from Langchain for the RAG pipeline, and do not encapsulate the entire pipeline into a Langchain's **chain** object to avoid any abstraction overhead.

For the fine-tuning techniques, we monitor time for the fine-tuning stage as well as the answer generation phase. As mentioned in section 5.3.2, Unsloth was used for all LLMs for an optimized fine-tuning interface. For text generation, a simple huggingface pipeline is used.

6 Evaluation

This section contains the performance metrics as well the computation resources utilized by each experiment - RAG and Finetuning.

To evaluate the performance of both techniques on the question answering tasks, we monitored the following set of metrics: BLEU, ROGUE-1, ROGUE-2, ROUGE-L, BERT Score, and BLEURT. BLEU is used to measure the overlap of n-grams between the generated and reference answers. Originally adopted for machine translation it can help us to estimate fluency and phrase matching (Papineni et al.; 2002). ROUGE is used to assess the content similarity: ROUGE-1 capture individual tokens, ROUGE-2 captures bi-gram overlap, and ROUGE-L evaluates the longest common subsequence, reflecting token similarity between two texts (Lin; 2004). We needed metrics to evaluate the semantic meaning of texts as well, because BLEU and ROUGE fail to capture the context equivalence. We chose BERT Score and BLUERT to mitigate this challenge. BERT Score (Zhang et al.; 2020) compares two text by converting into embeddings using a BERT Model. It is further decomposed into: Precision - how similar are tokens generated, Recall - How much of information is retained, F1 - A harmonic mean among the two; provides a balanced metric. Lastly, BLEURT uses specialized trained models on human judgment data to evaluate text quality in terms of meaning, fluency and naturalness. These metrics provides a good balance of token matching and semantic similarity insights between the generated answers and original texts.

For resource utilization, we monitor the GPU VRAM required and time taken to generate a single answer on the longest text and the shortest text in the dataset. RAG experiments consist of the whole pipeline, from document retrieval to answer generation. On the other hand, Fine-tuning consists of the resources required and time consumed for fine-tuning stage in addition to above mentioned monitoring.

6.1 RAG Results

This section contains the evaluation results for the RAG experiments. We first monitor the performance metrics, and highlight the distinct strengths and weakness of selected models and compare them. Secondly, we monitor the computation resources required and time consumed to generate answer for the entire RAG pipeline.

6.1.1 Performance Metrics

Model	BLEU	ROUGE			BERT Score			BLEURT
		1	2	L	Prec.	Recall	F1	
Google Gemma 3 4B	13.76	41.51	24.65	32.47	71.01	61.95	65.85	49.52
LLaMA 3.1 8B Instruct	16.92	39.36	22.11	27.77	61.96	66.52	63.94	48.94
Mistral Small 3 (2501) Instruct 24B (8bit)	18.80	45.08	23.40	31.86	67.50	65.52	66.20	51.95

Table 5: Evaluation metrics (in %) for RAG Experiments.

Table 5 represents all the performance metrics for selected LLM models using the RAG architecture. The Mistral 24B Instruct model demonstrates the most superior performance overall in most of the cases including BLEU, ROUGE-1, which indicates a high overlapping of generated tokens with the reference text, and BERT Score - F1, BLEURT indicating a good balance of information retention and human-like text generation. On the other hand, Gemma 3 4B, consisting of 6 times less number of parameters, excel at ROUGE 2 and L indicating a high overlap of bigrams and the longest sequence. Moreover, Gemma outperforms every other model in BERT Score - Precision, showcasing ability to generate tokens semantically similar to that of the original domain source.

6.1.2 Resource Consumption

This section reports the GPU resource consumed and time taken to generate the answer for the RAG Experiments. The question that would generate the longest prompt was selected as the benchmark. The pipeline for which monitoring was performed consist of the entire RAG pipeline, from vector database query to answer generation. The retriever used for all experiments and design of the RAG system remains the same for all 3 models.

Model	Generation Time (s)	Max GPU Memory Allocated (MB)	Max Reserved GPU Memory (MB)
Google Gemma 3 4B	11.63	8578.03	8664.00
LLaMA 3.1 8B Instruct	8.90	15743.26	15908.00
Mistral Small 3 (2501) Instruct 24B (8bit)	40.09	25534.97	28834.00

Table 6: Resource Consumption and Generation Time for RAG Experiments

Table 6 indicates the Mistral Small 24B Instruct 2501 consumes the most resources as well as longest time to generate the answer out of the three models because of consisting the most number of parameters. On the other hand, Llama is the fast out of the three while having more parameters than Gemma 3, possibly due to being a simpler and older architecture.

6.2 Fine-Tuning Results

This section contains the evaluation metrics as well as the resources utilized by the fine-tuning methodology, same as section 6.1.

6.2.1 Performance Metrics

This section presents the performance metrics along with the loss results from the fine-tuning stage for all the LLMs selected for the study.

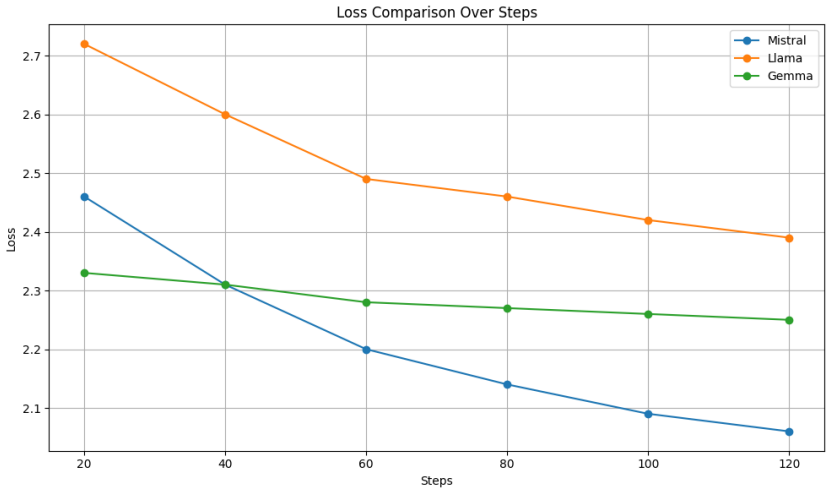

Figure 3: Loss Comparison

Figure 3 shows the loss curves for all the model using the training arguments mentioned in Table 4. Mistral has the lowest loss that the end of the finetuning stage, suggesting that the probability distribution produced is closest to the actual ground truth of the dataset as compared to other models. Gemma started with the lowest loss but is unable to learn from the dataset, having a very steep loss curve. Llama has the highest loss out of the batch, indicating its probability distribution deviates most from the training data, unable to capture the semantic meaning of the text which is reflected in the next segment.

Model	**BLEU**	**ROUGE**			**BERT Score**			**BLEURT**
		1	**2**	**L**	**Prec.**	**Recall**	**F1**	
Google Gemma 3 4B	3.65	30.46	7.95	17.40	53.24	59.23	55.93	41.19
LLaMA 3.1 8B Instruct	2.56	24.31	7.04	15.10	53.61	59.27	56.20	46.14
Mistral Small 3 (2501) Instruct 24B (8bit)	4.39	30.84	9.33	18.98	58.08	58.13	57.80	45.09

Table 7: Evaluation metrics (in %) for Fine-tuning Experiments.

Table 7 shows the evaluation metrics for the finetuned models on the domain specific dataset. Again, the Mistral 24B Instruct model outperforms other models in most of the aspects with higher BLEU and ROUGE scores. However, LLaMA 3.1 8B Instruct performs best for BLEURT metric, indicating a semantically similar text to the original

documents. Overall, models have significantly underperformed as compared to the RAG architecture. Potential reasons for the decline in performance will be discussed later in section 6.3.

6.2.2 Resource Consumption

This section presents the resource utilized and time consumed to finetune as well as to generate the response.

Model	Finetuning Time (Minutes)	Max Reserved GPU Memory (GB)
Google Gemma 3 4B	54.95	9.732
LLaMA 3.1 8B Instruct	83.79	8.145
Mistral Small 3 (2501) Instruct 24B (8bit)	225.56	21.541

Table 8: Resource Consumption and Training Time for Fine-Tuning Models

Model	Generation Time (s)	Max GPU Memory Allocated (MB)	Max Reserved GPU Memory (MB)
Google Gemma 3 4B	14.13	8278.29	8314.00
LLaMA 3.1 8B Instruct	11.21	15324.63	15466.00
Mistral Small 3 (2501) Instruct 24B (8bit)	37.64	24371.35	24648.00

Table 9: Resource Consumption and Generation Time for Fine-Tuning Answer Generation Experiments

Table 8 shows the time consumed by each model for fine-tuning. As expected, Gemma 3 consisting of the least number of parameters, finetuned the fastest out of all three models. Surprisingly, LLaMA 3.1 consumed the least amount of GPU memory for the fine-tuning stage while having twice the number of parameters of Gemma 3.

Table 9 reports the resource consumed and generation time for each model at the experimentation stage. Like RAG, the longest question was selected as the benchmark. The results are identical to those of RAG, Gemma demonstrates the highest efficiency in GPU memory consumption, while LLaMA excels at generating the fastest output.

6.3 Discussion

In this section, we analyze, compare and interpret the results of both RAG system and Finetuned models reported in section 6.1 and 6.2. We, evidently, not only present the evaluation metrics but also evaluate the computation resources consumed by both the approaches. This dual comparison will assist us to explore the trade-offs between models and techniques to assist us proposing a suitable approach given the resource constraint nature of real word applications. Finally, we recommend an approach that can be explored further in the future for optimization of LLM-based domain experts.

6.3.1 Performance Comparison

The RAG-based system significantly outperform the finetuned models across all evaluation metrics. Among the RAG architecture, Mistral 24B 8-bit Quantized delivers the best performance overall, closely followed by Gemma 3 4B. Finetuned models, on the other hand, lack consistency in generation similar tokens as the original text, which is a critical issue for domain expert systems. This indicates the a dynamic retrieval system is able to provide context grounding, allowing the RAG models to generate relevant jargon-full text, emerging as the better technique out of the study.

Despite the lowest scores in BLEU and ROUGE metrics, Finetuned models are able to generate semantically similar text, exhibiting ability to retain information. Even though not producing technical answers, the generated text is meaningful when compared to the original text. The results indicate that general purpose model might require rigorous finetuning stage to improve its lexical precision. While these model under perform quantitatively, there seem to be encouraging signs in generating semantically similar text.

RAG systems perform better primarily due to their ability to access the integrate the knowledge dynamically. The retrieval mechanism allows it to access the relevant information from the source itself instead of incorporating knowledge into its weights. Light fine-tuning can restrict the model's performance to a significant extent and over fine-tuning can lead to catastrophic forgetting, finding a right balance is a challenging and time-consuming task itself.

6.3.2 Resource Utilization Comparison

When comparing the time consumption and GPU resources utilized, both RAG and Fine-tuning demonstrate identical performance but there are important points that needs to be considered to determine the better approach, which are highlighted in this section.

Both RAG and fine-tuned models takes almost same time to generate a response despite RAG using an additional retrieval step prior to answer generation, subsequently not effecting the latency of the system. Potentially, retriever is operating with efficiency, and the retrieval time appears to be negligible in most scenarios, indicating that it is not a bottleneck. However, finetuning involves an additional cost - the finetuning stage itself - which can be time-consuming depending on the size of dataset and the model architecture. The cost is a one time cost, later to which the model generates responses as quick as RAG, even close to 10% quicker in case of Mistral 24B 8-bit Quantized. It is also worth noting that RAG required creation of a vector embeddings for the retriever, but this process can occur offline on RAM without introduction any time-delays from APIs or GPU on-loading and off-loading as it can be efficiently performed on CPUs depending on the size of embedding model. Given, both approaches require same time investment for answer generation and fine-tuning consisting of an extra time-consuming stage of finetuning itself, RAG becomes the superior technique out of the two.

In terms of GPU resource consumption, similar patterns occurs with models displaying identical usage during generation phase. However, finetuned models appear to be just slightly efficient, likely because of the input prompts being short as RAG architecture appends the context information in the prompt itself at generation time, making them longer and slightly more memory-intensive. As a result, RAG might require more GPU memory per request during inference, although the difference is not significant. As discussed earlier, finetuning stage itself require GPU resources but the usage generally stay within the range of GPU resources required for generation itself. This the approach

feasible enough for most standard GPU setups. On the other hand, RAG does not consist of any GPU overhead. The retriever component can easily run on CPU and RAM - keeping the GPU solely for LLM answer generation. In summary, both methods are GPU friendly with fine-tuning offering slight efficiency due to concise inputs, while RAG does not involve GPU overhead.

Each approach has its pros and cons depending on the specific scenario. However, a broader perspective is necessary, that compares both performance metrics related to generation quality and the resources required to achieve them. This rigorous analysis will enable us to identify the most suitable approach for different use cases in the next section.

6.4 Evaluating Trade-offs between Performance and Resource Consumption for RAG and Finetuning

It is crucial to evaluate the trade-offs between the performance offered by an approach with respect to the resources consumed in real-world deployments, especially when working with domain specific tasks. Resource constraint like GPU memory and inference time are common in production environments, particularly on small devices. Additionally, trade-offs becomes increasingly relevant when compare different types of techniques - RAG or Finetuning - as they have different architectures and computational demands. Understanding this balance will help practitioners and developers deploy solutions that are feasible and efficient, ensuring model is performing up to standards while being within the boundaries of system resources.

As discussed in section 6.3.1 and 6.3.2, RAG outperforms fine-tuned models while using identical resources. RAG's retrieval posses several advantages over finetuning:

1. It can access real-time domain specific information to allow jargon-rich and contextually accurate information.

2. No prior training or finetuning cost is associated, which saves time and energy.

3. Retrieval mechanism can run efficiently on CPU and RAM, reducing the need for any GPU overhead.

Which Approach is the Best?

As mentioned earlier in section 5.2.2, we performed experiments on three different base models for RAG architecture. Notably, Gemma 3 matched the performance of Mistral on most of the evaluation metrics which consuming 3 times less GPU VRAM and producing responses 3 times faster. While LLaMA consumed nearly twice the resource of Gemma and underperformed on almost all metrics except BLEU (see table 5), it is excluded from further consideration. Mistral, while being the most resource intensive model (requiring approx. 24GB of VRAM and nearly 3× the inference time - 40s vs 14s), showed about 30% increase in BLEU score and 5% improvement in ROUGE and BERT Score (R). However, in some cases, it underperformed when compared to Gemma, averaging out its performance. Therefore, if the hardware availability is limited (¡24GB VRAM), Gemma becomes the natural choice, offering equivalent performance with significantly lower resource requirements. On the other hand, if resource constrains is not a problem, Mistral can be preferred - particularly when BLEU improvements are critical in tasks such as

domain specific question answering, where sentence phrasing and terminology matters immensely.

Can Fine-Tuning Be a Viable Option Despite Lower Performance?

While fine-tuned models did not match the performance for the given parameter configuration (see Table 4), they can still offer some value in certain deployment scenarios. Fine-tuning can be a viable option when the dataset is static and unlikely to evolve. In such cases, it can be a good approach as the initial investment - the finetuning stage - is a one-time cost, and inference can be more efficient, saving around 2-3% of VRAM when compared to RAG due to shorter prompts. Furthermore, fine-tuned models are self-contained and do not depend on any external tools like retriever, removing the risk of poor response generation due to poor retrieval quality. Finetuned models contain the information in their weights, resulting in a deterministic behavior and consistent outputs, unlike RAG where results depend upon the retrieved information. Lastly, in settings where external data access is restricted due to privacy, compliance, or latency concerns, fine-tuning offers a secure alternative. A leak of model weights cannot leak the information in the LLM, whereas, data leak of a vector database can result in release of explicit private information.

7 Conclusion and Future Work

7.1 Conclusion

The study reveals that Retrieval Augmented Generation is significantly outperforming Finetuned models while using comparable amount of computational resources and similar inference time. Among the RAG Models, Gemma achieved performance on par with Mistral while the one-third of the resources, making it the most resource-efficient technique in the evaluation.

Finetuning using LoRA demonstrated an ability to retain the semantic meaning, but failed to adapt to the domain specific terminology as reflected by the low BLEU scores. This might imply that while fine-tuning can enhance general comprehension, an attentive or sophisticated strategy can assist the model to capture domain-specific jargon.

7.2 Future Work

The curated dataset in the study lacked inclusion of equations or images, which might be prevalent in the technical documentation. Including multi-modalities or OCR techniques to extract text can help assist the model to generate more accurate answers. Additionally, the fine-tuning experiments performed were intentionally simple to produce a baseline for future potential work, which could include advanced finetuning techniques to improve the overall performance on the domain-specific language while maintaining computational efficiency.

References

Beltagy, I., Lo, K. and Cohan, A. (2019). Scibert: A pretrained language model for scientific text.
URL: *https://arxiv.org/abs/1903.10676*

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. and Amodei, D. (2020). Language models are few-shot learners.
URL: *https://arxiv.org/abs/2005.14165*

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.
URL: *https://arxiv.org/abs/1810.04805*

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L. and Jégou, H. (2024). The faiss library.

Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D. and Smith, N. A. (2020). Don't stop pretraining: Adapt language models to domains and tasks.
URL: *https://arxiv.org/abs/2004.10964*

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M. and Gelly, S. (2019). Parameter-efficient transfer learning for nlp.
URL: *https://arxiv.org/abs/1902.00751*

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L. and Chen, W. (2021). Lora: Low-rank adaptation of large language models.
URL: *https://arxiv.org/abs/2106.09685*

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B. and Liu, T. (2025). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, *ACM Transactions on Information Systems* **43**(2): 1–55.
URL: *http://dx.doi.org/10.1145/3703155*

Jie Li, FuyongZhao, P. J. X. H. L. M. Y. M. (2025). An astronomical question answering dataset for evaluating large language models, *Scientific Data* .
URL: *https://doi.org/10.1038/s41597-025-04613-9*

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D. and tau Yih, W. (2020). Dense passage retrieval for open-domain question answering.
URL: *https://arxiv.org/abs/2004.04906*

Landis, J. R. and Koch, G. G. (1977). The measurement of observer agreement for categorical data, *Biometrics* **33**(1): 159–174.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H. and Kang, J. (2019). Biobert: a pre-trained biomedical language representation model for biomedical text mining, *Bioinformatics* **36**(4): 1234–1240.
URL: *http://dx.doi.org/10.1093/bioinformatics/btz682*

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V. and Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.
URL: *https://arxiv.org/abs/1910.13461*

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S. and Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks.
URL: *https://arxiv.org/abs/2005.11401*

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries, *Tech Report WS-04-06*, SRI International, Barcelona, Spain. Presented at the ACL-04 Workshop on Text Summarization Branches Out.

OpenAI (2024). Gpt-4o, `https://openai.com/index/gpt-4o`. Accessed: 2025-07-28.

Papineni, K., Roukos, S., Ward, T. and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, ACL, pp. 311–318.

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

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W. and Liu, P. J. (2023). Exploring the limits of transfer learning with a unified text-to-text transformer.
URL: *https://arxiv.org/abs/1910.10683*

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks.
URL: *https://arxiv.org/abs/1908.10084*

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P. and Hashimoto, T. B. (2023). Stanford alpaca: An instruction-following llama model, `https://github.com/tatsu-lab/stanford_alpaca`.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E. and Lample, G. (2023). Llama: Open and efficient foundation language models.
URL: *https://arxiv.org/abs/2302.13971*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. (2023). Attention is all you need.
URL: *https://arxiv.org/abs/1706.03762*

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N. and Zhou, M. (2020). Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers.
URL: *https://arxiv.org/abs/2002.10957*

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q. and Artzi, Y. (2020). Bertscore: Evaluating text generation with bert, *International Conference on Learning Representations (ICLR)*.
URL: *https://openreview.net/forum?id=SkeHuCVFDr*