AMERICAN UNIVERSITY OF BEIRUT

Project 53 - Spring Semester: The CommGPT

by

David Abboud, CCE
Alexander Menassa, CCE
Alex Eid, CCE

Main Advisor: Dr. Jihad Fahs

Co-Advisors: Dr. Ibrahim Abou Faycal and Dr. Fadi Zaraket

Sponsored by: Electrical and Computer Engineering Department - AUB

A final year project
submitted in partial fulfillment of the requirements
for the degree of Bachelor of Engineering
to the Department of Electrical and Computer Engineering
of the Maroun Semaan Faculty of Engineering and Architecture (MSFEA)
at the American University of Beirut
Beirut, Lebanon

Acknowledgements

We are deeply grateful to have benefitted from the extensive knowledge and expertise of Dr. Jihad Fahs, Dr. Fadi Zaraket, and Dr. Ibrahim Abou Faycal. In particular, Dr. Fahs and Dr. Abou Faycal have been exceptional mentors, and we sincerely appreciate the communication data they generously shared with us. Special thanks also to Dr. Zaraket for his valuable insights in Machine Learning and Natural Language Processing.

We extend our gratitude to Miss Sally Choker for her continuous support and insightful advice regarding Large Language Models. Her guidance and patience have been invaluable.

Finally, we sincerely thank the Department of Electrical and Computer Engineering for their unwavering support and generous funding, without which this project would not have been possible.

Executive Summary

The CommGPT project attempts to create an AI teaching assistant that supports students in EECE 442, Communications Systems. Its motivation originates from the need of specialized chatbots that can provide guidance within technical fields like Communication Systems, where detailed, domain-specific knowledge is essential. Existing solutions lack the specificity and interactivity required for higher-level technical education, which is what drives this project. The design aims to provide responses that guide students towards solutions, mirroring the role of a human teaching assistant.

Several constraints impact the project, including computational limitations due to reliance on limited resources and technicalities, data security, sufficient accuracy, clarity and coherence of responses, and educational responsibility. These constraints guide the design and implementation choices to ensure that the system meets technical, qualitative, ethical, and educational standards.

The proposed solution involves a multistage system. It integrates retrieval-augmented generation (RAG) fusing the retrieval of open source documents based on cosine similarity between the query and specially created summaries of them with retrieved proprietary documents form another database. The documents are then reranked using a crossencoder, and passed to an LLM that was fine-tuned successively on easy, medium, and hard Q&A with our generated and curated domain-specific data. We developed a graphical user interface (GUI) using Gradio, integrating conversation history and a customizable system prompt.

Additionally, we constructed a dataset of 450 multiple-choice questions tailored specifically to Communication Systems for evaluating large language models (LLMs). Our final RAG and Fine tuning setup with Qwen 2.5 3B model successfully achieved 61.33 % score compared with a score of 57.11 % with the base model. As for mathsral 7B, our RAG setup achieved a score of 70.22 % vs a score of 58.22 % with the base model.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The incorporation of Artificial Intelligence (AI) in education (AIEd) has recently gained prominence as a key research field [2]. Today, learners of all ages and backgrounds are gravitating toward online chatbots to seek help in understanding technical, scientific, and engineering disciplines. They use these bots in various ways, including seeking guidance, solving exercises, double-checking answers, and clarifying concepts. However, these tools often lack domain-specific knowledge rich enough to help students accurately and effectively. In addition, commercially available chatbots often fail to consider the full complexity of a human teaching assistant's role, which requires a careful understanding of student learning styles as well as the exact degree and type of help needed [2].

Inspired by the above, we find that there is a growing need for a digital AI teaching assistant (TA) tool to support students in their educational experiences at the senior undergraduate engineering level. To effectively address this need, we begin by selecting a specific, advanced-level engineering course as an initial use-case, and developing a comprehensive pipeline from data collection and preprocessing to system design specifically for that course. We do so using a combination of techniques and insights gained from relevant literature and state-of-the-art practices.

Several previous works have developed similar tools using Large Language Models (LLMs). One notable work is last year's Final Year Project, *"My name is EECE-BOT and I will be your instructor"*, which focused on fine-tuning models by training them on data derived from relevant course material.

In our project, CommGPT, we will focus our attention on the course *EECE 442: Communication Systems*. Such a targeted implementation enables careful tuning to closely match the intricacies and pedagogical requirements of the chosen subject, but can easily be modified for any other subject or course. Due to the success of CommGPT in the challenging and complex setting of AUB's Communication Systems course, we have demonstrated its significant potential for adaptability and effectiveness in simpler or less specialized courses, highlighting the strength and broader applicability of our data preparation, system design,

and implementation across diverse educational scenarios.

1.1 Desired Needs

Students in the Senior undergraduate engineering course *EECE 442: Communication Systems* require a personalized digital teaching assistant to support their learning. They need a tool that specializes in the course material while also functioning effectively in the role of a teaching assistant. Also, the developed pipeline, including the data gathering process, processing, splitting, storage, the tools used and hyperparameter configuration, as well as the exact system design need to be easily generalizable and adaptable to any other course or topic.

1.2 Requirement Specifications

To ensure that the system aligns with both technical and educational goals, the following requirements must be met:

- **Response Time:** The system should provide responses within a reasonable time to maintain an interactive, real-time user experience for students.
- **Reliability:** The system must deliver consistent, reliable responses that maintain high accuracy across a range of queries, avoiding disruptions or frequent re-initializations.
- **Modular Design:** Each component, including external tools and datasets, must be modular, allowing for easy extension, replacement, or updating of individual parts without affecting the overall functionality. This is especially relevant as technological advances in this field are frequent. Also, since it is proven that as we scale the large language model used, an improvement in the quality and richness of the responses is observed and provable mathematically [3].
- **Educational Guidance Focus:** The system must be designed to guide students toward the answer rather than providing the answer directly. This is essential to foster a learning experience similar to that of a human teaching assistant.
- **Scalability:** The solution should be scalable to support a growing number of users.
- **Adaptability:** Our solution must be adaptable and extendable, allowing our methodology to be used with minor adjustments to create digital teaching assistants for other advanced engineering or scientific courses.

1.3 Deliverables

The final deliverable is a functional LLM Chatbot teaching assistant tool for students in the telecommunications course, designed with an interactive user interface and integrated with course-specific support.

- **Completed Teaching Assistant Model:** A fully functional, fine-tuned LLM teaching assistant capable of responding to student inquiries within the scope of EECE 442 topics
- **Interactive User Interface (UI):** A simple, accessible UI that allows students to ask questions and receive responses in real time.
- **Evaluation Report and Metrics:** A detailed report showcasing model performance in several different configurations, accuracy rates, and the quality of the domain lexicon grasp of the model.
- **Scalable Knowledge Base with Vector Embeddings:** A vector-based knowledge repository embedded with course material, capable of returning relevant content in response to queries. This repository will be easily updateable to reflect future changes in course content.
- **Documentation and User Manual:** Comprehensive documentation detailing system architecture, model training processes, integration instructions, and deployment guidelines.
- **Code-base for Extension and Maintenance:** An organized code repository that allows future developers or students to extend, improve, or update the model, UI, and RAG components based on evolving course requirements or technological advancements.
- **MCQ Benchmark:** A comprehensive benchmark composed of a multitude of MCQ questions that can be used to test the strengh of any LLM in Communication Systems knowledge. These questions range over the entire course topics, and include generated, manually written, extracted, and compiled questions.
- **Communication Systems Course Specific Dataset:** A dataset comprising of organized full markdown files as well as json files encompassing the entire chapters of the course, gathered from various open source online sources, books, and content. The dataset also includes metadata labelled chunks that contain many fields fit for very advanced Retrieval Augmented Generation (RAG). Lastly, it includes thousands of easy, medium, and hard Q&A that are suited for fine tuning, made in a format that can be easily adapted for any LLM.

1.4 Related Work

The development of CommGPT was informed by several prior initiatives that explored the integration of LLMs into domain-specific applications, particularly within telecommunications and education. It also was inspired by several important techniques and tools presented in various research papers. This section provides an overview of the above, highlighting their methodologies, contributions, and relevance to our work.

1.4.1 EECE-BOT as a Teaching Assistant

Our initial inspiration came from "My name is EECE-BOT and I will be your instructor," which was undertaken by a team of senior undergraduate students at the American University of Beirut during the 2023–2024 academic year. The project's objective was to develop an AI teaching assistant tailored for the EECE 442 course, Communication Systems. The team employed LLMs such as Flan-T5, Gemma-2.5B, and Gemma7BQ, selected based on the computational resources available through AUB's High-Performance Computing system. To optimize performance within these constraints, techniques like quantization, data sharding, and the use of DeepSpeed were implemented. Course Data was directly categorized into Q&A and factual segments then used in training.

While the project explored the feasibility of integrating LLMs into educational tools, it faced limitations due to restricted computational capacity, as well as minimal data processing and treatment. Nonetheless, the project provided valuable insights into effective techniques and highlighted areas for improvement, laying the groundwork for CommGPT. Specifically, we were inspired by it to use the HPC, and to select our model sizes. Also, we began by quantizing our models also as well as using deepspeed, but later moved on to full fine tuning with Unsloth.

1.4.2 TeleLLM: Domain-Specific LLMs for Telecommunications

Developed by engineers from Yale and AWS Amazon, the TeleLLM Series project introduced a suite of LLMs specifically tailored for the telecommunications domain, with model sizes ranging from 1B to 8B parameters [4]. To support this initiative, two open-source datasets were created: Tele-Data, comprising telecom-specific materials from sources like arXiv and 3GPP standards, and Tele-Eval, a question-answer dataset designed to evaluate telecom-related tasks.

The project explored both parameter-efficient fine-tuning (PEFT) and full fine-tuning (FFT) techniques, employing models such as Tinyllama-1.1B, Phi-1.5, Gemma-2B, and Llama-3-8B. To mitigate the issue of "catastrophic forgetting," a mixed dataset combining general-purpose and telecom-specific data was

used, allowing the models to retain general language capabilities while enhancing domain-specific performance.

TeleLLM's approach demonstrated that with appropriate data and fine-tuning techniques, domain-specific LLMs could outperform general-purpose models in specialized fields like telecommunications.

We were inspired by the supervised fine tuning (SFT) performed by the TeleLLM teams. Also, we began in the fall extensively understanding their strategy as well as all the technical approaches in their work, due to its relevance to our end goal. This has tremendously helped us in getting better at fine tuning. The difference between what TeleLLM and CommGPT is that CommGPT is specific to the material of the Communications Systems course, whereas TeleLLM is specialized to all of Telecommunications in general, which is dozens of courses. Hence, we expect CommGPT to be better at the specialized course than TeleLLM. Also, we first were exploring the use of the TELEEVAL as a means to evaluate our model, but later realized the complexity involved in judging the quality of LLM responses on Open-ended questions. Metrics like Rouge scores are not easily interpretable, unlike Q&As which is what we ended up doing.

1.4.3 TeleQnA: Benchmarking Telecommunications Knowledge

TeleQnA, a benchmark dataset designed to assess the telecommunications knowledge of LLMs, was also developed by the same team of TeleLLM [5]. TeleQnA comprises 10,000 multiple-choice questions distributed across five categories: Lexicon, Research Overview, Research Publications, Standards Overview, and Standards Specifications. Each question includes options, the correct answer, an explanation, and a category label, facilitating comprehensive evaluation. The explanation is never actually used, but was added as a way to debug wrong questions, as well as a way to force chain of thought reasoning in the generation of these questions.

In our project, we found the Lexicon category of TeleQnA particularly useful, as it encompasses questions focusing on general telecom terminology and definitions. We incorporated 75 of these lexicon questions into our benchmark to evaluate CommGPT's grasp of foundational telecommunications concepts. Furthermore, we structured our benchmark in a similar format to TeleQnA, ensuring consistency and enabling comparative analysis of model performance.

1.4.4 Other Telecommunications LLM Projects

OptiComm-GPT, developed by engineers from Qingdao University, China, focused on creating a specialized LLM for optical fiber communication tasks [6]. The project integrated Retrieval-Augmented Generation (RAG) within the LangChain

framework, enabling the model to retrieve relevant information from a structured database to enhance its responses. Prompt engineering techniques, including chain-of-thought prompting, were employed to improve the model's ability to decompose and address multi-step problems. All of the above made it into our final design. Nonetheless, the use of private-sourced datasets restricted our ability to benefit from this project.

We also investigated the TelecomGPT project, undertaken by engineers from Khalifa University, Abu Dhabi, UAE, aimed to address the limitations of general-purpose LLMs in understanding telecom-specific terminologies and tasks [7], which further demonstrated the effectiveness of fine-tuning strategies. However, the model's reliance on pre-defined benchmarks and DPO for alignment may hinder its adaptability to entirely new tasks outside its existing framework.

1.4.5 Other Tools

Some tools inspired our ideas, such as Autogen [8] which allows the use of multiple agents that collaborate in a Group-Chat system to provide desired outputs. AutoGen, developed by Microsoft Research, is an open-source framework that introduces customizable and conversable agents, enabling developers to create agents with different roles by configuring subsets of built-in capabilities, including LLM-backed, human-backed, and tool-backed functionalities. In the final version, we decided against the use of Autogen as we focus on creating a single LLM Chatbot instead of a multi-Agent system. Also, even in future versions of CommGPT, we found that that it is better to create one's own Multi-Agent System without Autogen as that would give more low level control over the implementation details and prompting of each agent, helping prevent creativity and hallucination in a strictly scientific engineering topic.

Furthermore, as we learned more about RAG, the new RAFT [9] concept, which is proposed to be used as a way to fine tune a model in order to enhance its use of RAG, was considered for our implementation. By incorporating elements from RAFT, our system could have potentially become more robust in producing grounded answers by providing the LLM with a more natural integration of RAG. Although, we considered Fine Tuning our model with RAFT, time constraints prevented us from doing do so as we focused on the fundamentals of our system. Potentially, RAFT can be considered as a means for improving the system.

Moving to the internal tools, FAISS [10] was another critical component adopted in our system for fast vector-based search. With FAISS, our system could fetch relevant documents and contextual snippets for user queries in milliseconds, making it a vital piece in the user experience pipeline.

Lastly, Unsloth [11] is a fine tuning library for LLMs that significantly improved the performance and memory efficiency of training smaller custom model. Despite DeepSpeed [12] being our primary choice at first due to its more wide adoption including in the EECE-BOT, we found that Unsloth gives a much more granular customization of all fine tuning hyperparameters. Not only that, but it is also more transparent in showing exactly how the configuration is made, and even is up to 2x faster and more memory efficient using clever mathematical tricks even for full fine tuning, which was only just added to it in March 2025. Drawing from Unsloth's performance benchmarks, we were able to run fine-tuning processes on limited hardware without compromising on quality.

1.4.6 Models

In our exploration of suitable models for CommGPT, we evaluated several state-of-the-art language models to address various aspects of our system, including embedding, retrieval, and response generation. For embedding and retrieval tasks, we found that the Stella en 400M v5 model [13] was the best. This model is designed for semantic textual similarity and text retrieval tasks, offering multiple dimensionalities (e.g., 512, 768, 1024, up to 8192) to balance performance and computational efficiency. Its architecture incorporates a fused Matryoshka layer, which reduces computational overhead while maintaining high relevance metrics.

To have enhance mathematical reasoning capabilities, we integrated Mathstral-7B [14], a specialized variant of Mistral 7B fine-tuned for mathematical and scientific tasks. The Mistral-7B model is known for its efficiency and high performance in real-world applications, employing grouped-query attention (GQA) and sliding window attention (SWA) mechanisms which helps it to handle longer sequences effectively while reducing inference costs [15]. Mathstral 7B features a 32k token context window and has demonstrated state-of-the-art performance in mathematical reasoning benchmarks.

Additionally, we incorporated the Qwen 2.5 Series [16], particularly the Qwen 2.5-3B model, which is part of a family of decoder-only language models ranging from 0.5B to 72B parameters. Qwen 2.5 models have been pretrained on large-scale datasets encompassing up to 18 trillion tokens and exhibit significant improvements in instruction following, long-text generation, and structured data understanding. The Qwen 2.5-3B model responded positively to fine-tuning, effectively assimilating our domain-specific data and playing a central role in our system's performance. It was easier to fine tune than Mathstral as more information was more readily available on that.

In summary, the literature reviewed in EECE 501 as well as in the start of EECE 502 provided both theoretical insight and practical tools that were instrumental in shaping this project. Each paper contributed uniquely in design, implementation, or evaluation.

Chapter 2

Design Process

2.1 Technical/Non-Technical Constraints

2.1.1 Technical Constraints

The proposed LLM-based teaching assistant encounters several technical constraints, primarily due to the computational demands of large language models, data handling limitations, and integration requirements. These constraints impact model performance, scalability, and deployment feasibility.

Computational Resources

The fine-tuning, inference, and deployment of large language models require substantial computational resources, which were constrained in the Fall term by the available High-Performance Computing (HPC) infrastructure at the American University of Beirut, hence the Nvidia V100, with 32 GB of VRAM. Thankfully, in the spring, the constraint became the budget of $600 granted to us by the ECE department, which we used to rent GPU Pod machines on Runpod. Primarily, we used Nvidia A100 GPUs, equipped with 80GB of VRAM each. The system must operate within the memory and processing power limits provided by the A100 GPU, which eventhough superior to the V100 of the HPC, still restricts the size and complexity of models that can be fine tuned on it, as well as those that can be used on it for inference. Resource limitations would also limit the number of simultaneous users or interactions that the system can handle effectively (in the final system, this only ends up happening at a massive scale where more than 100 users are using CommGPT simultaneously on the A100).

Response Time and Latency

As a responsive educational tool, the teaching assistant must maintain a low response time to foster an interactive experience for students. Achieving the target

response time that will not hinder the study flow of users may be challenging due to the model's computational complexity, potential latency in retrieving and embedding context data via Retrieval-Augmented Generation (RAG), and integration with potential external tools such as Wolfram Alpha. Optimizing each component to minimize latency while ensuring response accuracy remains a significant constraint. In our final system, the latency is between 10 and 20 seconds on average, depending on response length. Given the other constraints, it was difficult to decrease it further, but more work is encouraged in this regard.

Data Processing and Storage

The system's vector database, used for RAG, requires substantial storage capacity for the embedded course data, and updates to this data must be managed efficiently. Embedding and querying large datasets are memory-intensive processes, and frequent updates to the database, reflecting new course materials or changes, could introduce additional computational strain. Efficient data management and storage strategies are essential to maintain the system's responsiveness and relevance. We ended up using the FAISS database, which offers methods that efficiently take care of queries and updates.

Scalability and Extensibility

While the system is designed with modular components to facilitate future expansions, scalability may be limited by current model architecture and database size constraints. Handling a growing number of concurrent users, increasing the knowledge base, or adding new courses would require additional computational resources and storage, potentially exceeding current hardware capabilities.

2.1.2 Non-Technical Constraints

Beyond technical limitations, several non-technical constraints impact the development and deployment of the proposed LLM-based teaching assistant. These considerations include the system's long-term sustainability, social impact, and adherence to health and safety standards. By addressing these constraints, the system can better serve its educational purpose while aligning with broader ethical and societal values.

Content Integrity

It is crucial to ensure that the model's responses adhere to ethical guidelines, particularly to avoid misinformation or misleading advice. Responses should be monitored to prevent the dissemination of incorrect or harmful information, particularly in fields that involve technical calculations or advice. An additional safety consideration includes ensuring that the model's outputs respect privacy

and prevent the generation of biased or inappropriate responses that could harm users emotionally or academically. While our current system prompt tries as much as possible to ensure this, reinforcement learning with human feedback (RLHF) is required to ensure that ethics is better infused into the model. [17]

Educational Responsibility

Given that this system will be used in an academic setting, the assistant is intended to aid learning, not to replace it, and must be configured to provide guidance rather than direct answers. This approach encourages students to engage in critical thinking, aligning with educational best practices. In other words, the system we are designing should not provide the students with the answer to the problems they need help with, rather respond as a Teacher Assistant in guiding the students towards the solution in a step by step or question answer manner. In order to achieve this to a greater degree, finetuning the Chatbot on full conversations between a TA and a student on the designated topics might be helpful.

Data Security

Since a portion of the data used to train the model is provided by the Professors who own the data, the system should protect itself from data leaks, in order not to allow answers to assignments and exams to be dispersed to the users. Furthermore, some of the data used might be copyrighted by the authors who authorized its use in the training of the model, thus protecting it from leaks might endanger its misuse. In this light, we make sure to tell the bot in the prompt to not explicitly refer to any of the retrieved chunks, but only to use them. RLHF is required for even stronger protection.

2.2 Design Alternatives

2.2.1 Full System Alternatives

A Union of Designs

This design alternative combines elements from the previous designs to create a robust and adaptable solution. It involves multiple specialized LLM agents working in unison. A central Conversable Agent coordinates interactions, managing input preprocessing, response generation, and refinement. Auxiliary Agents include a fine-tuned LLM for domain-specific expertise and a Retrieval-Augmented Generation (RAG) module that retrieves relevant data from a vector database embedded with structured course materials. This ensures responses are precise, contextually relevant, and aligned with requirements. To enhance flexibility and

functionality, the system incorporates the AutoGen framework, which facilitates multi-agent collaboration by allowing agents to perform distinct tasks, such as retrieving, augmenting, and validating data. This enables dynamic task distribution and efficient handling of complex queries. Fine-tuning and instruction tuning are employed to optimize the LLM's responses for clarity, relevance, and guidance, while external Tool Agents are integrated for handling computations or advanced operations when necessary.

Pipelined System with Progressive Stages

This design alternative involves a step-by-step process that begins with the input from the student and ends with the response. The process begins with an optional query preprocessing, where the student's question is refined by correcting typos, rephrasing ambiguities, and generating similar variations to better capture intent. Using a RAG, a Summary based Retriever that searches a vector database to identify the most relevant course materials, ensuring the data retrieved is precise and domain-specific. These documents are then passed to an reranker, which uses a crossencoder to rerank the chunks. Then, a prompt structures the information into a format optimized for the Target LLM. The Target LLM is guided by a customized system prompt that provides detailed course context and incorporates the chatbot's interaction history to maintain continuity in the conversation. This input structure enables the LLM to produce accurate, tailored, and context-aware responses. An optional refinement step allows an agent to enhance the LLM's output, improving clarity and user-friendliness. Finally, the response is delivered to the student.

2.2.2 Fine Tuning Alternatives

Fine-Tuning using LoRA and QLoRA

In this design instead of training the entire model and altering all its weights, we train only a subset of the weights. Thus, this technique is faster, though it still requires a GPU. We experimented with this on the HPC. Note that the difference between QLoRA and LoRA is that QLoRA is LoRA but with a quantized model.

RAFT

As we learned more about RAG, the new RAFT [9] concept, which is proposed to be used as a way to fine tune a model in order to enhance its use of RAG, was considered for our implementation. By incorporating elements from RAFT, our system could have potentially become more robust in producing grounded answers by providing the LLM with a more natural integration of RAG.

Full Fine Tuning

This design alternative was the first trial we experimented with on generic data in the fall. This solution involves fine-tuning the entire weights of the model to specialize it entirely on the topic of interest. This is done by feeding it data specific to the topic and training it further in this subject. While this technique can lead to the most accurate results, it is very resource-intensive. Thankfully, we received funding in April to implement this Option for Fine tuning.

2.2.3 Choice of Generative Model

To perform RAG or Fine tuning, the choice of large language model is an important one. The LLM must effectively integrate retrieved knowledge, provide accurate and contextually appropriate responses, and support seamless integration into the RAG pipeline. Additionally, the model must align with specific constraints such as computational feasibility, domain expertise, and cost efficiency. Based on these constraints and requirements, we require a model with the following criteria:

- **Integration and Compatibility** The model should be open-source and readily available on Hugging Face to ensure transparency and ease of access. It must support seamless integration with vector databases and LangChain, providing robust endpoints for efficient deployment in RAG pipelines. Availability on Hugging Face is crucial for streamlined workflows and compatibility with existing tools.

- **Customization Capabilities** The model must support fine-tuning with proprietary or custom datasets and enable prompt engineering for specific use cases.

- **Domain Expertise** The model must demonstrate knowledge and understanding of advanced telecommunications terminology and concepts. It should thus have had prior exposure to telecommunications data to enhance domain relevance and ensure accurate contextual understanding. Note that this information is not always made public; even when an LLM is open source, its training data need not be. In such cases, we make assumptions based on how the training process of the model is described. In the end, the best way of knowing for sure is to conduct our own benchmark on the topic.

- **Cost Efficiency** The model must operate efficiently, avoiding excessive infrastructure or operational costs while delivering high performance.

- **Low Latency** The model must respond in real-time or within an acceptable latency to support interactive applications and deployments.

- **Contextual Dependence** The model must effectively utilize retrieved context and follow instructions without conflating it with potentially outdated internal knowledge, ensuring accurate and relevant outputs.

- **Support and Community** The model must be supported by comprehensive documentation and an active user community to facilitate troubleshooting, implementation, and collaboration.

- **Benchmark Performance** The model must achieve competitive results on standard benchmarks related to science and mathematics. Performance on telecommunications-specific benchmarks is highly desirable.

- **Size Variability** It is preferred if the model is available in multiple sizes. This allows initial testing locally on smaller versions and later scaling to larger versions for deployment on the HPC system.

- **Context Size** The model must support a medium to large context size, ideally accommodating at least 8,000 tokens based on our readings. This is crucial for RAG workflows to handle long documents or interactions and is particularly important for eventual chatbot deployment to ensure it can process and maintain extended user interactions.

- **Parameter Size and Deployment Feasibility** The model must not exceed 8 billion parameters to fit within HPC constraints or runpod A100 constraints. While larger models may require distributed training or quantization, smaller models should support single GPU deployment for fine-tuning and testing.

After conducting a review of available models, we have selected the following three models: **Llama 3.1 8B**, **LLama-3-8B-Tele**, and **Mathstral 7B**. Below is an evaluation of each model based on our specified criteria and their distinctive features.

Llama 3.1 8B The Llama 3.1 8B model, developed by Meta, is an 8-billion-parameter auto-regressive language model with an optimized transformer architecture [15]. It offers a substantial context length of up to 128,000 tokens, facilitating the processing of extensive documents and complex interactions, which is advantageous for RAG workflows and chatbot applications. Designed for efficient performance, Llama 3.2 8B balances computational requirements with operational costs, making it suitable for deployment on HPC. Its open-source nature and availability on platforms like Hugging Face ensure transparency and ease of integration into existing pipelines. However, it lacks specialized training in telecommunications, which may limit its effectiveness in domain-specific applications. Despite this, the benchmark of its performance in reasoning and math

done by META and found on the huggingface page, as well as its superiority to other similar sized models in math, physics, and engineering according to the benchmark of GAIR NLP group hints at its potential in communications [18]. It also comes in both a base and instruct option, which allows us to later fine tune.

LLama-3-8B-Tele The LLama-3-8B-Tele model is an 8-billion-parameter LLM specialized in telecommunications [19]. It is based on Meta's Llama 3 8B and has been continually pretrained on Tele-Data, a comprehensive dataset comprising approximately 2.5 billion tokens of telecommunications material, including articles, standards, and related web content. This specialized training enables the model to outperform the base Llama 3 8B on telecommunications benchmarks such as Tele-Eval. It is available in both a base and instruct format. The model supports a context length of up to 8,192 tokens, suitable for processing substantial documents and facilitating chatbot interactions. As an open-source model accessible on Hugging Face, it allows for fine-tuning with proprietary datasets and prompt engineering, ensuring adaptability to specific use cases. While its parameter size aligns with HPC constraints, distributed training or quantization may be necessary for fine-tuning.

Mathstral 7B Developed by Mistral AI, Mathstral 7B is a 7-billion-parameter model specializing in mathematical reasoning and scientific tasks [14]. Built upon the Mistral 7B foundation, it has been fine-tuned to tackle complex calculations and problem-solving. Mathstral 7B achieves impressive results on industry-standard benchmarks, scoring 56.6% on the MATH benchmark and 63.47% on the MMLU benchmark. Thus, we find it to be a good contender for telecommunications. It supports a context length of up to 32,000 tokens, advantageous for processing extensive documents and facilitating chatbot interactions. The model is open-source and accessible on Hugging Face, promoting transparency and flexibility. With its 7 billion parameters, Mathstral 7B is well-suited for deployment on HPC systems.

Qwen 2.5 The Qwen 2.5 Series, developed by Alibaba Group, includes a family of decoder-only language models ranging from 0.5B to 72B parameters [16]. For our implementation, we focused on the Qwen 2.5-3B and 7B models due to their balance between performance and computational feasibility. These models are designed with enhanced instruction-following capabilities and improved reasoning performance, particularly on long-form text and structured data. They support a context window of up to 128,000 tokens, making them highly suitable for RAG workflows and extended chatbot interactions.

In addition, the availability of both 3B and 7B variants enables testing on the smaller version and scalability to the 7B model and up to 70B for deployment on high-performance computing (HPC) infrastructure.

2.2.4 Retrieval-Augmented Generation (RAG) Alternatives

RAG involves a LangChain-based retriever that retrieves relevant chunks from a vector database. These chunks are prepared beforehand using the data we gathered. For each question asked by a student, several chunks are used and appended to the query. This is then sent to the LLM to generate an answer.

Keyword Matching with BM25

To capture textual details, we incorporate the BM25 algorithm, a classical probabilistic information retrieval model. BM25 is based on term frequency and inverse document frequency (TF-IDF), which assigns weights to terms in a query and documents. It prioritizes exact keyword matches, making it ideal for retrieving results where specific words or phrases are critical.

Semantic Meaning with Cosine Similarity

To capture semantic meaning, we adopt the approach of using cosine similarity between retrieved chunks and the input query. Cosine similarity measures the angle between two vectors in a multidimensional space, which is particularly useful for comparing dense vector representations of text. By encoding both the query and document chunks into embeddings using a transformer-based model (more on that in next section), the system ensures that semantically similar chunks are ranked higher, even if they do not share identical wording with the query. This method effectively handles paraphrased or conceptually similar queries and documents.

Additionally, we propose a more advanced approach inspired by langchain's parent document retrieval (PDR). The original PDR method ensures that retrieved chunks are contextually linked to the parent document they originate from, preserving the integrity of the document's overall meaning. We propose a similar approach where instead of doing a similarity comparison based on child chunks, we do it based on summary metadata (hence each document has an associated summary field generated by an LLM; this approach is only to be done with open source data). By retrieving and scoring summaries based on their relevance to the query and then extracting relevant full document chunks, this approach minimizes the loss of context that can occur when dealing with isolated fragments of text.

2.2.5 Choice of Embedding Model

The embedding model plays a crucial role in transforming textual data into numerical vectors, which are essential for capturing the semantic meaning and contextual relationships within the text. These embeddings enable the system to perform information retrieval.

TELE-Embedder

Notably, any language model can function as an embedding model by leveraging its attention mechanisms to generate contextualized vector representations of text. In selecting an appropriate embedding model, we prioritized those trained on communication data to ensure rich vector representations of domain-specific concepts. However, our search did not yield any models explicitly trained in this area. Consequently, we create an embedding model using the Gemma-2B model of the TeleLLM series.

Stella400M-v5

An alternative option is the Stella 400M v5 model, which has demonstrated strong performance in retrieval tasks while maintaining computational efficiency. Stella 400M v5 is a 400-million-parameter model trained to produce high-quality text embeddings. It has achieved impressive results on the MTEB, indicating its capability in capturing semantic meanings effectively. The model is optimized for a sequence length of 512 tokens, as longer sequences have shown diminished performance in specialized long-text retrieval datasets. [13].

2.3 Design Decisions

To effectively meet the identified engineering requirements and address both technical and non-technical constraints, we made several strategic design decisions that shaped the final implementation of CommGPT. The most important architectural choice was the adoption of a pipelined system with progressive stages. This design involves a structured, step-by-step process starting from query preprocessing to the delivery of a final response. Each stage in the pipeline—such as chunk retrieval, reranking, prompt construction, model querying, and optional output refinement—is implemented as a modular component. This modularity not only ensures clarity in the overall design but also allows for flexibility in swapping, updating, or optimizing components independently. This decision directly addresses the requirements for modularity, scalability, and maintainability and was a practical solution given our constraints related to computational power, model size, and evolving technologies. Below you can see the final design architecture.

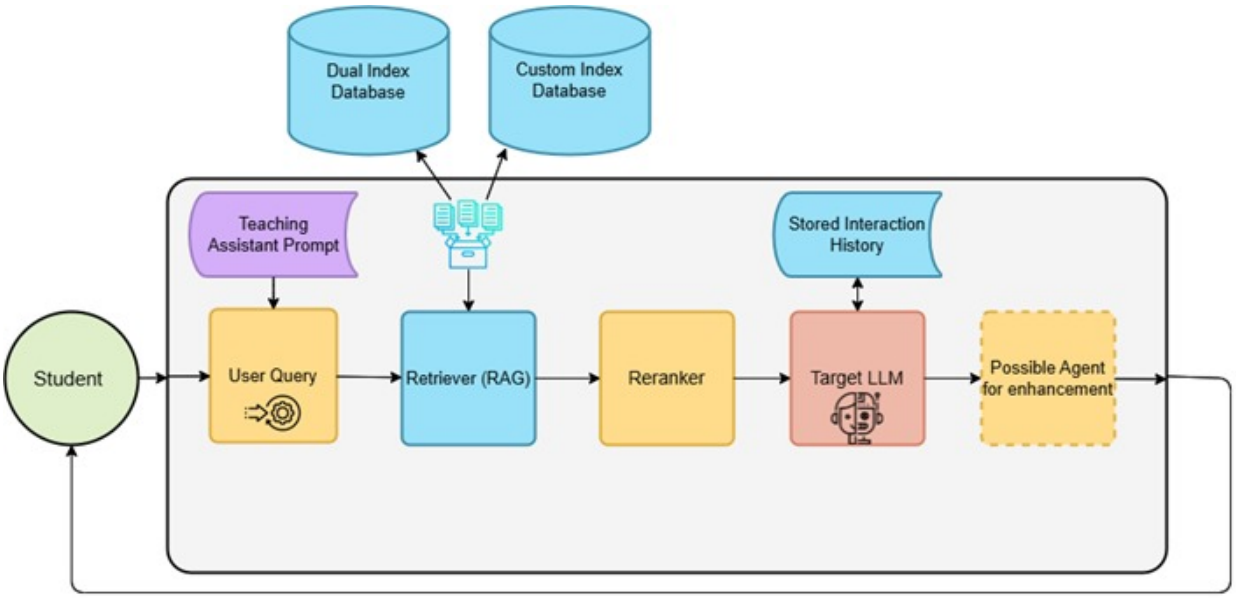


Figure 2.1: Iterations of RAG

For the generative model, we selected two primary LLMs: Qwen 2.5 3B and Mathstral 7B. The Qwen 2.5 3B model, developed by Alibaba, was chosen due to its outstanding performance in instruction-following and its compatibility with our infrastructure, particularly for fine-tuning on a single A100 GPU. Its low weights count and responsiveness to fine-tuning made it ideal for iterative development and deployment. Complementing it, Mathstral 7B was chosen for its strong performance on STEM benchmarks, particularly in mathematics and scientific reasoning. Its expanded context window and mathematical fluency make it well-suited to supporting engineering students through step-by-step explanations and numerical tasks. We validated this choice with an initial test comprising 300 MCQs drawn from the TeleQnA dataset, which helped us gauge each model's understanding of telecommunications-specific content.

Regarding embeddings and retrieval, we compared two options: a domain-specific TELE-Embedder based on Gemma-2B and the Stella 400M v5 model. After benchmarking on our own datasets, Stella was selected due to its compact size, efficiency, and strong semantic retrieval performance. Its performance on the MTEB benchmark, along with its robustness across our own benchmark, made it a reliable choice under our resource constraints. For retrieval infrastructure, we employed FAISS, a high-performance vector database optimized for scalable semantic search. FAISS was configured with dot-product similarity and integrated with LangChain to ensure compatibility with our pipeline.

To enhance the accuracy of retrieved context, we implemented a reranking mechanism using a cross-encoder model, specifically cross-encoder/ms-marco-

MiniLM-L-6-v2 [20]. This model re-evaluates the top retrieved document chunks by computing pairwise relevance scores between the query and each candidate chunk. The chosen cross-encoder is optimized for passage ranking and has been pre-trained on MS MARCO, making it well-suited to fine-grained relevance assessment within our context. This component was critical to improving the quality of responses, especially when handling subtle or ambiguous student queries.

In terms of model adaptation, we selected full fine-tuning as our strategy. This decision was driven by the availability of GPU resources, made possible through the $600 budget granted to us by the ECE department, which we used to rent A100 GPUs on Runpod. Full fine-tuning allowed us to deeply customize the model's behavior on a dataset that included thousands of custom telecommunications Q&A pairs, factual passages, and simulated tutor-student conversations. While alternative methods like LoRA or RAFT were considered, they were excluded due to time constraints and their complexity. RAFT, in particular, though promising in aligning models more effectively with RAG workflows, could not be integrated within our timeline. Ultimately, full fine-tuning gave us the best possible performance and alignment with our educational goals, especially in ensuring the system provides guidance rather than direct answers, simulating the behavior of a real teaching assistant.

2.4 Design Iterations

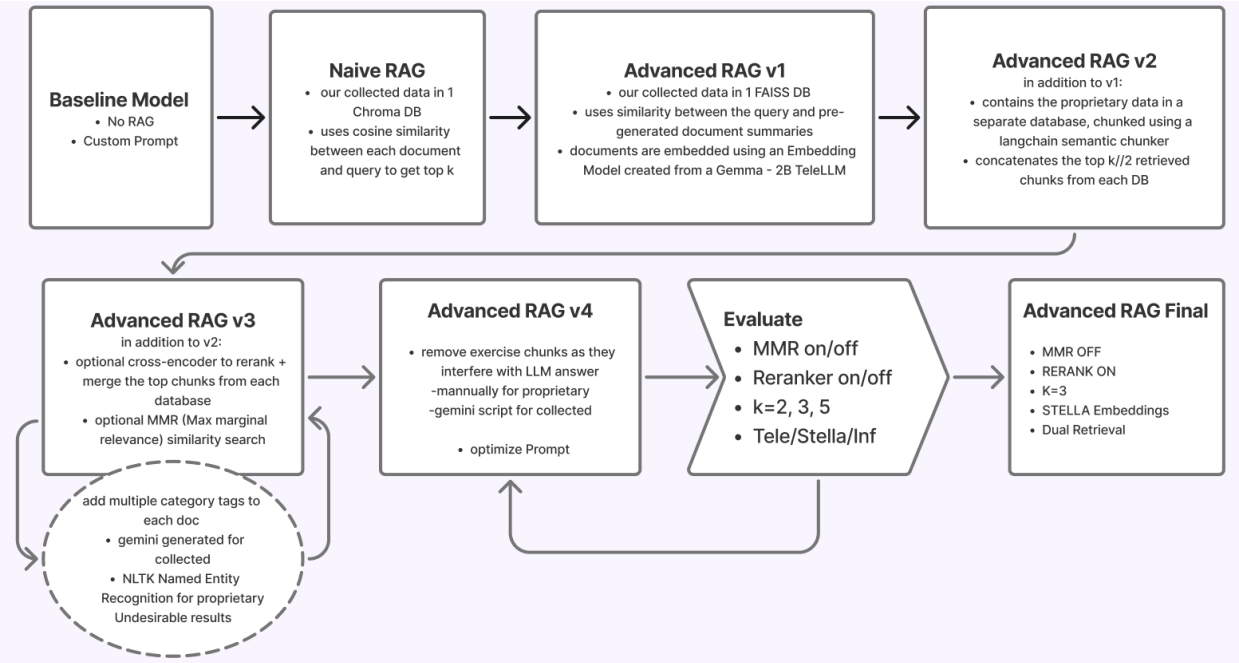


Figure 2.2: Iterations of RAG

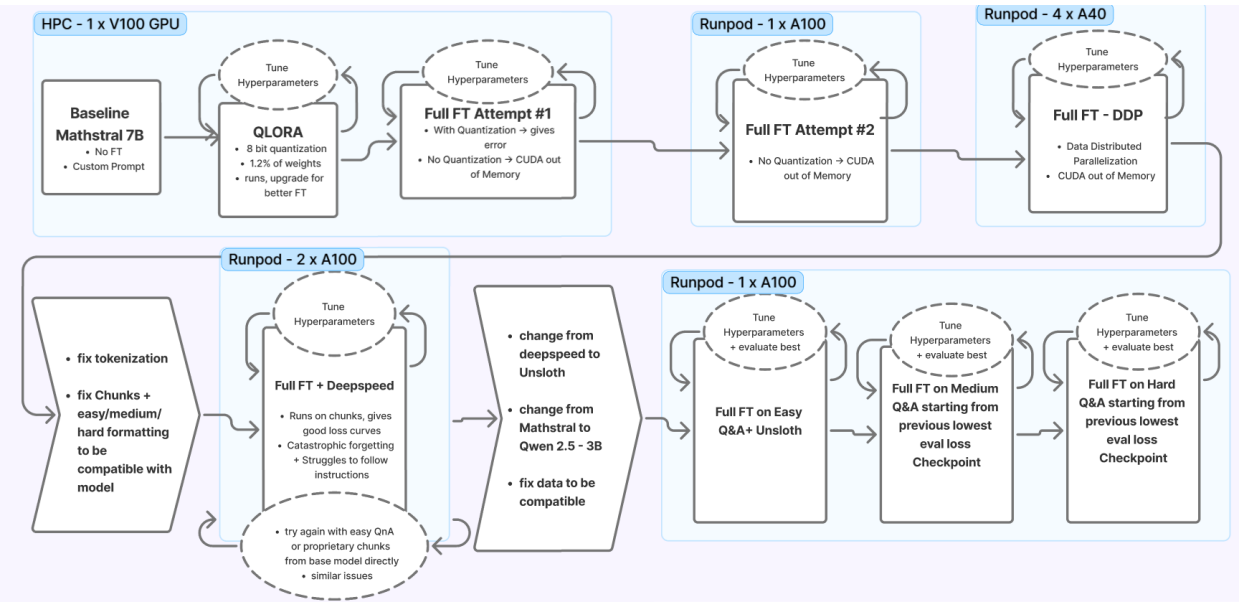


Figure 2.3: Iterations of Fine Tuning

In the above figures, one can see the different iterations we have gone through in EECE 502 in each of RAG and FT. We outline in more details the different design iterations we have gone through in each of RAG, FT, and the UI in the Implementation section.

2.5 Standards

The development of the LLM-based teaching assistant adheres to established international standards to ensure software quality, data security, and pedagogical alignment. These standards have informed decisions throughout the design process, from data handling to system architecture and educational interaction.

2.5.1 ISO/IEC Standards

- **ISO/IEC 25010:2011 – Systems and Software Quality Requirements and Evaluation (SQuaRE)**: defines a comprehensive quality model for software products, encompassing characteristics such as functional suitability, reliability, usability, and performance efficiency. These attributes guided the design of the assistant to ensure it meets the necessary quality level [21].

- **ISO/IEC 27001:2022 – Information Security Management Systems (ISMS)**: specifies requirements for establishing, implementing, maintaining, and continually improving an ISMS. It is tied to the way we handled, shaped and stored our data (for RAG, for fine tuning, as well as user conversations data in Gradio), where we focus on ensuring the confidentiality and integrity of student interactions with the assistant [22].

2.5.2 IEEE Standards

- **IEEE P7002 – Data Privacy Process**: provides guidelines for managing privacy issues in systems that collect personal data. Although our system minimizes the storage of user data, adherence to this standard ensured that any necessary data handling is conducted with utmost care to prevent unauthorized access or breaches [23].

- **IEEE P7004 – Standard for Child and Student Data Governance**: outlines governance practices for managing student data responsibly. Incorporating its principles ensures that the assistant's design respects student privacy [24].

2.5.3 Educational Standards

- **Bloom's Taxonomy for Educational Objectives Framework**: categorizes educational goals to promote higher-order thinking skills. It has been applied to structure the assistant's responses, encouraging students to engage in analysis, evaluation, and creation, rather than mere recall of information. While the current version is not perfect at this, we explain in this report how it can become better at it in more advanced implementations and improvements, especially using Reinforcement Learning with Human Feedback (RLHF) [25].

- **AI for K-12 Initiative Guidelines**: advocate for responsible AI integration in educational settings, emphasizing transparency, fairness, and skill development. They have influenced the assistant's design to ensure it supports problem-solving abilities and aligns with ethical standards in AI education. [26].

Chapter 3

Implementation

This chapter discusses the various stages and aspects of our implementation of CommGPT. The main components of our development include the work done on the Data side, the RAG, the Fine Tuning effort and finally the Graphical Interface deployment.

3.1 The Data

The foundation of any AI system lies in the quality, relevance, and structure of its data. Data is the backbone of our project as it's what will make it the Teacher Assistant in Advanced Communication Systems Engineering. Given the mathematical and conceptual complexity and nuances of the subject, the Chatbot must be capable of understanding the vocabulary, the details of the material, and the mathematical implication in order to service the students - its users - by generating domain-specific information with a high degree of accuracy and relevance.
This section details the multi-stage approach we have undertaken to curate and prepare the data used to fine tune and augment the chatbot. Each stage was carefully designed to ensure that the dataset not only reflects the theoretical and practical aspects of the course but is aligned with the specificities of EECE 442.

3.1.1 Data Collection

In order to effectively Fine-Tune and implement domain-specific Retrieval-Augmented Generation (RAG), it was necessary to gather a large and targeted set of high-quality, relevant data. The "Data Collection" stage was focused on assembling comprehensive educational documents that contain the depth and complexity of the subject all while maintaining instructional goals and academic standards.

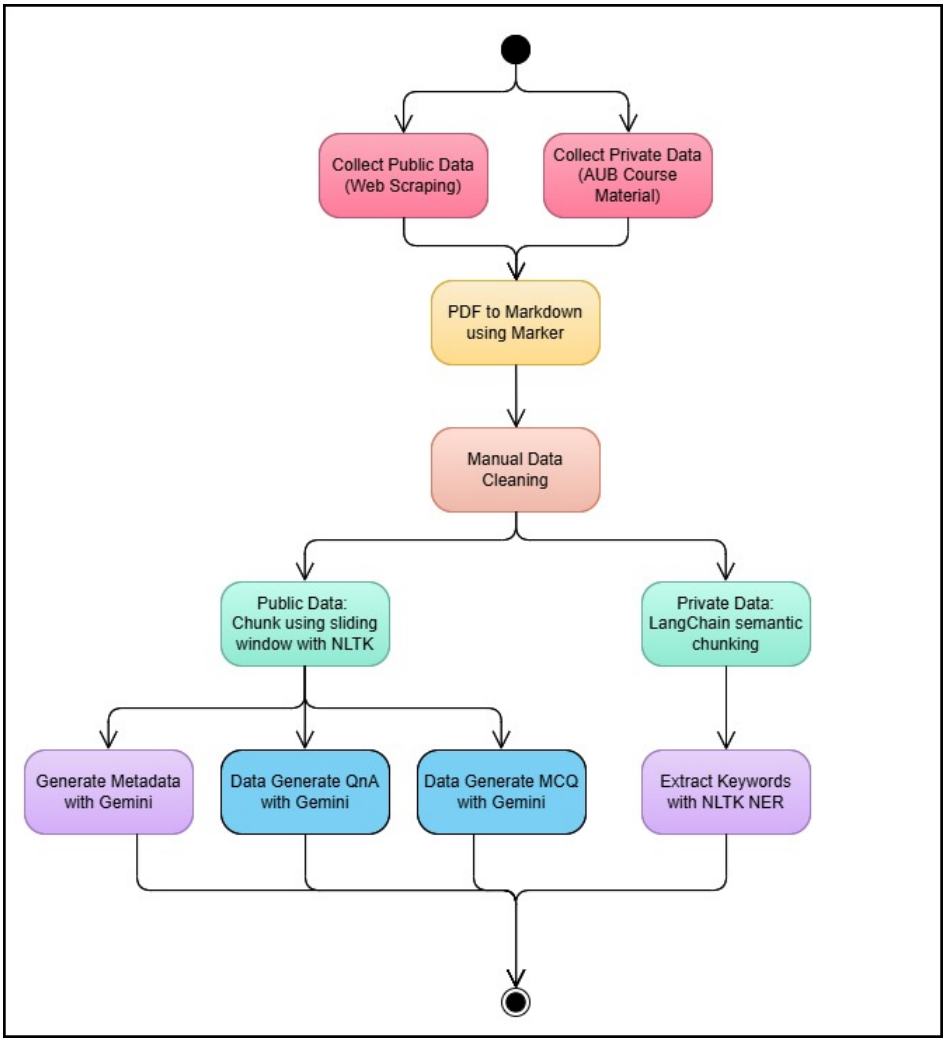


Figure 3.1: The Data Workflow Pipeline

The data was sourced from two primary channels: publicly available online resources on one hand and proprietary problems and books from Prof. Abou Faycal at the American University of Beirut on the other.

- **Online Open-Source Scraping:** The bigger portion of the dataset was collected through targeted web scraping of open-source academic content available to the public. The focus was on educational platforms, open university courseware repositories, and technical forums offering high-quality materials related to communication systems. The search was conducted based on well-define criteria to ensure the quality of the data such as the degree of relevance to EECE 442 and the closeness in terms of materials, document format and readability for high processing quality and sufficiency in depth and comprehensiveness. Overall, we were able to collect 85 online sources which were open universities' course contents and materials and separate individual materials.
- **University-Provided Course Materials:** In addition to public content, Professor Ibrahim Abou Faycal from the Electrical and Computer Engineering department at the American University of Beirut generously provided part of the course's proprietary data from his own Book to enrich the project's capabilities. While the private data constitutes a much smaller portion of the full dataset, it is the best content we have since the course is based on it directly. Thus its relevance is the highest and was given equal weight in the RAG system we developed.

Data Diversity and Representation:
Throughout the collection process, we ensured that the dataset captured a broad representation of topics within the domain of Communication System Design and as closely related to EECE 442's contents and course objectives. The search was based on the chapters covered in the course and the sub-topics of these chapters. EECE 442 is a course that begins as general in the sense that the first few chapters can be found in many similar courses in US top engineering universities, however some topics were more focused and nice, thus requiring targeted search into mathematics materials and graduate level engineering courses to cover the full range of topics.

3.1.2 Data Processing

Now that we had a diverse repository of both public and proprietary collected content, the next critical phase was data processing. The objective of this phase was to transform raw, unstructured materials into structured, queryable, labeled, and useful datasets tailored for fine-tuning and RAG. Given the broad types of techniques we needed to implement, processing was particularly nuanced and involved multiple stages.

Objectives of the Data Processing Pipeline

Data is the vital essence of LLMs, not only does it give the model its understanding or reasoning but it has numerous purposes all throughout development. Before delving into our implementation, we defined the following objective of the data process.

A) **QnA Dataset for Fine-Tuning:** A curated and labeled question-answer dataset was essential for reinforcing the model's understanding of key domain concepts, terms, and problem-solving techniques. The QnA format enables fine-tuning the model in a way that reinforces knowledge recall and synthesis. These questions were divided into easy, medium, and hard.

B) **Metadata for RAG Retrieval:** Metadata for the chunks is essential to support accurate document retrieval in the RAG framework, as well as to have the possibility of expanding to a more advanced RAG system in the future. Metadata includes topic tags, difficulty levels, sources of the chunks, semantic tags, and many more tags that can be leveraged to improve the retrieval task.

C) **MCQ-Based Evaluation Dataset:** One of the most important features in our project is the ability to benchmark and monitor the system's performance during development. Therefore, an important aspect of data processing involved constructing a multiple-choice evaluation dataset. These MCQs are based on the processed data and span a range of difficulty levels to assess both factual-recall and deeper conceptual understanding.

Data Format Normalization

The raw data at its first stage was purposefully collected with strict formatting criteria where we limited our collection to LaTeX styled PDF documents, or academic PDF formats, thus we excluded all PowerPoints, PDF parsed from different formats and watermarked docs. This drove us to eliminate unfit sources from our collection reducing our dataset to 70 sources.

As a first step, all of our content was normalized into a common file format of Markdown. Markdown was selected for its lightweight structure, human readability, ease of further processing using text-based tools and most importantly its compatibility with LangChain and other major tools. As simple as it seems, this process it no trivial task. The versatility of PDF format and its tolerance of images, mathematical equations, and many more document components presented challenges to parsing that would require rigorous work and involved the use of third-party tools. This challenge led us to look for new software and ML models that can assist in this laborious task.

The first method we tested was to try and bluntly provide commercial AI models to parse PDF files directly into Markdown. Without the use of API calls, we relied on a direct interaction with ChatGPT-4o, which with extensive explanation and clarification successfully converted one small PDF into a Markdown. The results of this trial showed extensive slowness in response time, errors in parsing and inability to process larger PDFs and the exposition of our data to external sources in this case the model host companies, thus the option of using such tools was not appropriate to our project.

The second method we explored was using local tools and system that would work on private hardware to parse PDF documents to Markdown. We found two major systems that are most popularly used “Nougat” and “Marker”. After some research about both, it was clear by the performance comparison analysis that Marker is the latest more powerful system of the two exceeding Nougat in speed and accuracy while preserving the same computation requirements on the hardware. Hence, we used Marker for our parsing process, which yielded near-satisfactory results in terms of accuracy, success rate in using Optical Character Recognition (OCR) for figures and graphs extractions and a good mathematical conversion tool for the formulas and math. Marker is a system that uses small LLMs, python libraries and OCR tools and techniques to read the PDF contents and translate them to Markdown with LLMs Fine Tuned for that purpose all on our local machines allowing us to preserve the data integrity of our private data.

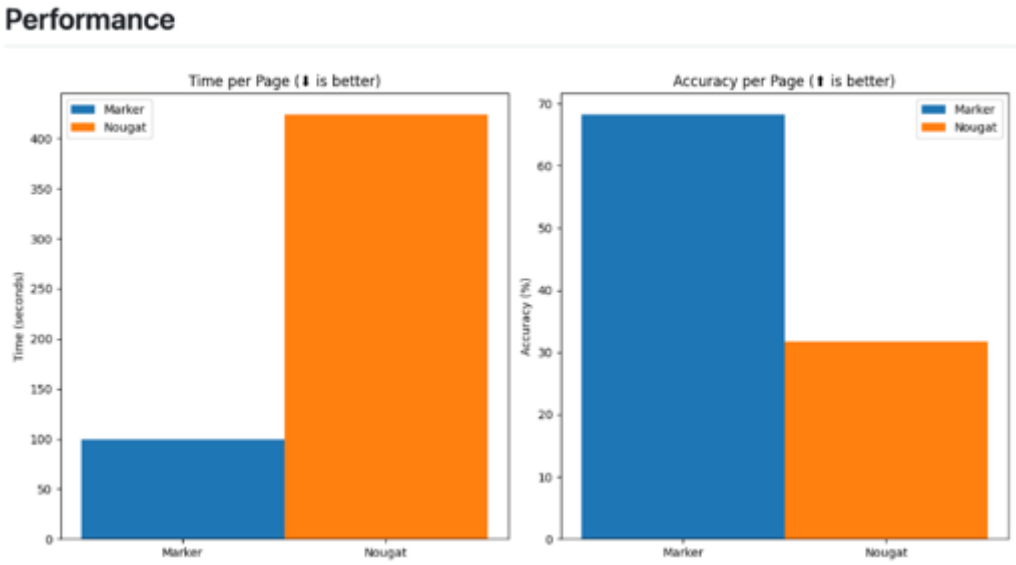


Figure 3.2: Nougat vs Marker Performance [1]

Data Cleaning

Once we had converted our PDF files, a thorough cleaning process was applied to remove parsing errors that prevent the rendering of the content, fix wrongly parsed errors, correct OCR faults, and eliminate noise. Moreover, mathematical equations are passed in LaTeX format, which required more work to correct errors due to the harder readability. This process was done in a purely manual fashion, and required extensive work to effectively clean the entire dataset and make it suitable for RAG and Fine Tuning.

Chunking Strategy

RAG systems rely on context-relevant retrieval from vectorized document chunks, so we applied a multi-tiered chunking strategy. Therefore, it was necessary that we split our large data into consistent chunks with similar size to make the most out of the RAG system.

This part of the workflow requires separate practices for the public and the private data for purposes related to later stages of the process, primarily in the Metadata Generation. Chunking affects the way we will generate the metadata for RAG, since we eventually use techniques that expose the data to API calls, we processed the private data in a different manner to preserve its privacy.

For the public data, the chunking process was done using a series of python scripts that firstly, gather all the markdown-parsed files into one directory, com-

bining all the files from the different public sources together in a single Markdown document while adding placeholder between each source to preserve the separation of the sources. These processes were done using the python function below:

```python
def merge_markdown_files(src, output):
    md_files = [f for f in os.listdir(src) if f.lower().endswith(".md")]
    if not md_files:
        print("No markdown files found in the source directory.")
        return
    with open(output, "w", encoding="utf-8") as outfile:
        for idx, md_file in enumerate(md_files):
            file_path = os.path.join(src, md_file)
            outfile.write(f"<!-- Source: {md_file} -->\n")
            with open(file_path, "r", encoding="utf-8") as infile:
                outfile.write(infile.read())
            if idx < len(md_files) - 1:
                outfile.write("\n\n")
    print(f"Merged {len(md_files)} markdown files into: {output}")
```

Figure 3.3: Python function that merges all our markdown files in one place and separates them with placeholders

Finally, using the resulting Markdown document, we iterated on that document and applied a rule-based and length-aware sliding window that applied multi-tiered chunking (chunking based on textual context like small sections, subsections, bullet-points and so forth) to a flexible limit of 750 words per chunk. In addition to that technique of chunking, we enhanced the sliding window using the "NLTK" library, that prevents the abrupt chunking of sentences in case of length limit excess, by ensuring that chunking occurs at the closest end of sentence despite the chunk limit size. This resulted in a coherent, size-efficient, and semantically appropriate chunking of the data and stored the chunks in a JSON file for further processing. All that process is applied through the python function below:

```python
def chunk_markdown_file(file_path, max_words=750):
    ...
    #code that gets the source name tag and where it begins
    ...

    for match in matches:
        subsection_name = match.group(2).strip()
        subsection_name = re.sub(r"\*\*(.*?)\*\*", r"\1", subsection_name)
        subsection_name = re.sub(r"^\d+(\.\d+)*\s*", "", subsection_name)
        subsection_pos = match.start()
        for pos, source in sorted(source_positions.items()):
            if subsection_pos >= pos:
                current_source_name = source
            else:
                break
        if subsections:
            subsections[-1]['content'] = content[last_pos:subsection_pos].strip
                ()
        subsections.append({
            'subsection': subsection_name,
            'content': '',
            'source': current_source_name
        })
        last_pos = match.end()
    if subsections:
        subsections[-1]['content'] = content[last_pos:].strip()

    result_dict = {}
    for subsection in subsections:
        section_name = subsection['subsection']
        source_name = subsection['source']
        full_name = f"{source_name}_{section_name}"

        sentences = sent_tokenize(subsection['content'])
        current_chunk = []
        chunk_index = 1
        chunk_list = []

        for sentence in sentences:
            current_chunk.append(sentence)
            if len(" ".join(current_chunk).split()) >= max_words:
                chunk_list.append(" ".join(current_chunk))
                current_chunk = []
        if current_chunk:
            chunk_list.append(" ".join(current_chunk))
        if len(chunk_list) == 1:
            result_dict[full_name] = chunk_list[0]
        else:
            for idx, chunk in enumerate(chunk_list, start=1):
                result_dict[f"{full_name}_chunk_{idx}"] = chunk

    return result_dict
```

Figure 3.4: Python function chunks the large markdown file in a customized manner

To process our private data, we chose to use a secure LangChain-based chunking tool backed by NLTK that employs semantic chunking via gradient-based and percentile-aware methods. We choose to apply semantic chunking here - a superior method to the python script above - due to the importance and scarcity of

our proprietary data. It is more computationally expensive initially as it relies on the semantic meaning in the embeddings of each chunk, given the embedding model of our choice, and thus this technique is only applied to the smaller proprietary data. Gradient technique looks at the change in the semantics of the data and chunks it accordingly, and by adding a percentile metric to it LangChain detects a threshold in the percentage change in semantic between sentences and cuts off at a specific percentage of difference. For our data, based on trial and error, we tuned the percentile threshold to 92.5% difference in semantic meaning.

Additional API-Based Cleaning

Once QnA and metadata were generated, we applied a second round of cleaning specific for RAG to make each chunk easier to be semantically fetched. This was done using an API pipeline that calls an external LLM model. This cleaning step targeted mainly references to figures, tables, or sections not present in the chunk since this information will not be useful when retrieving the documents and would act as noise. Thus, we were able to increase the retrieval quality by making each chunk further self-contained while minimizing external dependencies and symbol ambiguity.

3.1.3 QnA Generation for Fine-Tuning

Domain Fine Tuning is based on Question-and-Answer pairs that are used to enhance the models with words, concepts, and context. Since the data we collected was purely educational and lecture styled, we had to rely on creating our own QnA dataset to train our model. Although there exist few open-source datasets focused on telecommunications in general, we found after extensive research that these repositories are too general, go way beyond EECE 442 and not into the relevant material enough to be helpful in our system. Therefore, we found its best to generate our own QnA pair-set based on the data we collected.

This stage of the workflow involved generating synthetic QnA pairs to simulate domain-specific interactions. We used a free, experimental version of Google Gemini (gemini-2.0-flash-thinking-exp), accessed via API, to iteratively generate difficulty-based categorized questions for each document chunk. The model chosen was powerful enough to perform this task, and required no funding to operate, however, collects the data provided to it. Hence, we only use the public data we had collected for this stage.

Each call to the API included the context chunk along with a prompt instructing the model to generate tiered question-answer pairs.

```python
system_prompt = (
    "You are an AI developer for LLMs and you are working on data to fine-tune a
         base model on a complex topic: 'Communication and Telecom Systems.'"
    "Your task is to generate 9 QnA pairs from the chunk of data to train the
        LLM with the concepts, terminology, and nuances in the material."
    "Ensure that each question is self-contained and does not refer to the text
        since the chunks are parts of a larger document."
    "Generate easy, medium, and hard questions with their answers in JSON format
        , structured as follows: "
    ...Sample QnA...
)
for index, (chunk_key, chunk) in enumerate(chunks, start=1):
    print(f"\nProcessing chunk {index}/{len(chunks)}...")
    retry_attempts = 6
    for attempt in range(1, retry_attempts + 1):
        try:
            print(f"Attempt {attempt} for chunk {index}")
            response = model.generate_content(system_prompt + "\n\n" + chunk)
            raw_text = response.text.strip()
            #... code that uses regex to clean the response in order to avoid
                JSON parsing issues...
            qna_pairs = json.loads(raw_text)
            clean_chunk = qna_pairs[0].get("clean_chunk", "").strip()
            if clean_chunk:
                clean_chunk_path = os.path.join(output_dir, "clean_chunks.json")
                with open(clean_chunk_path, "a", encoding="utf-8") as f:
                    f.write(f'    "{chunk_key}" : "{clean_chunk}",\n')
                print(f"Saved clean chunk for chunk {index}")
            categorized_qna = {"easy": [], "medium": [], "hard": []}

            for item in qna_pairs[1:]:
                question = item.get("question", "").strip()
                answer = item.get("answer", "").strip()
                difficulty = item.get("difficulty", "").strip().lower()
                if question and answer and difficulty in categorized_qna:
                    categorized_qna[difficulty].append({"question": question, "
                        answer": answer})

            #...Code that saves each difficulty category in a separate JSON file
                ...

            break
```

Figure 3.5: Python script that generates QnA pairs in easy medium and hard difficulty using API calls

The results were structured in JSON format and post-processed to standardize formats and eliminate unneeded responses from the model. This process yielded over 17,000 Questions and Answer pairs for us to use in Fine-Tuning, equally distributed over easy, medium, and hard questions. Here are samples of QnA pairs generated:

Sample QnA Pairs:

- **Easy QnA Sample:**

  {

```
"question": "What symbol is commonly used to represent the analog
             baseband message signal?",
"answer":   "The analog baseband message signal is commonly denoted by m(t)."
}
```

- **Medium QnA Sample:**

```
{
"question": "What is the primary trade-off when choosing between
             QPSK and 16QAM modulation schemes in terms of spectral
             efficiency and power efficiency?",
"answer":   "16QAM is spectrally more efficient than QPSK because
             it transmits more bits per symbol (4 bits/symb vs 2 bits/symb)
             thus achieving a higher data rate for the same bandwidth.
             However, 16QAM is less power-efficient than QPSK,
             requiring a higher Signal-to-Noise Ratio (SNR), or Eb/N0,
             to achieve the same BER."
}
```

- **Hard QnA Sample:**

```
{
"question": "What is the relationship between the bandwidth
             of the complex envelope and the physical bandwidth
             of a passband signal in linear modulation?",
"answer":   "The two-sided bandwidth of the complex envelope of a
            linearly modulated signal is equal to the physical bandwidth
            of the corresponding passband signal.
            This equivalence simplifies bandwidth analysis by allowing us
            to work with the baseband complex envelope."
}
```

3.1.4 Data Tokenization for Fine Tuning

Now that our fine tuning data is ready, it was necessary to tokenize it.

First, for using the data with deepspeed and accelerate fine tuning on Mathstral 7B, we must carefully process it. We maintain consistent formatting across all inputs using Mistral's expected format, and implement proper padding strategies (left-side padding) which is crucial for causal language models. The tokenizer is configured with appropriate special tokens and padding tokens to ensure correct model inputs. Below, we use an 85/15 train-validation split for each dataset. The validation sets are used for monitoring the model's performance during training.

We prepare all this data only once and save it to disk for future use. Next, we begin by tokenizing by the means of test functions in order to plot the distribution of lengths of input_ids, ie the lengths in number of tokens of the tokenized chunks or QA pairs. From the graphs, we are above to figure out a good choice for the maximum length of the input_ids that we should use for the tokenizing. Then, we proceed to tokenize using the true tokenizing functions also below. Note that in the tokenizing functions, there is a labels field, which is used for the loss computation during training. The labels field is a copy of the input_ids field, except for the question part, which is set to -100. This is because we want to ignore the question part during loss computation, hence training to predict the answer part only. Negative 100 is the conventional choice in transformer models for tokens we want to ignore during training.

Here is the main code for the above:

```

import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

#MISTRAL based models expect inputs like: <s>[INST] {question} [/INST] {answer} </s>

def formatting_func_docs(example):
    instruction_placeholder = " "
    return f"[INST] {instruction_placeholder} [/INST] {example['text']}"


def formatting_func_qa(example):
    text = f"[INST] {example['question']} [/INST] {example['answer']}"
    return text


base_model_path = "/workspace/model_save"

tokenizer = AutoTokenizer.from_pretrained(
    base_model_path,
    padding_side="left",
    add_eos_token=True,
    add_bos_token=True,
)
tokenizer.pad_token = tokenizer.eos_token


def generate_and_tokenize_prompt_docs_test(prompt):
    return tokenizer(formatting_func_docs(prompt))

def generate_and_tokenize_prompt_qa_test(prompt):
    return tokenizer(formatting_func_qa(prompt))

def generate_and_tokenize_prompt_docs(prompt, max_length):
    result = tokenizer(
        formatting_func_docs(prompt),
        truncation=True,
        max_length=max_length,
        padding="max_length",
    )
    chunk = f"{prompt['text']}"
    chunk_tokens = tokenizer(chunk, add_special_tokens=False)
    labels = chunk_tokens["input_ids"]

    # If labels exceed max_length, keep only the rightmost tokens (since left padding aligns with model logic)
    if len(labels) > max_length:
        labels = labels[-max_length:]
    else:
        pad_length = max_length - len(labels)
        # Left-pad labels with -100 so that the labels are right-aligned
        labels = [-100] * pad_length + labels

    result["labels"] = labels
    return result

def generate_and_tokenize_prompt_qa(prompt, max_length):
    # tokenizing the question part only
    question = f"[INST]{prompt['question']}[/INST]"
    question_tokens = tokenizer(question, add_special_tokens=False)

    answer = f"{prompt['answer']}"
    answer_tokens = tokenizer(answer, add_special_tokens=False)

    # tokenizing the full text
    full_text = formatting_func_qa(prompt)
    result = tokenizer(
        full_text,
        truncation=True,
        max_length=max_length,
        padding="max_length",
    )

    labels = [-100] * len(question_tokens["input_ids"]) + answer_tokens["input_ids"]

    # If labels exceed max_length, take only the rightmost tokens.
    if len(labels) > max_length:
        labels = labels[-max_length:]
    else:
        pad_length = max_length - len(labels)
        labels = [-100] * pad_length + labels

    result["labels"] = labels
    return result
```

Figure 3.6: Tokenization for Mathstral Fine Tuning Code

Note that we also generated the below chart for the token counts in our entire dataset. The QnA data is around 6 million token, whereas the chunk data 5 million. Then chunk data ended up being used only for RAG, so we can say there is aroung 5 million tokens of data in our databases (note: actual token count will vary as the tokenizer used for the database is different from the mathstral one. similarly, tokenizer used in QWEN is different form mathsral one. however, the token count discrepency is a minimal one.)

```
Per-split token counts:
  train_docs          : 3,900,000 tokens
  val_docs            : 690,000 tokens
  train_custom_docs   : 282,000 tokens
  val_custom_docs     : 50,400 tokens
  train_easy          : 829,500 tokens
  val_easy            : 146,400 tokens
  train_medium        : 1,823,700 tokens
  val_medium          : 321,900 tokens
  train_hard          : 2,462,000 tokens
  val_hard            : 434,800 tokens

Aggregates:
  total_qna_tokens         : 6,018,300 tokens
  total_chunk_tokens       : 4,922,400 tokens
```

Figure 3.7: Token Counts for the entire dataset

As for tokenization for QWEN starting from having saved the data in the above way, the process is seen in the below code snippet.

[7]:
```python
from datasets import load_from_disk

raw_train_dataset = load_from_disk("/workspace/ft_data/saved_datasets/train_easy")
raw_val_dataset = load_from_disk("/workspace/ft_data/saved_datasets/val_easy")

def create_conversation(example):
    return {
        "conversations": [
            {"role": "user", "content": example["question"]},
            {"role": "assistant", "content": example["answer"]}
        ]
    }

train_dataset = raw_train_dataset.map(create_conversation)
val_dataset = raw_val_dataset.map(create_conversation)

train_dataset = train_dataset.remove_columns(["question", "answer"])
val_dataset = val_dataset.remove_columns(["question", "answer"])
```

[8]:
```python
from unsloth.chat_templates import get_chat_template

tokenizer = get_chat_template(
    tokenizer,
    chat_template = "qwen-2.5",
)

def formatting_prompts_func(examples):
    convos = examples["conversations"]
    texts = [tokenizer.apply_chat_template(convo, tokenize = False, add_generation_prompt = False) for convo in convos]
    return { "text" : texts, }
pass
```

[9]:
```python
train_dataset = train_dataset.map(formatting_prompts_func, batched=True)
val_dataset = val_dataset.map(formatting_prompts_func, batched=True)
```

[10]:
```python
train_dataset[5]["conversations"]
```

[10]:
```
[{'content': 'What is the support of the random variable Mn?', 'role': 'user'},
 {'content': 'The support of Mn, the maximal value of a random walk up to time n, is the set of integers from 0 to n, inclusive: {0, 1, ..., n}.',
  'role': 'assistant'}]
```

Figure 3.8: Tokenization for QWEN Fine Tuning Code

3.1.5 Metadata Generation for RAG

Generation for public data

The performance of RAG can be drastically enhanced with the use of metadata in its various forms to increase semantic precision of retrieval. Although we have the data as clean chunks, we decided to generate our custom metadata for each processed document chunk using the same Gemini-based API employed for QnA generation. The pipeline based on a python script iterates over every chunk document separately and with a standardized prompt, fetches a response from the API that contains the metadata for that particular chunk. Below is a snipped from the python code responsible for this task.

Each chunk was paired with a structured JSON object containing the metadata parameters we chose and would have a structure that includes full chunk text, the subsection or topic, the content source, and an incremental chunk number for the case where a subsection is split into several chunks due to word limit excess. So far, our documents already have these parameters. However, these details are not enough to help with the semantics, so we decided to add the following fields using the API-called LLM.

Firstly, we included a compacted semantic summary that rewrites the chunk in a summarized way with just the necessary words to keep the meaning clear. We also added a list of key words since sentences revolve around key technical words that can be leveraged to link the user's query to the closes key words assemblage. Additionally, we added a difficulty rating that can be used on top of the more semantically relevant parameters. This rating is scaled from 1 to 10 based on the technical difficulty of the chunk. Finally, we added a flag indicating whether the chunk contained mathematical formulas and equation in case we wanted to look for certain chunks with specific formulas.

This metadata improves chunk document filtering and relevance scoring during vector retrieval. The inclusion of semantic summaries and keyword tags allow for more accurate query matching, while fields like difficulty and equation inclusion tag enable further filtering needs. Moreover, the metadata structure was designed to be modular and extensible, allowing for future integration of learning objectives or concept linkage. Although we had included all these tags and filters, using them all at once is not definite as we will discuss in the RAG section of the paper.

```python
def generate_metadata(chunk_text, max_retries=3):
    cleaned_text = clean_text(chunk_text)
    prompt = f"""
    You are an AI engineer working on data to create metadata for chunks of text.
    Given the chunk, return a JSON with:
    {{
        "chunk": "The cleaned chunk without LaTeX/Markdown code",
        "compact_chunk": "Rewriting the chunk in a concise and compact form, do not use phrases like 'This text ...' or 'This text highlights...' only write the ideas directly.",
        "key_words": ["keyword1", "keyword2", "keyword3", use as many words as needed],
        "difficulty_rating": an integer from 1 to 10,
        "contains_equations": "Yes or No"
    }}

    Text:
    {cleaned_text}
    """

    for attempt in range(1, max_retries + 1):
        try:
            response = model.generate_content(prompt)
            response_text = response.text if hasattr(response, 'text') else ""
            print(f"Response (Attempt {attempt}): {response_text[:200]}...")
            json_match = re.search(r"\{.*\}", response_text, re.DOTALL)
            if json_match:
                metadata = json.loads(json_match.group(0))
                return metadata
            print(f"Attempt {attempt}: Failed to extract valid JSON, retrying...")
            time.sleep(2)
        except Exception as e:
            print(f"Attempt {attempt}: Error - {e}")
            time.sleep(2)
    print(f"Skipping chunk after {max_retries} failed attempts.")
    return {
        "chunk": cleaned_text,
        "compact_chunk": "",
        "key_words": [],
        "difficulty_rating": 1,
        "contains_equations": "No"
    }
```

Figure 3.9: Python script that generates Metadata using API calls

Metadata JSON Sample:


```
{
    "chunk": ...
    "subsection": "Conventional AM",
    "source": "AnalogueModulationTechniques_UCSantaBarbara",
    "chunk_number": 1,
    "compact_chunk": "Conventional AM adds a carrier to DSB-SC for envelope...",
    "key_words": [
        "conventional AM",
        "amplitude modulation",
        "DSB-SC",
        "carrier component",
        "Fourier transform",
        ...
    ],
    "difficulty_rating": 7,
    "contains_equations": "Yes"
}
```


Generation for Proprietary Data

As mentioned earlier, handling the private data should ensure it remains private with regard to external online tools. That being said, to generate metadata for the private chunks, we relied on a tagging system based on the NER from NLTK library. NER stands for name entity recognition, which as its name mentions, recognizes textual entities in a sentence as objects and labels their specific instances based on pre-defined entities like "Person's name" or "Object". In our case, we leveraged this tool to extract the key words in the private data chunks to use in our semantic retrieval.

3.2 The RAG System

In this chapter, we will present the design and architecture of our RAG system. The objective behind the use of RAG is to allow the chatbot to dynamically retrieve and append the most relevant segments of the course content as well as the dataset to the prompt, rather than relying solely on its own learned knowledge. By doing so, we allow the model to be context-aware in question answering which directly reflects the course material, both in style and in substance. The chatbot thus becomes capable of revolving its responses around the material covered in EECE 442, especially given the high technical and mathematical complexity of the subject.
A prerequisite to the RAG development and in order for the system to be capable of retrieving content with high precision and domain relevance, a tailored specifically to the EECE 442 syllabus, it is imperative to possess a cleaned, chunked and metadata-enriched dataset. Once this requirement is obtained, we create a vector database, where each data chunk is transformed into numerical representations using an embedding model. These embeddings are then stored in a vector index which allows for efficient similarity-based search.
Our final RAG system also includes an introspection function that allows for real-time inspection of both indices and a sample of the retrieved documents. This utility was valuable during debugging and development, since it enables us to validate that chunking, tagging, and embedding were functioning correctly. Furthermore, it allowed us to visualize the impact of query strategies and metadata filtering on retrieval results, which in turn helped us tune parameters and improve the RAG model's output consistency.

3.2.1 The Metadata Structure

After the data pipeline process, we were ready to develop our RAG system with the data available with its metadata. Understanding the structure of our data is crucial to understanding the structure of our Indexing system. The data for RAG we had prepared is split into two groups as explain in the Data Chapter earlier: "public" and "private" datasets. The "public" dataset contains the data collected cleaned and chunked, however, within the metadata we had included a "Compact Summary" of the chunked data to assist in RAG. Thus, we decided to use this summarized version of the chunk as an "In-Parallel" dataset whose goal is make the semantic comparison easier. The idea of that structure is to have our RAG system compare the user's query to the summarized document dataset which will in turn point to the chunk that it summarized as the actual document to be used for augmentation. This is inspired by langchain's Parent Document Retrieval (PDR) where a child chunk does the same job our chunk summary does here. Thus, for all the publicly gathered dataset, the actual chunk

content is stored as metadata, while the summary becomes the langchain document page_content.
The remainder of the metadata includes "Source name" that cites the source of the given query, "Subsection Title", the title of the chunk in the original document, the "key" technical words of that chunk, the "difficulty rating" of the chunk in terms of semantics from a scale from 1 to 10, and an indicator on whether the chunk contains mathematical equations. Although the summarized chunk of the data is included in the metadata, the system uses it in a different manner than the remaining metadata that could be used at the search and filtering stage of the RAG process, especially in a more advanced RAG setting.
As for the "private" dataset, we leave the chunks themselves in the page_content as our means of search retrieval.

3.2.2 Removing Problematic Chunks

As we ran our evaluation run for the first time, we noticed a recurring problem. When the MCQ would require the LLM to perform some type of calculation or solving a mathematical problem, thus by similarity the retriever would find similar samples from the data and add it to the context. While this looks harmless on the surface, the size and nature of our LLM caused it to use these mathematical chunks as factual data and distort the reasoning or generating process to yield inconclusive answers or even completely wrong answers.

As we have discussed our Metadata structure earlier, the contains equation tag could have been helpful since we could exclude chunks with equations in general however this would exclude material critical chunks from being retrieved and causing low effectiveness regarding RAG generally. Therefore, we decided to add a field to each Metadata chunk called "is-exercises", which is zset to true if the chunk contains mathematical applications, exercises, problems or applied examples. This was achieved by iterating over the entire Metadata set and processed via API calls to a Gemini model that would judge each chunk and fill the new field. Hence, after updating our metadata structure, we iterated over the Metadata set and created a new version of it excluding all chunks with "is-exercise" set to true, thus filtering it to enhance the RAG performance.

Subsequently, the RAG drastically improved answering properly the MCQs, rather than skipping them one by one. Prior to this improvement, no full evaluation was possible as most MCQs were being skipped due to inconclusive answer, whereas after this process, little to no loss is occurring proving this effort to be significantly effective.

3.2.3 Embedding and Indexing

The embedding process constructs the vector-based infrastructure of our RAG system. Each chunk of data for both of our datasets (public, private) are passed through an embedding model that would transform the natural language of each langchain page_content into high-dimensional vector representations. These embeddings capture the semantics of the text, allowing for similarity comparisons between user queries and document chunks.
Embedding models are the backbone of RAG performance. The quality of embeddings varies from model to model, and from domain to another depending on the embedding model training data. Therefore, testing models individually with a standard RAG config setup was highly important to uncover which embedding model would best fit our system. After extensive research about models, we settled on three possible models, Gemma 2B TeleLLM, Stella-400M, and Inf-1.5-B. As explained earlier, we reasoned that taking a Gemma 2B TeleLLM and transforming it into an embedding model by extracting embeddings from its last hidden layer is promising. This is because it has previous training on scientific and telecommunications data and terminology. Below is our code that we carefully developed to achieve this task.

```python


class CustomEmbeddings(Embeddings):
    def __init__(self, model_name: str, device: str = "cuda", save_directory:
        str = None):
        ...

    def embedding_llm(self, sent: str) -> List[float]:
        # Tokenizes with padding and fixed max length.
        tokenized_sent = self.tokenizer(
            sent, padding="max_length", return_tensors="pt", max_length=128
        )
        tokenized_sent = {k: v.to(self.device) for k, v in tokenized_sent.items
            ()}
        self.model.eval()
        with torch.no_grad():
            output = self.model(**tokenized_sent, output_hidden_states=True)
        # Averages the last hidden state over the token dimension.
        last_hidden_state = output.hidden_states[-1]  # shape: (1, seq_length,
            hidden_size)
        embedding = last_hidden_state.mean(dim=1).squeeze()  # shape: (
            hidden_size,)
        return embedding.cpu().tolist()

    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        embeddings = []
        for text in texts:
            emb = self.embedding_llm(text)
            embeddings.append(emb)
        return embeddings

    def embed_query(self, text: str) -> List[float]:
        # For a single query, simply calls embed_documents and return the first
            result.
        return self.embed_documents([text])[0]

    def get(self):
        # Simply returns self so that embedding_model.get() is callable and has
            the langchain expected interface.
        return self

    def save_pretrained(self, save_directory: str):
        ...
```

Figure 3.10: Code for Custom Embeddings from an LLM, used for Gemma 2B TeleLLM

On the other hand, the Stella-400M has proven to yield very positive increases in performance in RAG previously, with a remarkably low number of weights. Finally, we settled on the Inf-1.5-B as a good model used for RAG as well. Before testing each and every model above, we evaluated the different configuration filters of RAG to get the best setup for the RAG. After extensive testing and evaluation, we found that the Stella-400M performed the best,and hence it made it to our final setup. The details of the evaluation are shown in the RAG Evaluation section.

We applied in our RAG vector a “dual-index” structure built on top of the

FAISS vector store that integrates LangChain's document handling modules. This structure involves two parallel indices: a summary-index that contains compacted semantic representations of each public data chunk, and a full-index that stores the complete content of the original proprietary chunks. Both indices use vector embeddings which are computed using a custom wrapper for consistent batch processing and compatibility with LangChain. Note that since we are trying and evaluating 3 different embedding models, we had to recreate both indices 3 times each, as their content will include the vector representations which directly depend on the embedding model choice (also, the way we split the proprietary data differs since we use a semantic chunker for it). Below is the function that implements the dual indexing as explained above. Also note that for the standard non-dual indexing code has similar logic.

```
    def index_documents(self, documents: List[Dict[str, Any]]):
        print("[INFO] Creating new FAISS dual indices...")
        summary_docs = []
        full_docs = []
        for d in documents:
            metadata = {
                "chunk": d["chunk"],
                "source": d["source"],
                "subsection": d["subsection"],
                "difficulty_rating": d["difficulty_rating"],
                "contains_equations": d["contains_equations"],
                "tags": d.get("key_words", [])
            }
            summary_doc = Document(page_content=d["compact_chunk"], metadata=
                metadata)
            full_doc = Document(page_content=d["chunk"], metadata=metadata)
            summary_docs.append(summary_doc)
            full_docs.append(full_doc)
        summary_index = FAISS.from_documents(summary_docs, self.embedding_model.
            get())
        full_index = FAISS.from_documents(full_docs, self.embedding_model.get())
        summary_path = os.path.join(self.index_dir, "summary")
        full_path = os.path.join(self.index_dir, "full")
        summary_index.save_local(summary_path)
        full_index.save_local(full_path)
        return
```

Figure 3.11: Indexing Code

3.2.4 Full RAG Setup Details

In this section, we discuss in details the implementation, and then we attach relevant code snippets. At query time, the RAG system executes a multi-stage retrieval and generation workflow. The user submits a question to the system, which is then vectorized and used to query both the summary and full index using either Maximum Marginal Relevance (MMR) or standard similarity search (Cosine Similarity), depending on the configuration. Additionally, we added the

option to activate custom metadata filtering, which for our end result product, we found out that they either do not help substantially, or they need more work to be of good use. Based on the enabled filter tags, the system attempts to infer from the question relevant tags by embedding the query and matching it against all known tags using cosine similarity. This auto-tagging mechanism narrows down the search space to documents containing semantically aligned metadata. This process occurs twice, the first occurrence for the public dataset, comparing the semantics of the summarized set to the query in order to retrieve its full chunk complementary, and the other occurrence on the private data specific to EECE 442. Note that the data that has been embedded must be retrieved using the same embedding model as the encoding methods vary from model to another.
Both results are passed to a Reranker, for which we will have specified the required number of chunks, k. The re-ranker then uses a cross-encoder architecture to take in the query and the retrieved documents and scores each pair and sort the results by semantic relevance with higher precision rather than pure embedding-based similarity. The selected top-k documents are then concatenated into a context block used as input for the final generation phase. Subsequently, a prompt template is constructed that includes both the question and the concatenated document content along with the private data fetched and is sent to the LLM instance for response generation.

```python

class FAISSDualRetriever(VectorIndex):
    def __init__(self, embedding_model:Embeddings, index_dir: str = "faiss_index_0"):
        self.embedding_model = embedding_model
        self.index_dir = index_dir
        self.summary_index = None
        self.full_index = None
        self._load_indices()

    def _load_indices(self) -> bool:
        summary_path = os.path.join(self.index_dir, "summary")
        full_path = os.path.join(self.index_dir, "full")
        if os.path.exists(os.path.join(summary_path, "index.faiss")) and os.path.exists(os.path.join(
             full_path, "index.faiss")):
            self.summary_index = FAISS.load_local(summary_path, self.embedding_model.get(),
                 allow_dangerous_deserialization=True)
            self.full_index = FAISS.load_local(full_path, self.embedding_model.get(),
                 allow_dangerous_deserialization=True)
            print("[INFO] Loaded FAISS dual indices from disk.")
            return True
        else:
            raise FileNotFoundError("Dual indices not found. Please build them first using FAISSDualIndexer
                 .")


    def get_retriever(self, mmr: bool = True, k: int = 6) -> VectorStoreRetriever:
        strategy = "mmr" if mmr else "similarity"
        summary_retriever = self.summary_index.as_retriever(search_type=strategy, search_kwargs={"k": k})
        return summary_retriever


class FAISSRetriever(VectorIndex):
    def __init__(self, embedding_model, index_dir: str = "faiss_index_custom"):
        self.embedding_model = embedding_model
        self.index_dir = index_dir
        self.custom_index = None
        self._load_index()

    def _load_index(self):
        custom_path = os.path.join(self.index_dir, "custom")
        if os.path.exists(os.path.join(custom_path, "index.faiss")):
            self.custom_index = FAISS.load_local(custom_path, self.embedding_model.get(),
                 allow_dangerous_deserialization=True)
            print("[INFO] Loaded custom FAISS index from disk.")
        else:
            raise FileNotFoundError("Custom FAISS index not found. Please build it using FAISSIndexer.")

    def get_retriever(self, mmr: bool = True, k: int = 6) -> VectorStoreRetriever:
        strategy = "mmr" if mmr else "similarity"
        retriever = self.custom_index.as_retriever(
            search_type=strategy,
            search_kwargs={"k": k}
        )
        return retriever
```

Figure 3.12: Retriever Code for public (dual) and private (regular) chunk retrieval

```python
class Reranker:
    def __init__(self, model_name: str = "cross-encoder/ms-marco-MiniLM-L-6-v2"):
        self.model = CrossEncoder(model_name, device="cuda")

    def rerank(self, query: str, docs: List[Document], top_k: int = 4) -> List[Document]:
        pairs = [(query, doc.page_content) for doc in docs]
        scores = self.model.predict(pairs)
        scored_docs = list(zip(scores, docs))
        ranked = sorted(scored_docs, key=lambda x: x[0], reverse=True)
        return [doc for _, doc in ranked[:min(len(ranked), top_k)]]
```

Figure 3.13: Reranker Code

```python


class RAGPipeline:
    def __init__(self,
                 embedding_model_class, embedding_model_name: str, hf_model_path: str, dual_index_dir: str,
                       custom_index_dir: str, prompt_template: str):
        self.embedder = embedding_model_class(embedding_model_name, device="cuda")
        self.dual_retriever = FAISSDualRetriever(self.embedder, index_dir=dual_index_dir)
        self.custom_retriever = FAISSRetriever(self.embedder, index_dir=custom_index_dir)
        self.llm = LLMWrapper(model_path=hf_model_path)
        self.use_reranker = True
        self.reranker = Reranker()
        self.prompt_template = prompt_template
        self.tag_embeddings = None
        self._compute_all_tags()

    def _compute_all_tags(self):
        """
        Gathers tags from docstores of both retrievers and stores them in self.all_tags. Precomputes all
             tag embeddings. If saved files exist, load them instead of recomputing.
        """
        ...


    def auto_select_tags(self, query: str, top_n: int = 20) -> List[str]:
        """
        Uses self.all_tags to find the top_n closest tags to the query (cosine similarity).
        """
        if not self.all_tags:
            print("[WARN] No tags available for auto_select_tags.")
            return []

        query_embedding = self.embedder.embed_query(query)
        scores = []
        for tag, tag_emb in zip(self.all_tags, self.tag_embeddings):
            sim = self._cosine_similarity(query_embedding, tag_emb)
            scores.append((tag, sim))

        scores.sort(key=lambda x: x[1], reverse=True)
        selected_tags = [tag for (tag, _) in scores[:top_n]]
        return selected_tags

    def query(self, question: str, conversation_history:str, top_k: int = 4,
              use_mmr: bool = True,
              auto_tag_filter: bool = False,
              tag_filter_top_n: int = 5,
              use_reranker: bool = False,
              ) -> Tuple[str, List[Document], List[Document], List[Document]]:
        """
        Main query method. Optionally filters docs by tags, merges results from dual & custom retrievers,
        optionally reranks, then uses the prompt_template to generate final answer.
        """
        metadata_filter = None
        if auto_tag_filter:
            selected_tags = self.auto_select_tags(question, top_n=tag_filter_top_n)
            metadata_filter = {
                "tags" : {"$in": selected_tags}
            }

        if use_reranker:
            print("[INFO] Reranker is enabled.")
            dual_retriever_obj = self.dual_retriever.get_retriever(mmr=use_mmr)
            dual_docs = dual_retriever_obj.invoke(question, filter=metadata_filter)
            custom_retriever_obj = self.custom_retriever.get_retriever(mmr=use_mmr)
            custom_docs = custom_retriever_obj.invoke(question, filter=metadata_filter)
        else:
            per_retriever_k = max(1, top_k // 2)
            dual_retriever_obj = self.dual_retriever.get_retriever(mmr=use_mmr)
            dual_docs = dual_retriever_obj.invoke(question, k=per_retriever_k, filter=metadata_filter)
            custom_retriever_obj = self.custom_retriever.get_retriever(mmr=use_mmr)
            custom_docs = custom_retriever_obj.invoke(question, k=per_retriever_k, filter=metadata_filter)

        for d in dual_docs:
            d.metadata["summary"] = d.page_content
            if "chunk" in d.metadata:
                d.page_content = d.metadata["chunk"]
            else:
                print("[WARN] Document missing 'chunk' in metadata:", d.metadata)

        docs = dual_docs + custom_docs

        if use_reranker and self.reranker:
            if len(docs) != 0:
                docs = self.reranker.rerank(question, docs, top_k=top_k)
                print("[INFO] Reranker used on merged documents.")

        context = "\n\n".join([doc.page_content for doc in docs])

        prompt = self.prompt_template.format(
                history=conversation_history,
                question=question,
                context=context
                )

        response = self.llm.generate(prompt)

        return response, context
```


Figure 3.14: Integrated RAG Pipeline Class

```
"""
    [INST] You are CommGPT, a friendly, motivational,
       and knowledgeable teaching assistant for
       Telecommunications systems.

    The conversation so far (each pair is user ->
       assistant):
    {history}

    The user's newest question is: {question}

    You have access to the following retrieved
       documents from the knowledge base:
    {context}

    Please provide a helpful, accurate, and context-
       aware answer, using the retrieved docs **if**
       they are relevant, but without referring to them
       .
    If no relevant knowledge is found in the documents,
        or they do not help in addressing the user's
       question, then answer from your general
       knowledge or reasoning.
    If the user asks a questions not related to
       Telecommunications Systems, **gently yet
       assertively** redirect the discussion.
    [/INST].
"""
```

Figure 3.15: Full RAG Pipeline Instantiation

3.3 Fine Tuning

To conduct our fine-tuning procedures, we adopted a two-phase process to systematically enhance our models' domain knowledge and reasoning capabilities. In the first phase, we expose the base model to "informational chunks" extracted from our curated public and private corpora. These chunks—each containing a coherent segment of lecture notes, derivations, or definitions—serve to ground the model in EECE 442's specialized vocabulary, mathematical notation, and conceptual structure. Once this grounding is established, we transition to a multi-stage Question-Answer (QA) sequence designed to reinforce reasoning skills. Stage 1 ("Easy") focuses on straightforward factual and definitional queries, ensuring

the model can retrieve and restate key concepts. Stage 2 ("Medium") introduces multi-step questions that require the integration of related ideas and basic problem-solving. Finally, stage 3 ("hard") presents complex open-ended problems and derivations that require deeper mathematical reasoning and domain expertise. By gradually increasing task complexity and guiding the model through successively challenging contexts, this approach has been shown to enhance complex reasoning capabilities in large language models [27].

By separating the fine-tuning into these two complementary phases, we aim to combine deep content grounding with targeted reasoning practice, yielding a model that not only "knows" the material but can also apply it accurately when confronted with advanced queries. The following subsections describe how we implemented this workflow for Mathstral-7B and Qwen 2.5-3B.

3.3.1 Mathstral-7B Fine-Tuning

We began our experiments by applying 8-bit quantization to the Mathstral-7B model, followed by a QLoRA (Quantized Low-Rank Adapter) fine-tuning setup on AUB's high-performance computing cluster. Quantization refers to the process of reducing the numerical precision of a model's weights—in our case, from 32-bit floating point down to 8-bit integers—thereby drastically cutting memory usage and enabling larger batch sizes or fitting larger models on limited GPU memory. While this can accelerate training and inference, it also introduces approximation error, which may degrade model performance if not handled carefully.

QLoRA builds on this concept by freezing the quantized weights of the base model and injecting a small number of trainable low-rank adapter matrices. Specifically, we applied LoRA adapters to just 1.2% of the model's parameters; the remaining 98.8% remained in their 8-bit frozen state. During fine-tuning, only these low-rank matrices are updated, which dramatically reduces the required GPU memory and compute cost compared to full-parameter updates. QLoRA thus attempts to combine the efficiency of quantization with the flexibility of adapter-based tuning. Despite these optimizations, our initial runs showed no meaningful improvement in either training or evaluation loss.

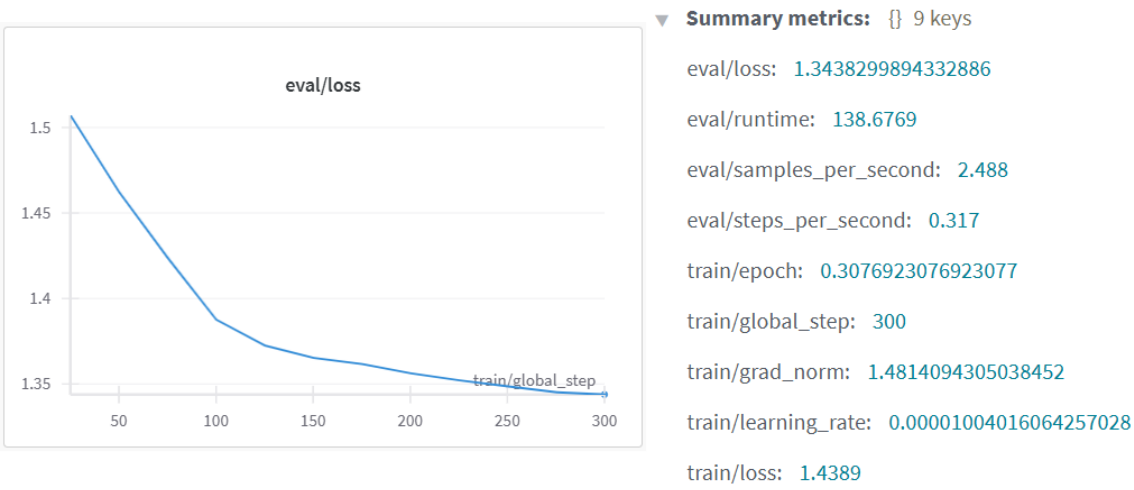


Figure 3.16: Evaluation loss curve and Summary metrics for Mathstral-7B 8-bit quantized QLoRA fine-tuning.

Building on the same training configuration, we then replaced QLoRA with a standard LoRA setup on the full-precision Mathstral-7B model, this time without applying quantization. The goal was to isolate whether the quantization step had negatively impacted model adaptability. While this setup allowed for greater precision by keeping the model weights in full floating-point representation, the resulting changes in both training and evaluation loss curves remained minimal. The model showed only slight adjustments in loss metrics, indicating that even without quantization, LoRA-based tuning on this model was insufficient to yield noticeable improvements in learning performance within our domain.

Following the underwhelming results of LoRA fine-tuning, we attempted to perform full fine-tuning on the quantized version of Mathstral-7B, aiming to update all model weights directly while still benefiting from reduced memory usage. However, this approach quickly failed with an error during the backpropagation step. Upon investigation, we learned that full fine-tuning on quantized models is fundamentally unsupported: quantization locks the model weights into low-precision formats that are not differentiable, thereby preventing gradient updates. Since the underlying quantized tensors cannot participate in standard optimization routines, full backpropagation becomes infeasible under such constraints.

In an effort to salvage the full fine-tuning approach, we removed quantization entirely and attempted to train the full-precision Mathstral-7B model. However, this led to immediate CUDA out-of-memory (OOM) errors on AUB's HPC infrastructure. Recognizing the memory demands of a model of this scale, we secured

departmental funding to rent more powerful GPUs via RunPod. RunPod is a cloud-based platform that offers on-demand access to high-performance GPUs, including Nvidia A100s and H100s, commonly used for deep learning workloads. It allows researchers to spin up GPU instances preconfigured with popular machine learning frameworks at a fraction of the cost of major cloud providers. Despite these upgraded resources, the OOM issue persisted. The memory footprint of full-precision fine-tuning for a 7B-parameter model exceeded what could be reasonably handled on one GPU without advanced optimization strategies.

To address the persistent memory bottlenecks, we attempted to scale training using Distributed Data Parallel (DDP), a native PyTorch library designed to parallelize the training process across multiple GPUs. DDP works by replicating the entire model on each participating GPU and splitting the training data across them. During backpropagation, each GPU computes gradients for its own data shard, and a collective communication step synchronizes the gradients across all replicas. This allows the model to process more data in parallel while maintaining consistent weight updates across devices. DDP is widely regarded for its efficiency and ease of use in multi-GPU environments, particularly when training on large datasets or with large batch sizes [28].

However, a critical requirement for DDP is that the full model must fit into the memory of each GPU individually, since the entire model is instantiated on every device. In our case, the 7B-parameter Mathstral model when loaded for fine tuning by pytorch, remained too large to fit even a single copy per GPU, leading to out-of-memory errors during initialization. Thus, DDP proved ineffective for our use case, and we were forced to explore more memory-efficient alternatives for model training.

So, given the persistent memory constraints encountered during full fine-tuning, we adopted the DeepSpeed and Accelerate libraries to enable more efficient training of large-scale language models. DeepSpeed is a high-performance optimization library developed by Microsoft for scaling deep learning models across multiple GPUs. It provides memory-saving techniques such as ZeRO (Zero Redundancy Optimizer), optimizer offloading, and model parallelism. We specifically used ZeRO Stage 3, which partitions not only the optimizer states and gradients but also the model parameters themselves across all available devices. This allows DeepSpeed to drastically reduce GPU memory consumption during training, enabling full fine-tuning of models that would otherwise not fit into memory.

In addition, Hugging Face Accelerate was used to handle hardware abstraction and distributed environment setup. Accelerate simplifies multi-GPU training by automatically resolving device placement, gradient synchronization, and mixed-precision policies. Its seamless integration with DeepSpeed allows us to easily deploy training scripts with complex hardware configurations, including both

local and remote clusters.
Our training configuration was written in a structured DeepSpeed JSON object, as shown in the excerpt below:

```
ds_config = {
    "bf16": {"enabled": True},
    "zero_optimization": {
        "stage": 3,
        "overlap_comm": True,
        "contiguous_gradients": True,
        "reduce_bucket_size": 1e7,
        "stage3_prefetch_bucket_size": 1e7,
        "stage3_param_persistence_threshold": 1e5,
        "stage3_gather_16bit_weights_on_model_save": True
    },
    "optimizer": {
        "type": "AdamW",
        "params": {
            "lr": 2e-5,
            "betas": [0.9, 0.999],
            "eps": 1e-8,
            "weight_decay": 0.01
        }
    },
    ...
}
```

Figure 3.17: DeepSpeed configuration for ZeRO Stage 3 with mixed-precision and optimizer settings.

This configuration enabled bf16 mixed-precision training for lower memory usage while preserving model performance. The optimizer settings reflect standard AdamW parameters with a learning rate of 2e-5, chosen based on prior literature and common practice in LLM fine-tuning.
We trained the model using Hugging Face's Trainer class, configured with the arguments shown below:

```
training_args = TrainingArguments(
    num_train_epochs=5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    gradient_accumulation_steps=4,
    eval_strategy="steps",
    eval_steps=21,
    save_strategy="steps",
    save_steps=21,
    learning_rate=2e-5,
    bf16=True,
    gradient_checkpointing=True,
    report_to="wandb",
    deepspeed="ds_config.json"
)
```

Figure 3.18: TrainingArguments for Hugging Face Trainer with DeepSpeed integration.

We enabled gradient checkpointing to reduce memory usage during the backward pass, at the cost of increased compute. Additionally, logging and monitoring were conducted via Weights & Biases (W&B), which allowed us to visually track evaluation loss, training progression, and system resource utilization.

To determine the correct frequency for saving and evaluating, we calculated the number of training steps per epoch using the formula:

$$\begin{aligned}\text{Steps per epoch} &= \frac{\text{Total training samples}}{\text{Devices} \times \text{Batch size per device} \times \text{Gradient Accumulation Steps}} \\ &= \frac{5530}{2 \times 8 \times 4} \approx 86\end{aligned}$$

We set eval_steps and save_steps to 21 in order to evaluate and save the model approximately four times per epoch. This gave us fine-grained visibility into the training progression without overwhelming storage or evaluation resources.

Despite leveraging DeepSpeed's ZeRO Stage 3 optimization and running on a cloud setup with upgraded GPU hardware via RunPod, the fine-tuning attempt ultimately failed to yield meaningful improvements. As training progressed, we observed signs of severe catastrophic forgetting. While the training loss initially decreased, the evaluation loss began rising after only one epoch—indicating the model was memorizing and overfitting on the training data without generalizing to unseen examples. The model also quickly lost its original generative capabilities: instead of producing coherent and informative responses, it started repeating phrases or copying chunks of context verbatim.

To try avoiding this behavior, we experimented extensively by combining various subsets of our available dataset and adjusting key hyperparameters such as learning rate, batch size, and gradient accumulation steps. However, the problems persisted. In one instance, when fine-tuning solely on the "easy" QA dataset, the model began responding by repeatedly asking new questions instead of providing answers—further illustrating a collapse in its generative alignment.

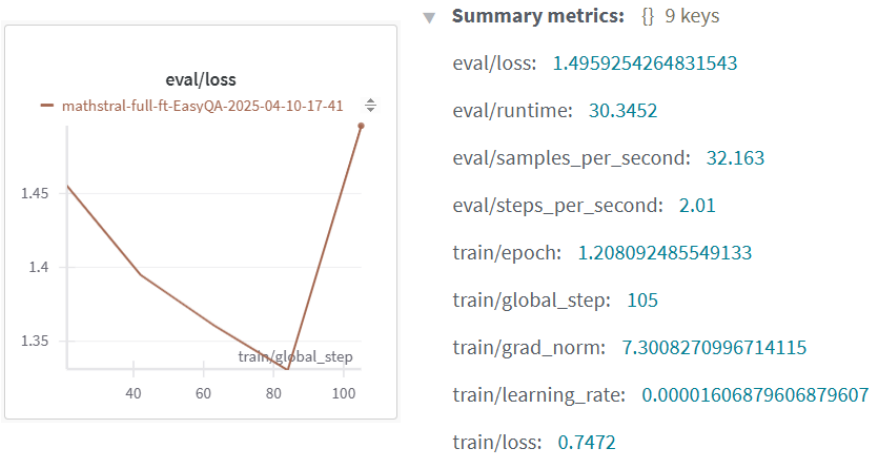


Figure 3.19: Evaluation loss curve and Summary metrics during DeepSpeed-based full fine-tuning of Mathstral-7B.

Several factors may have contributed to this degradation in model behavior. First, there may have been a mismatch in numerical precision: while we trained in bfloat16, the original Mathstral-7B model has been pretrained using fp32, which is not directly supported by deepspeed, introducing instability in gradient calculations. Second, Mathstral-7B is known to be trained on scientific and mathematical data, potentially including content closely related to communication systems in its own pre or post training dataset. This overlap may have caused the model to overfit more rapidly, especially when exposed to fine-tuning data with redundant patterns or low variance.

Due to these challenges—along with the considerable resource requirements of full fine-tuning—we decided to halt the Mathstral-7B experiments and shift our focus toward smaller, more stable models. The next section details our fine-tuning pipeline for the Qwen 2.5-3B model, which proved significantly more successful.

3.3.2 Qwen 2.5-3B Fine-Tuning

Following the challenges encountered with Mathstral-7B, we transitioned to a smaller and more practical model, the Qwen 2.5-3B Instruct. Developed by Alibaba's DAMO Academy and released with open weights, Qwen models are part of a family of decoder-only transformers that have demonstrated strong performance in open-domain reasoning tasks and multilingual instruction following [29]. The "Instruct" variant of Qwen 2.5-3B is specifically fine-tuned for alignment with human instructions and conversational prompts, making it a suitable backbone for our teaching assistant use case.

The switch to Qwen 2.5-3B brought multiple advantages. At approximately 3 billion parameters, the model offered a balanced trade-off, it was significantly

smaller than Mathstral-7B, enabling full training on a single GPU without the need for DeepSpeed, model parallelism, or distributed strategies, yet large enough to retain robust language modeling capabilities. In addition, Qwen's pretraining data did not appear to heavily emphasize telecommunications, reducing the risk of overfitting to domain-adjacent patterns, as had been the case with Mathstral. Lastly, the Qwen 2.5 series includes several models of various sizes, making it easy to expand.

To carry out the fine-tuning of Qwen 2.5-3B, we used the Unsloth library— a lightweight and memory-optimized framework designed for fast training of small to mid-scale instruction-tuned language models. Unsloth supports full fine-tuning of models using the Hugging Face Trainer interface, while handling tokenization, chat prompt formatting, and loss masking internally. However, a key constraint of Unsloth is that it only supports datasets formatted in question–answer (QA) style. As a result, we were unable to utilize the structured informational "chunks" prepared during Phase I of our data pipeline. Instead, we limited our training exclusively to the QA dataset developed in Phase II, which was split into three tiers of difficulty (Easy, Medium, and Hard). For this first experiment, we started on the Easy QA subset.

The model and tokenizer were loaded using Unsloth's optimized wrapper:

```
from unsloth import FastLanguageModel

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/Qwen2.5-3B-Instruct",
    full_finetuning = True,
    max_seq_length = 2048,
    dtype = None,
    load_in_4bit = False
)
```

Figure 3.20: Loading Qwen 2.5-3B with full fine-tuning enabled via Unsloth.

Here, full_finetuning=True enables weight updates across the entire model, rather than using adapter methods. We opted for standard precision by setting dtype=None, allowing Unsloth to auto-detect the model's best format.

Next, the training and validation datasets were loaded from disk. Each entry originally contained a question and its corresponding answer. These were transformed into dialogue-format dictionaries using the following function:

```python
def create_conversation(example):
    return {
        "conversations": [
            {"role": "user", "content": example["question"]},
            {"role": "assistant", "content": example["answer"]}
        ]
    }

train_dataset = raw_train_dataset.map(create_conversation)
val_dataset = raw_val_dataset.map(create_conversation)
```

Figure 3.21: Conversion of question–answer pairs into conversational format.

This format aligns with Qwen's instruction-tuning structure, which expects alternating user and assistant roles. Once created, the original question and answer fields were removed to avoid redundancy.

We then applied Qwen's official chat template using Unsloth's tokenizer utilities. This step converts each structured conversation into a textual prompt the model understands:

```python
from unsloth.chat_templates import get_chat_template

tokenizer = get_chat_template(
    tokenizer,
    chat_template = "qwen-2.5",
)

def formatting_prompts_func(examples):
    convos = examples["conversations"]
    texts = [tokenizer.apply_chat_template(convo,
                tokenize=False,
                add_generation_prompt=False) for convo in convos]
    return { "text" : texts }

train_dataset = train_dataset.map(formatting_prompts_func, batched=True)
val_dataset = val_dataset.map(formatting_prompts_func, batched=True)
```

Figure 3.22: Applying the Qwen 2.5 chat template to structure conversational prompts.

By applying this template, each training example becomes a formatted text string that mimics a real conversation. The role separation and turn-taking are essential for instruction-tuned models to interpret prompts correctly and generate aligned outputs.

Once formatted, the data was ready to be passed into Unsloth's SFTTrainer, a fine-tuning wrapper built on top of Hugging Face's Trainer API. The training configuration was as follows:

```python
from unsloth import SFTTrainer
from transformers import TrainingArguments

trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    dataset_text_field="text",
    args=TrainingArguments(
        output_dir="./qwen-ft-EasyQA",
        num_train_epochs=5,
        per_device_train_batch_size=8,
        gradient_accumulation_steps=4,
        eval_steps=21,
        save_steps=21,
        logging_steps=21,
        learning_rate=2e-5,
        bf16=True,
        report_to="wandb",
        run_name="qwen-ft-EasyQA",
        load_best_model_at_end=True,
        save_total_limit=2,
        remove_unused_columns=False,
        metric_for_best_model="eval_loss",
        greater_is_better=False
    )
)
```

Figure 3.23: Initialization of the SFTTrainer with Hugging Face-style TrainingArguments.

The configuration mirrors our earlier DeepSpeed setups in terms of batch size, epochs, and learning rate. However, the training was much simpler to execute and debug due to the model's smaller size and the memory efficiency of Unsloth. Logging and monitoring were also done through Weights & Biases (W&B) for real-time visualization of training dynamics.

The training process proceeded smoothly, with consistent convergence and no signs of early overfitting. Both training and evaluation loss curves followed a healthy downward trend, confirming that the model was learning progressively and generalizing well across the validation set. This early-stage run—focused on the Easy QA tier—showed that our fine-tuning setup was both stable and effective.

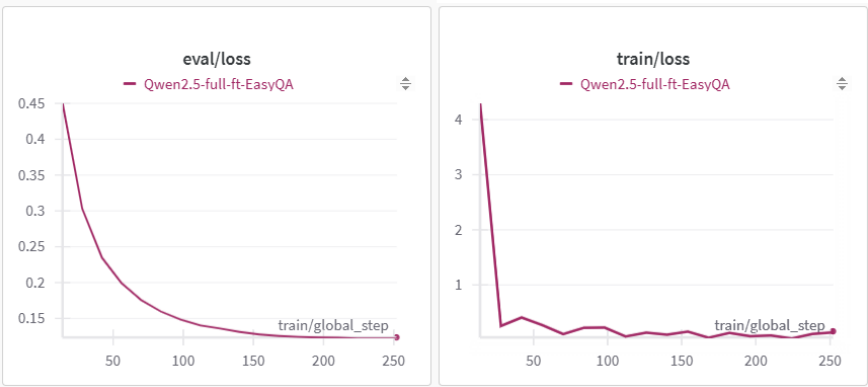


Figure 3.24: Training and evaluation loss curves during fine-tuning of Qwen 2.5-3B on Easy QA dataset.

Following the successful fine-tuning on the Easy QA dataset, we proceeded with the remaining stages of our training pipeline. For each stage, we resumed training from the best checkpoint obtained in the previous run—starting with Medium, followed by Hard. This sequential progression ensured that the model retained its accumulated knowledge while adapting to increasingly complex tasks. The strategy proved both efficient and stable, allowing us to leverage the benefits of curriculum-style fine-tuning.

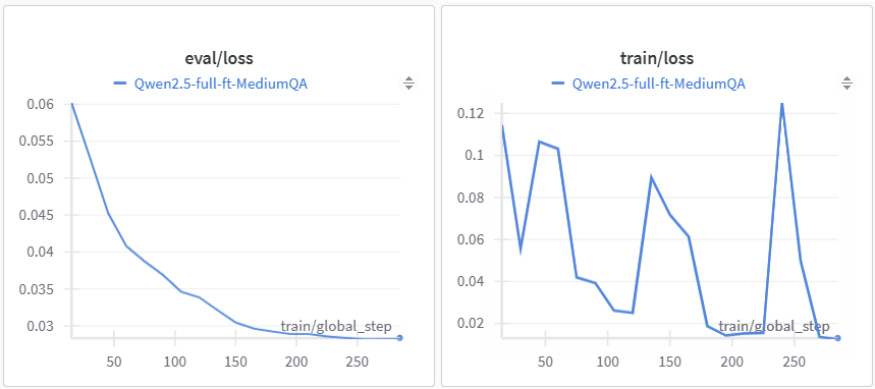


Figure 3.25: Training and evaluation loss curves during Medium QA fine-tuning.

Figure 3.26: Training and evaluation loss curves during Hard QA fine-tuning.

To validate the effectiveness of the fine-tuned Qwen 2.5-3B model, we conducted brief informal tests by prompting it with representative questions from the domain. The model responded coherently and without exhibiting the breakdown issues previously observed with Mathstral—such as repetition, instruction misalignment, or direct copy-pasting of training content. These responses indicated that the model had retained its generative capabilities while integrating the domain knowledge introduced during fine-tuning, and is now ready to be benchmarked. Note that in all the fine tuning efforts above, we extensively had to tune the hyperparameters like learning rate and batch size to get better training.

3.4 Graphical User Interface

To create an accessible, user-friendly interface for our system, we implemented two separate Graphical User Interfaces (GUIs) using Gradio, an open-source Python library that enables rapid deployment of machine learning models as web applications. These GUIs allowed users to interact with our LLMs in real-time and test their performance in a controlled environment, simulating the user experience of a virtual teaching assistant for EECE 442. Gradio was chosen for its simplicity and compatibility with our system, as well as its compatibility with Huggingface which is useful if we choose to share the model there. It allowed us to swiftly create input/output components with minimal overhead and development effort. Also, it allows us to configure our own chat function and store or process what we desire in it. Lastly, Gradio provides us with a clear, interactive view of how user queries are processed, and it is easily updated after testing and changes. After deploying an instance of the GUI, we are able to to handle multiple users at once in a parallelizable fashion (up to 100 users on the A100) for 72 hours until the link is expired. To renew the link we simply have to rerun the code.

System 1: Mathstral-7B with RAG
The first GUI integrates our **Mathstral-7B** language model. This system is connected the RAG pipeline. When a user inputs a query, the GUI passes it through our RAG system, which performs semantic retrieval from the public and private datasets, applies filtering and re-ranking, and appends the top relevant context to the model prompt. The Mathstral-7B model then generates a response based on the augmented prompt.
The backend of this system is powered by a function named chatrag, which handles user inputs and processes them for the RAG pipeline. Specifically, this function constructs a prompt by concatenating a short history (last three exchanges) of the conversation between the user and assistant. It then queries the RAG pipeline using this prompt with predefined parameters, which in turn returns a response that may contain extra formatting, so we apply regular expressions to extract and clean the model's final answer. Only the assistant's reply is returned and shown in the GUI. This modular design allows efficient memory usage and quick response generation, while maintaining context relevance within a recent sliding window.

We used Gradio's `ChatInterface` class to bind this function to a dynamic chatbot component named `"The CommGPT"`, enabling real-time dialogue. It includes a conversational area with markdown disabled (to prevent undesired rendering), example questions for quick testing, and scalable concurrency. An important part of the code is shown below.

```python
def chat_rag(user_message, history):
    conversation_text = ""
    for i, (old_user, old_assistant) in enumerate(history):
        conversation_text += f"User: {old_user}\nAssistant: {old_assistant}\n"

    recent_history = history[-3:]

    response_text, *_ = rag_pipeline.query(
        question=user_message,
        top_k=3,
        auto_tag_filter=False,
        use_mmr=False,
        conversation_history=recent_history,
        use_reranker=True
    )

    # Extract the final answer after '[/INST]'
    answermatch = re.search(r"\[/INST\](.*)", response_text, flags=re.IGNORECASE
        | re.DOTALL)
    if answermatch:
        generated_answer = answermatch.group(1).strip()
    else:
        generated_answer = response_text

    # Clean up any leading or trailing punctuation/spaces
    generated_answer = re.sub(r"^\s*\.", "", generated_answer).strip()

    # Return **only** the assistant's text:
    return generated_answer


gr.ChatInterface(
    fn=chat_rag,
    title="The CommGPT",
    description="An LLM-powered TA for all your communication engineering
        questions.",
    chatbot=gr.Chatbot(height=400, label="Conversation",render_markdown=False),
    concurrency_limit=None,
    theme="default",
    examples=["What is Nyquist rate?", "Why do we use error correction?", "
        Explain OFDM in simple terms."],
    submit_btn=True,
    cache_examples=False,
    fill_height=True
).launch(share=True)
```

Figure 3.27: Gradio Interface Code

This setup is ideal for queries that require heavier mathematical understanding, and due to its larger size, is better equipped to answer out of topic answer if prompted to. The GUI is equipped with a multi-section layout: one for user input, one for the retrieved context displayed in real-time, and another for the model's answer.

System 2: Fine-Tuned Qwen 2.5-3B using RAG

The second GUI leverages the **Qwen 2.5-3B** model, which we fine-tuned on a

three stage QnA dataset specific to EECE 442 and other Communication Systems data, and enhanced with RAG capabilities. This GUI is has the same frontend design as the one for Mathstral-7B.

Ultimately, provided this project passes to a production phase, a single GUI can be kept by integrating it with the best model or configuration.
Below are sample back and forth interactions with our final model with through the GUI, taking the 7B version as an example:

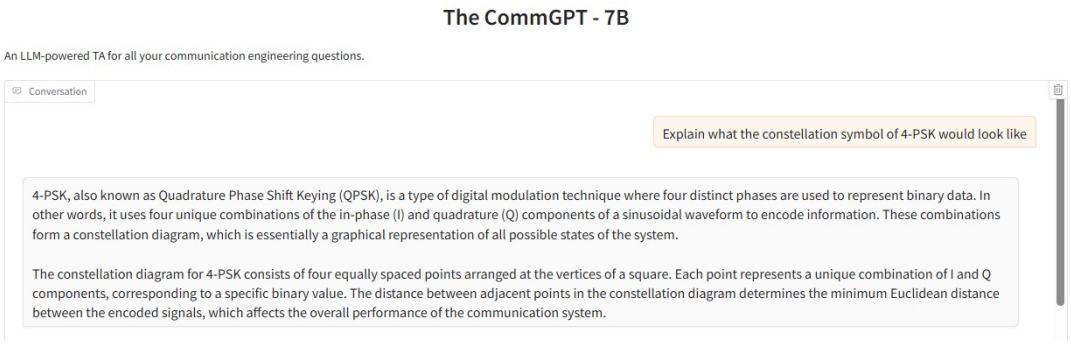


Figure 3.28: GUI interaction with sample conversation Pt.1

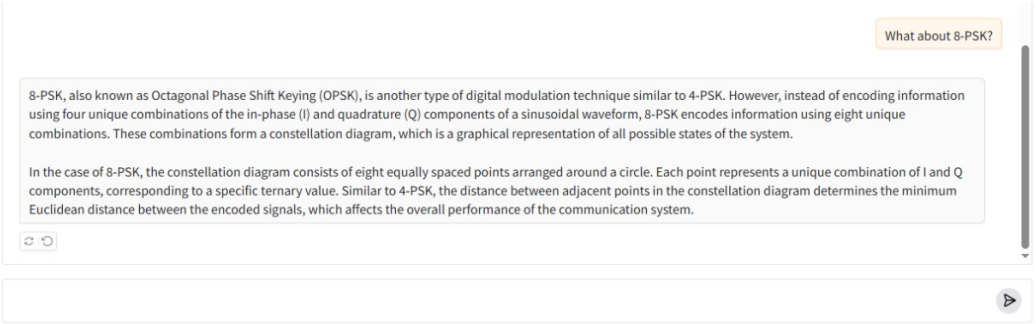


Figure 3.29: GUI interaction with sample conversation Pt.2

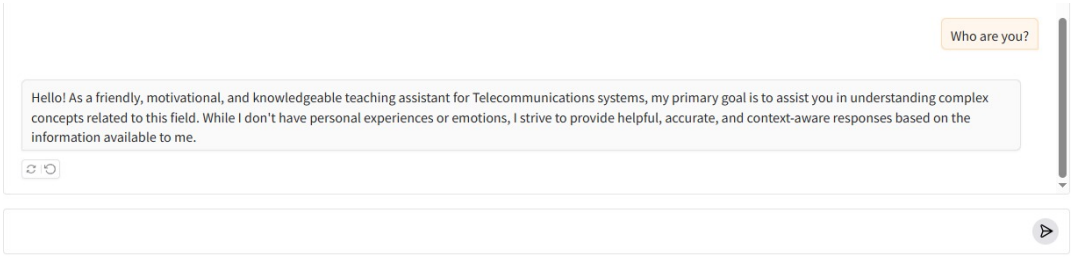


Figure 3.30: GUI interaction with sample conversation Pt.3

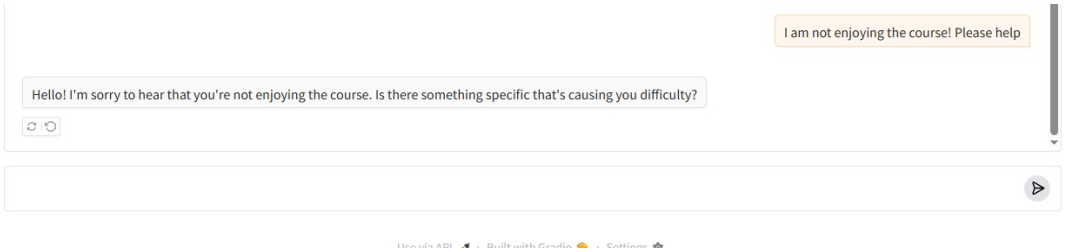


Figure 3.31: GUI interaction with sample conversation Pt.4

Chapter 4

Experimental Setup and Results

4.1 Experimental Setup

HPC and RunPod
As we began the implementation process, we focused on setting up the hardware environment as it was a prerequisite to any development. Initially, we had started learning how to use work on GPU clusters with the HPC of the American University of Beirut which is a free service for AUB students. After extensive testing, two major obstacles prevented us from continuing our development on the HPC, the VRAM limitation of 32GB, and most importantly major dependency conflicts. The first issue we faced was attempting to perform fine tuning of Mathstral 7B model on HPC as it required much more VRAM. However as we quantized the model, we were faced with another problem regarding dependency conflics expecially with CUDA. HPC runs on version of CUDA that are not easily compatible with DeepSpeed and Unsloth, which are imperative for our Fine Tuning effort. Without these libraries, we would have to triple our VRAM usage a luxury we did not have. Therefore, we decided to move to a paid service that would fit our necessary VRAM requirements and storage needs.

We choose **RunPod**, which is a platform that provides GPU rental services and large storage facilities. Such services require a thorough understanding of their services, the hardware we are renting and the software environment.
RunPod offers Linux based servers and provides as much storage as needed paid for on daily use. Furthermore, it offers a large variety of GPU with a broad range of VRAM and architectures. After extensive experimentation by trying to load our models on the VRAM of several GPUs, and after running several trials on Fine Tuning codes and Embedding RAG vectors, it was clear that our project required a minimum of a A100 GPU with 80GB of VRAM to be mounted and for the Fine Tuning, an additional A100 GPU was required to run using **DeepSpeed**. Although, this was the primary step of the RunPod setup, and as

we experimented with several techniques of Fine Tuning, by using **Unsloth** with QWEN, we were able to economize on GPU usage, and settle on using a single A100 GPU for our runs.
On the software side, as we were working on our own project but on a larger hardware cluster, we created our own virtual environment that would include all our libraries installed and all the tools saved on that virtual environment. As mentioned earlier, RunPod allows us to store our project; therefore, we saved our libraries in the virtual environment of our project which would serve for the entirety of our development with a total of 650GB of storage, mostly used to save model checkpoints.

Weights and Biases
Setting up the running environment was the first step to implementation, however having a clear visual on evaluation and progress metric is equally important as having a running code. We looked extensively into a tool called OPIK, which supports the full pipeline of LLM development, from unit testing to LLM as a judge evaluation to deployment and releasing. However, we chose **Weights and Biases**, as it is a more standard tool that was made free to us as AUB students. It helps visualize and store evaluation results of our LLM evaluation runs, and exists as the **Wandb** python library. Hence, we were all set to begin evaluating our progress by tracking the improvement of Fine Tuning and RAG.

Evaluation MCQ set
The evaluation stage is the final stage of each iteration, as we needed to have some metric to check whether our project is going in the right direction. Since we are developing a custom ChatBot, we realized that we had to develop some custom evaluation method that tests two things, how well the model is learning form the EECE 442 focused material, and how well it can generalize to broader communication systems topics or questions.
We began by generating a robust evaluation set in the form of Multiple-Choice Questions (MCQs) based directly or extracted from the cleaned and chunked dataset. Unlike ROUGE, BLEU or BERT, an MCQ evaluation is more interpretable. This evaluation dataset was designed to be used on every stage of development to check the improvement of the model at answering nuanced technical questions and compare the result to previous version of our system.

We decided to collect and generate a diverse MCQ set with a philosophy of diversification of data. This diversification involved generating the MCQ from internal sources that would test the LLM on its ability to learn from the data provided to it through RAG and Fine Tuning. However, in order to ensure that it is not being just overfitting on our provided datasets, we manually created 25 simple MCQ questions that are not from the fine-tuned data or RAG data to

test the generalization ability of our LLM. Furthermore, we collected 75 lexicon question form a TeleQnA dataset. Finally, we collected 55 MCQ question in the field of communications made open source for educational purposes, to be used in addition to the previous three sources of MCQ. Thus, we ended up with 450 MCQ question that will serve as metrics for our progress.

The generation of the internal MCQ questions inspired by our own data chunks was made by iterating over the cleaned chunks of data we had already processed from the public data collected, however, to ensure that the questions are all-encompassing the material, we manually selected the most relevant sources from the public repository collected, in a manner that distributes the number of chunks to be equal between all the chapters of EECE 442, resulting in around 30 to 40 chunk per chapter. Subsequently, these chunks are then passed to the MCQ generation script that sends the chunks via API calls to an external LLM for MCQ generation. While this method of generating the MCQs has its benefits, we were faced with the risk of having the MCQs' answer key not 100% correct. Therefore, to minimize that risk, we instructed the LLM being used to explain the answer step by step and justify its answer (chain of thought prompting). While this still does not guarantee correct answer, it was shown to drastically increase the chances of it getting it right thus minimizing our risk of having faulty answer keys [30].

These MCQ questions were not designed to be easy, rather the opposite, since their goal is to test the limits of the chatbot we are designing and revealing the added knowledge it is acquiring and not the base knowledge it already has.

MCQ Sample:

```
"question 21": {
"question": "Consider a signal z(t) created by multiplying a real-valued
             even signal e(t) with a real-valued odd signal o(t).
             What symmetry property does the Fourier Transform Z(j \Omega)
             of the signal z(t) exhibit?",
"option 1": "Z(j \Omega) is real and even.",
"option 2": "Z(j \Omega) is real and odd.",
"option 3": "Z(j \Omega) is imaginary and even.",
"option 4": "Z(j \Omega) is imaginary and odd.",
"answer": "option 4: Z(j) is imaginary and odd.",
"explanation": "Step-by-step solution:
                Determine the symmetry of the signal z(t) in the time domain.
                -z(t) is defined as the product of e(t) and o(t), where:
                e(t) is real and even
```

```
                    e(-t)=e(t)o(t) is real and odd o(-t)=-o(t).
                    Therefore,
                    z(t)=e(t)*o(t)
                    -z(-t)=e(-t)*o(-t)
                    =e(t)*(-o(t))
                    =-e(t)*o(t)
                    =-z(t)
                    Hence, z(t) is an odd signal.
                    Since both e(t) and o(t) are real-valued,
                    their product z(t) is also real.
                    Now consider the Fourier Transform of a real and odd signal.
                    The FT of a real, odd signal is known to be imaginary and odd.
                    Therefore, the FT Z(j \Omega) of z(t) is imaginary and odd.
                    The correct answer is option 4.",
"chapter": "Discrete Representation of Waveforms"
}
```


The manually written MCQ questions were developed by reading the titles and subtitles of the EECE 442 lecture notes and filtered by level of depth. After filtering the titles, we wrote basic questions about the material from these titles, that way we ensured uniform distribution of the evaluation on all the material of course. The questions were written to the best of our ability and compared to the difficulty of the questions developed by the Gemini form earlier. Since we wrote the questions, we made sure of the correctness by another LLM that confirmed that our own answers were correct. Therefore, we labeled these questions as "External Questions" and added them to the Eval Set.

As for the TeleQnA data, we visited the Git repository of the TeleLLM developers, and took a random sample of 75 questions from the 500 Lexicon-Labeled questions that they had developed. These questions served as a generalization benchmark that would test how well the LLM can get with terminologies is rarely encounters and would show the progress of the LLM in learning these terms as it progresses in fine-tuning and gets excerpts from RAG. Finally, the last pieces of data we added were an open source MCQ set form the internet that we went over manually and ensured aligned with the course material. This set of questions is made to remove any bias in our own judgment. The solving process and perspective on the material is different from the remaining questions and would serve as proof of genuine enhancement in the LLM provided the score of success increases.

Since we had created a custom evaluation set, the score of any given run does not have a recognized standard. Therefore, in order to benchmark our scores, we ran the evaluation set on each Base Model, prior to any Fine Tuning and without using any RAG system. These scores are then compared to every

iteration which will enable us to judge on whether the Fine Tuning worked or the RAG configuration did have some effect.

RAG Evaluation Process
The Evaluation process was the most pivotal piece of development that determined the best configurations and embedding model to use in our system. Evaluation was processed using the MCQ set discussed in the Data Chapter. Visualizing results on weights and biases (wandb) involved creating via our code a project and a run-name for each run of evaluation we conduct and configuring the measurements and charts in-code.

Regarding the metrics, we labeled each MCQ with the chapter is links back to, thus we used the chapter names to compare the performance of each RAG setup chapter by chapter. Additionally, we had included in our dataset four types of MCQ questions, Internal-Data MCQs, External-Data MCQs, Lexicon Chunks and Generalized-Data MCQ. All four are logged on individual charts as well. To get more results, we added charts that separate the Internal-Data MCQ and all the External ones separately as their interpretation mean different thing. Changes on the Internal data show the change in learning from our fine tuning and RAG, but changes in the external shows the effect of either loss of generalization or gain in generalization, and terminology enrichment. Finally, we included a chart that calculated the total score of all metrics. Scoring was done on the basic formula "Correctly Answered Count / Total Questions in Category".

In order to implement the evaluation of our RAG configurations, we set up the LLM wrapper in order to be able to call it on our MCQ sets. We then setup the RAG foundational code that will load the index vectors and fetch the questions to be iterated over. Afterwards, one by one the MCQ questions are appended to the prompt for evaluation and are sent to the LLM for a response. The prompt instructs the LLM to answer in a standard format as seen below:

RAG Evaluation Prompt:

```
{
"question_prompt_template": (
        "Question: {question_text}\n"
        "Options:\n{options_list}\n"
    ),
"rag_prompt_template": (
        "<s> [INST] You are a highly knowledgeable telecommunications expert."
        "Answer the following multiple-choice question accurately
```

```
         using the provided context, if it is relevant."
        "{context} Question: {question}"
        "Once you settle on an answer, write it exactly as
        FINAL_ANSWER:[correct option number] [/INST]"
    ),
}
```

Fine Tuning Evaluation Process
Similar to the RAG evaluation system, the Fine Tuning evaluation process function almost exactly as the RAG evaluation however without the whole retrieval functionality.

Evaluation False Negative Prevention
Since most of our judgment on the runs rely on the score generated, preventing false negative was of high importance to the evaluation process. Originally, after prompting our model with the MCQ and the guidelines to answer it, a Regex function captures the response from the fix format seen above **"FINAL ANSWER:[correct option number]"**.
However, as we are working with small models comparatively, the possibility of the model diverging from its instruction exists and where it might answer in its own way bypassing the expected answer format. This would naturally result in a wrong answer.
Therefore to remedy this issue, we established a safety measure that would check the answer before judging it as false when the regex does not catch the answer. This mechanism employs another LLM as a judge to the answer of our model and based on its interpretation of the answer, deems it correct or incorrect. The Judge LLM is a Gemini model large enough for this task and is prompted via the following function:

```python
def check_mcq_answer(question_json: dict, llm_response: str, max_retries: int = 3) -> int:
    prompt = f"""
                You are an exam grader, and you have to correct the answer of my student. The question is a JSON format with the answer, but my student answered without specifying his option.
                Check his work and tell me if he correctly answered the question or not. Be strict with the evaluation
                Question:
                {json.dumps(question_json, indent=2)}
                Student Answer:
                \"\"\"{llm_response}\"\"\"
                Instructions:
                - Return a JSON with this format:
                {{
                "student response": "correct" or "incorrect",
                "explanation": "your reasoning here"
                }}
                Only return a valid JSON. Do not explain anything outside the JSON.
            """
    for attempt in range(1, max_retries + 1):
        try:
            print(f"Attempt {attempt}...")
            response = model.generate_content(prompt)
            raw_text = response.text.strip()
            json_text = extract_clean_json(raw_text
            #...Remaining Attempt net logic...
```

Figure 4.1: Python function that calls the LLM as a judge for correction

4.2 Results and Discussion

4.2.1 Mathstral RAG Results

Each configuration of our RAG can be calibrated as discussed in the implementation section. Hence, to decide on the best setup for our system, we conducted a series of evaluation on all different combinations of configuration and at different metrics.

1- Before delving into evaluating each configuration, we chose to evaluate a basic RAG setup against the model without any RAG to discern whether the RAG has any added value in the first place to the system. Thus we ran the evaluation setup on both setups, and as we had expected, the RAG significantly outperformed the base model on almost all types of MCQs. The default RAG system we adopted used the TeleLLM as the embedding model.

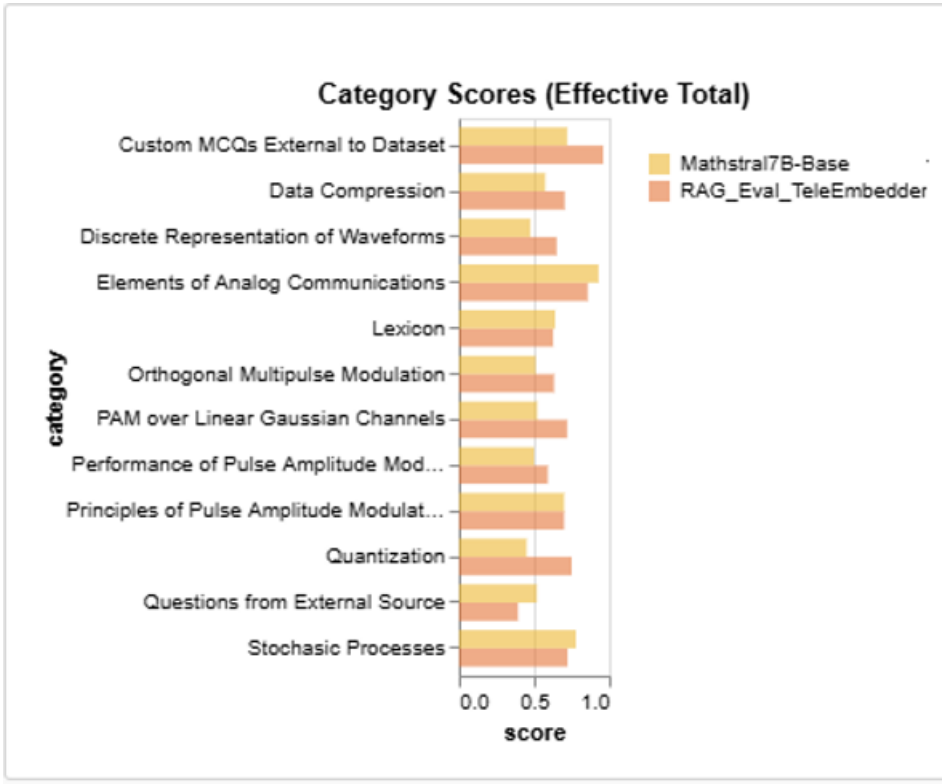


Figure 4.2: Performance of Base Mathstral-7b vs Basic TeleEmbedding Model RAG

Overall, the accuracy of the base model was 52% as for the basic TeleLLM RAG setup yielded a 69% accuracy. This wide difference proved that RAG was the way to go, however what remained was determining the best configuration settings to yield the best possible results.

2- The first evaluation focused on the Re-Ranker, as we ran a basic RAG setup without the Re-Ranker enabled at first. We had charted the results on “weights and biases”:

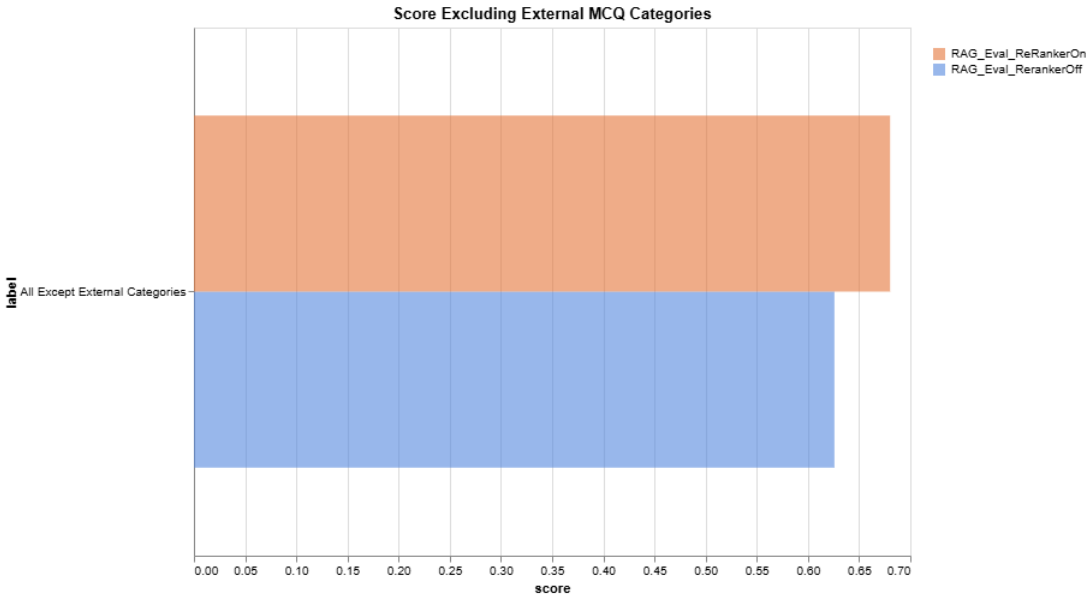


Figure 4.3: Performance of RAG with Re-Ranker vs without Re-Ranker Pt. 1

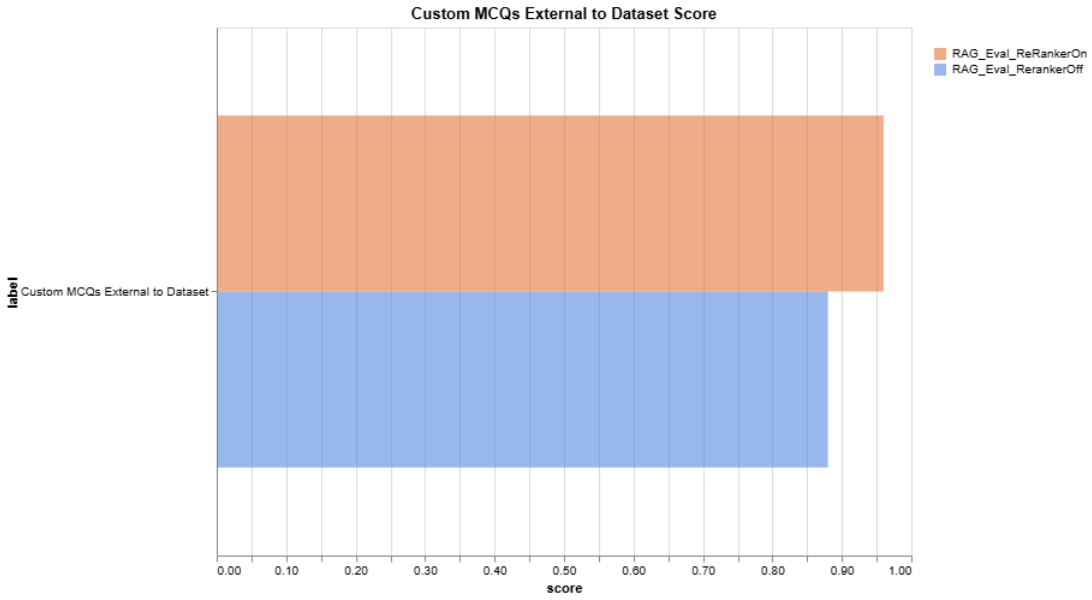


Figure 4.4: Performance of RAG with Re-Ranker vs without Re-Ranker Pt. 2

It can be found that the evaluation with the Re-Ranker enabled performed

better on both Questions based on external data to the RAG chunks as well as the Questions related to the chunks. The Re-Ranker enabled configuration scored a 69% against a 62% accuracy without it on the questions related to the chunks and scored a 96% accuracy on the external questions while without the Re-Ranker it reached 88%. Clearly the Re-Ranker yielded more accurate results.

3- The second configuration we evaluated was the MMR (Maximum Marginal Relevance) as the similarity calculator enabled instead of Cosine Similarity. At first, the results seemed to be similar, however, the Cosine Similarity performed better at the more scientifically and mathematically demanding questions.

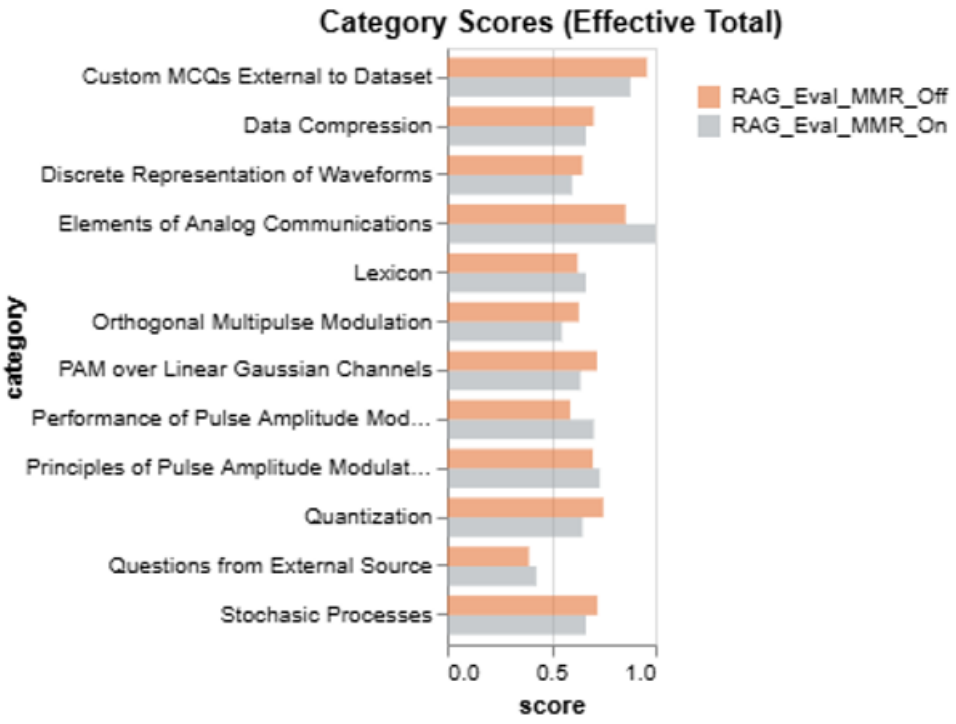


Figure 4.5: Performance of RAG using MMR vs Cosine Similarity

As seen above, the Pink Histogram (Cosine Similarity) had performed better on MCQs based on data external to the chunks, and on all topics with heavy math such as quantization, stochastic processes, and Orthogonal Multi-pulse Modulation and so forth. Although the option to use MMR can be an acceptable choice, we opted for the Cosine Similarity as we deemed it more relevant to have an edge on mathematically demanding subjects than regular textual prompts.

4- The fourth evaluation was targeted at testing the Re-Ranker's number of retrieved chunks kept configuration "k". We ran tests on k=2, k=3 and k=5 chunks per ranked retrieval and compared the results to decide on the best "k".

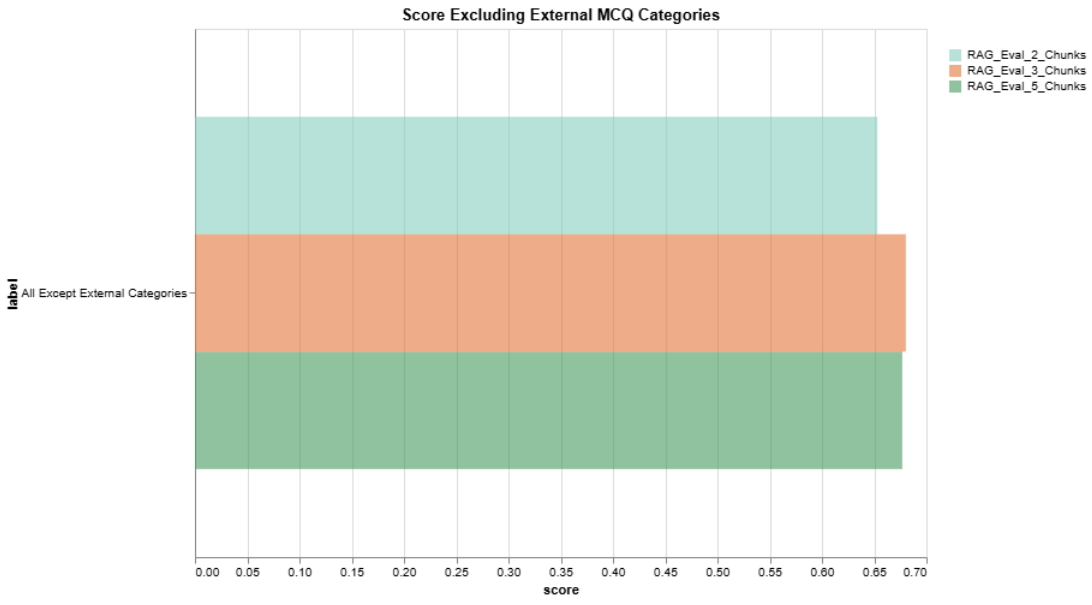


Figure 4.6: Performance of k=2 vs k=3 vs k=5 Pt.1

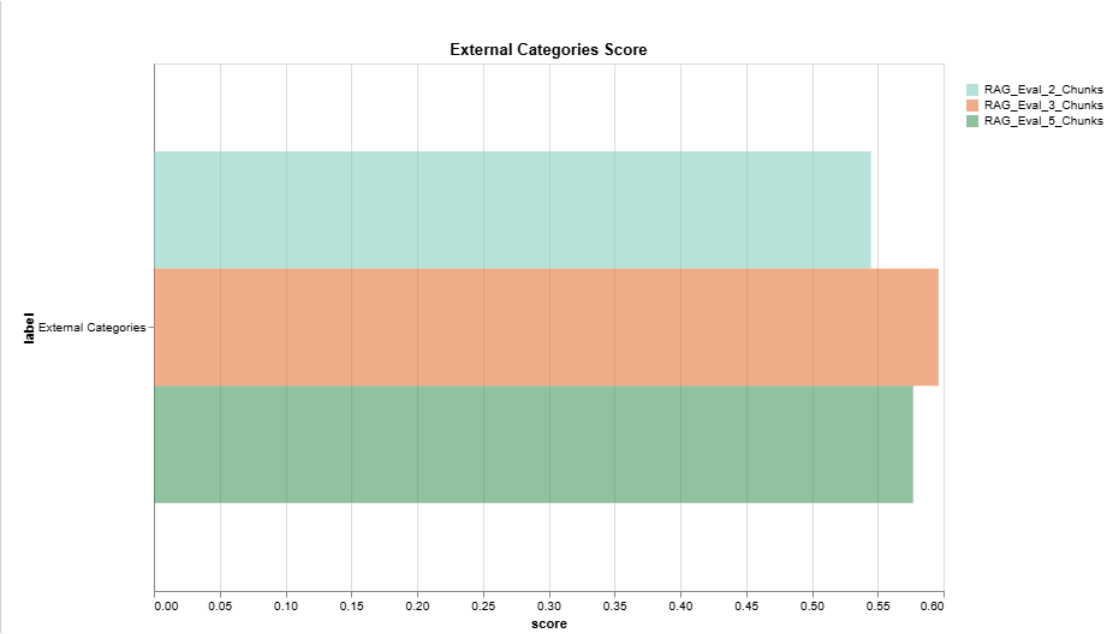


Figure 4.7: Performance of k=2 vs k=3 vs k=5 Pt.2

The graphs above show a slightly better performance on both MCQ external and internal sets for k=3 chunks kept after re-ranking, thus for k=3 the RAG performed best. K=3 scored 67% accuracy on the chunks related questions while for k=5 it yielded 66% and 65% for k=2 as for the question external to the chunks it scored 59% with k=3 chunks against 57% for k=5 and 54% for k=2. As a result we followed through with k=3 chunks with the Re-Ranker.

5- Finally, the final setting to decide was the embedding model of our database. This evaluation test focused on finding which embedding model performs best of our type of data. To successfully evaluate all three models, we had to embed and index our data chunks using all three models "TeleEmbedding Model", "Stella-400M" and "Inf-1.5B". The "TeleEmbeder" is an Embedding Model that we produced from the TeleLLM-Gemma-2B from the TeleLLM series. The Stella400M model is a popular Embedding Model used by LLM system developers all over the world and the INF-1.5B Embedding Model is another popular model used.

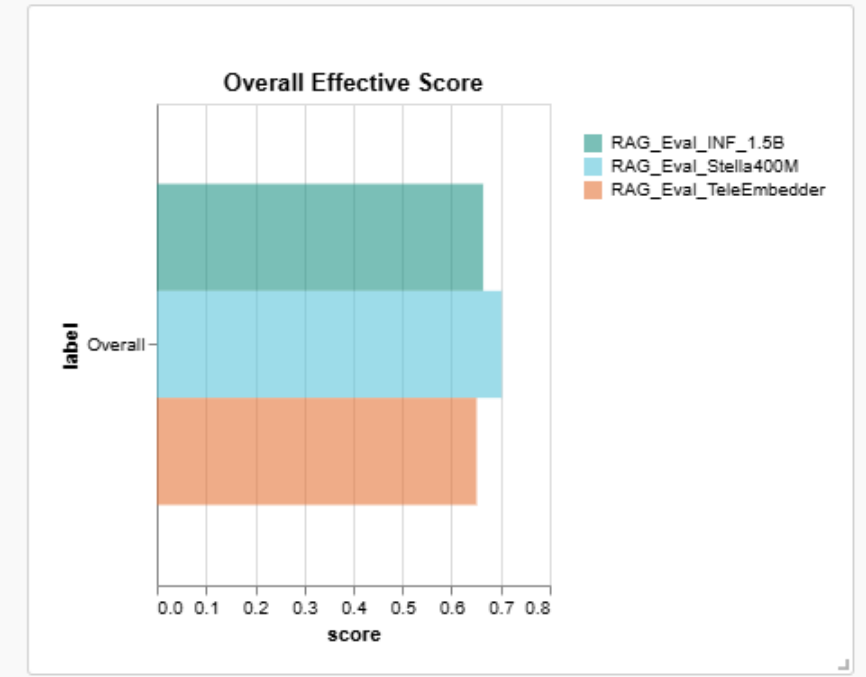


Figure 4.8: Overall Performance of different Embedding Models

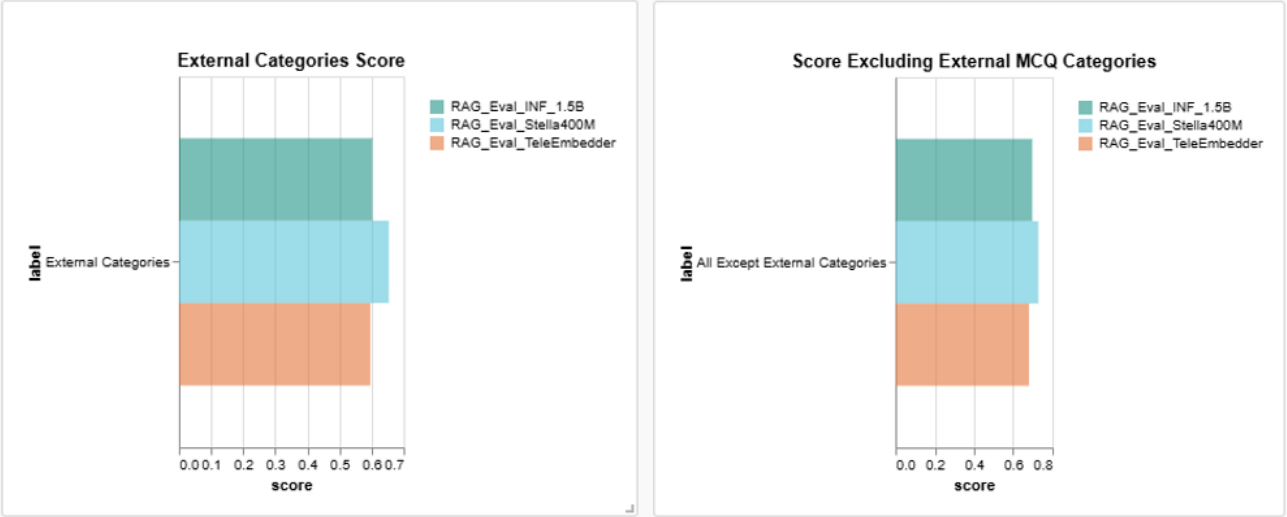


Figure 4.9: Performance of different Embedding Models on different MCQ sets

Although, we had initially expected the TeleEmbedder to perform the best out of the three Stella-400M yielded better results. All three embedding models performed similarly on most categories, however Stella-400M clearly outperformed the other two models with MCQs based on external data and

did seemingly similar but slightly better on the MCQs based on internal data, which concluded our decision on using Stella-400M.

RAG Evaluation Result Conclusion

Therefore, after testing all configuration setups, we were able to scientifically select the best configuration for our RAG which yielded significant results compared to the base model as is. We settled on using Stella-400M embedding model, with Re-Ranker enabled at k=3 chunks and using Cosine Similarity instead of MMR.

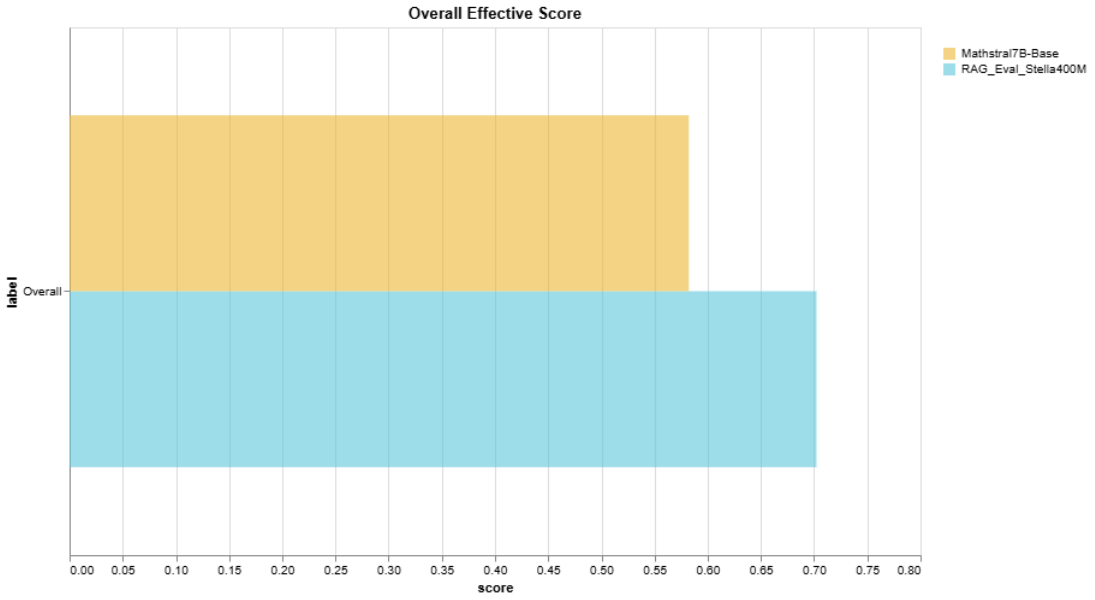


Figure 4.10: Performance of best RAG configuration vs Base Mathstral 7B Pt.1

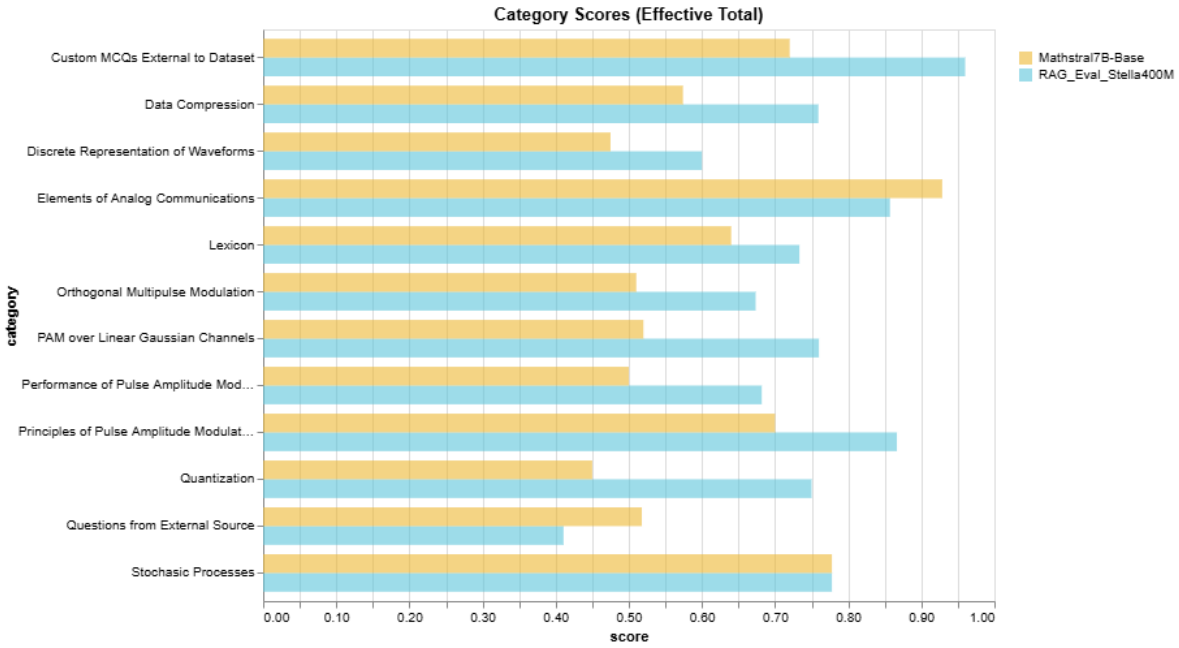


Figure 4.11: Performance of best RAG configuration vs Base Mathstral 7B Pt.2

4.2.2 Qwen Fine Tuning Results

In this section, we present the outcomes of our multi-stage fine-tuning on Qwen 2.5-3B. We first summarize key evaluation metrics across the three phases (Easy, Medium, Hard), and then discuss the implications.

Using the same evaluation function described earlier, we report the following overall effective score results:

- **Overall Effective Score:**
  - Base model: 57.0 %
  - After Easy QA fine-tuning: 57.5 %
  - After Medium QA (initialized from Easy): 58.2 %
  - After Hard QA (initialized from Medium): 59.7 %
  - Hard QA + RAG augmentation: 61.3 %

These results show a steady gain achieved after each and every fine-tuning stage, with an additional +1.6 % improvement when applying our RAG pipeline to the best (Hard) checkpoint.

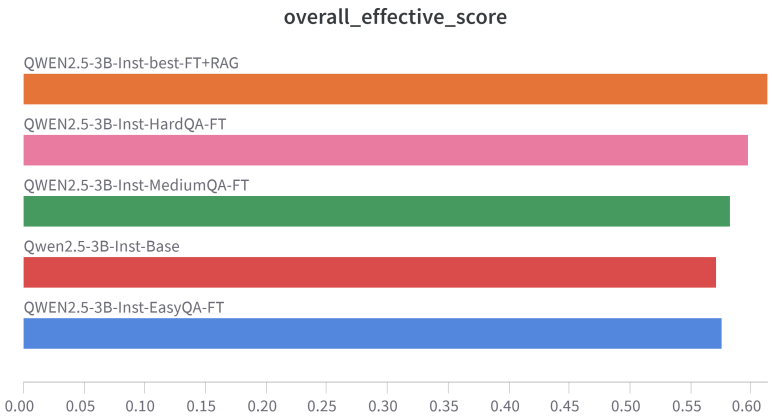


Figure 4.12: Overall effective score at each fine-tuning stage and after RAG.

The next figure visualizes the progression of the lexicon subscore across each fine-tuning stage and following RAG augmentation.

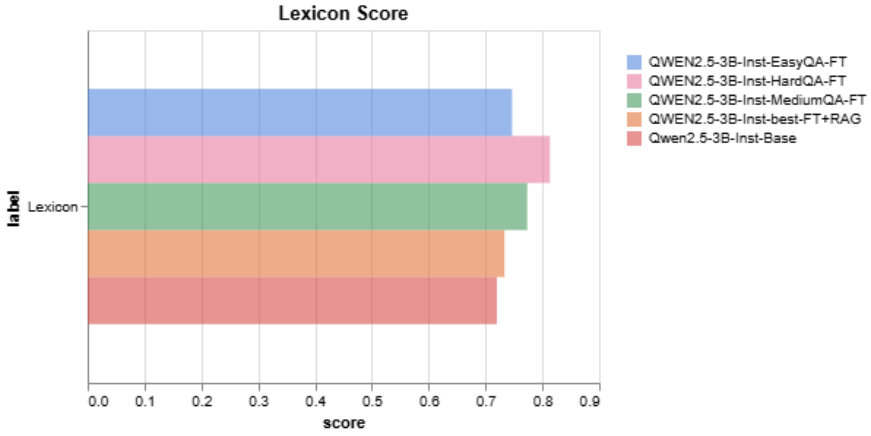


Figure 4.13: Lexicon subscore across fine-tuning stages and RAG.

Initially, we added the lexicon questions as a guardrail against overfitting to our specific data, as they ask general telecommunications questions. While it improved consistently through the curriculum stages, the addition of RAG caused a drop in this subscore—likely due to RAG's emphasis on specific EECE 442 data.

Finally, Figure 4.14 shows the "category score" effective score—a composite across all evaluation categories—highlighting the balanced improvements from factual recall to reasoning tasks.

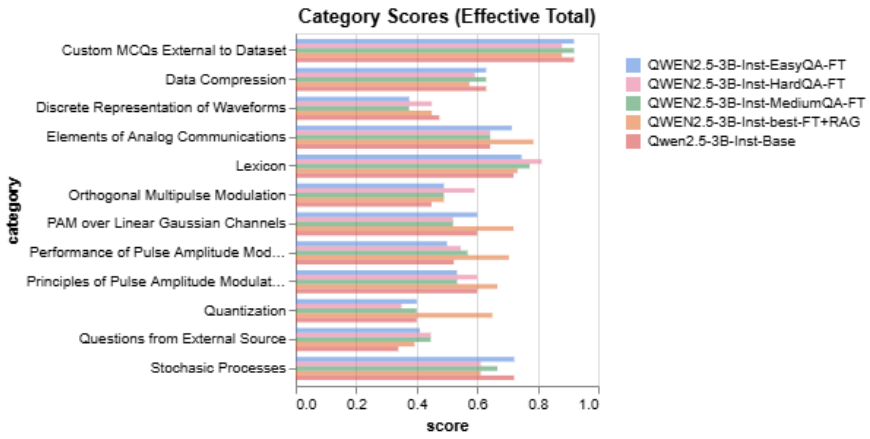


Figure 4.14: Category-wise effective score after each fine-tuning stage and RAG.

While our multi-stage fine-tuning pipeline delivered consistent, incremental improvements at each difficulty level, the relatively modest gains underscore the inherent limits of a 3B-parameter model when addressing complex, mathematically rigorous queries. At the same time, these steady improvements confirm that our curriculum-style fine-tuning approach—progressing from Easy through Medium and Hard QA stages—effectively imparts domain knowledge and reasoning skills to the base model. Research on neural scaling laws and emergent abilities demonstrates that increasing model capacity often leads to qualitatively new capabilities—such as improved long-range reasoning, robustness to out-of-distribution inputs, and enhanced generalization on dense technical content [3, 31].

4.3 Future Work

While the current implementation of CommGPT represents a strong foundation for an intelligent and domain-specialized digital teaching assistant, there remains substantial room for future development and improvement.

First, an important avenue is the incorporation of Reinforcement Learning from Human Feedback (RLHF). This technique has been shown to significantly improve model alignment with human preferences, especially for tasks requiring ethical considerations and pedagogical nuance [17]. Through RLHF, the system can be guided to generate responses that are more appropriate for diverse student learning styles and more cautious in the information it presents, especially in scenarios where technical inaccuracies may lead to misunderstandings.

Another promising enhancement involves fine-tuning the chatbot using full-length conversations between students and teaching assistants. While our current

dataset includes structured Q&A pairs and factual content, conversations that mimic real tutoring sessions could significantly improve the model's capability to emulate natural and educational interactions.

Moreover, the integration of reasoning-optimized models could enhance the model's capacity to solve complex, multi-step engineering problems. Models like Mathstral 7B have demonstrated strong performance in mathematical reasoning, but future work could explore models explicitly designed for logical coherence and stepwise deduction, further supporting the educational guidance objective.

An additional technique to consider is RAFT [9], a recent development that enhances a model's ability to utilize retrieved context during generation. Unlike traditional fine-tuning, RAFT directly trains the model to rely on retrieved documents, improving factuality and grounding in RAG-based systems. Applying RAFT could result in more precise and context-aware outputs, especially in the tightly scoped domain of communication systems.

To improve robustness, query preprocessing and postprocessing techniques can be introduced. Preprocessing could involve query rewriting and correction, while postprocessing could improve output readability, relevance, or structure through summarization or formatting. Enhancing the RAG module itself is another direction. This includes improvements in document processing, particularly in how equations and numerical expressions are handled. Currently, the system treats equations as plain text, potentially diluting their semantic value. Advanced preprocessing methods, including LaTeX-aware tokenization and semantic math parsing, could improve embedding and retrieval precision.

Another key improvement lies in broadening and diversifying the dataset. While our current data is curated and course-specific, expanding it with general communications engineering material can help reduce overfitting and improve generalization.

To further enhance model capabilities, scaling up to larger LLMs should be considered. Larger models have shown empirically and theoretically superior performance in both generalization and contextual reasoning tasks [3]. Deploying as well as fine tuning the 32B parameter version of current Qwen could significantly improve performance across complex tasks.

The current system architecture is based on a single-agent paradigm. However, future versions of CommGPT could benefit from the implementation of multi-agent systems, where specialized agents handle subtasks such as reasoning, retrieval, verification, and pedagogy. As previously discussed, while frameworks like AutoGen [8] offer infrastructure for multi-agent collaboration, we suggest a

custom implementation to allow fine-grained control and minimize hallucinations in this scientific context.

We also propose integrating external tools, such as Wolfram Alpha or SymPy, to enable symbolic computation, visualization, and equation solving. This would expand the assistant's capabilities in numerical problem solving and offer more concrete educational support.

In terms of ethical and privacy considerations, the system could implement data anonymization mechanisms, ensuring that all logged or stored queries are scrubbed of personally identifiable information (PII) and sensitive content. This will become increasingly important if CommGPT is deployed at a larger scale.

Lastly, the system could include an analytics dashboard for instructors, enabling professors to view aggregate statistics about system usage. Metrics such as commonly asked questions, peak usage times, and difficulty areas could provide valuable feedback for improving course delivery and identifying knowledge gaps among students.

By addressing current limitations and expanding its capabilities, CommGPT can evolve into a comprehensive educational tool that better supports engineering students and faculty alike.

Chapter 5

Broad Impact: United Nations SDGs

Impact and Relevance to UN Sustainable Development Goals (SDGs)

Our Final Year Project aligns with UN Sustainable Development Goal 4: Quality Education, which aims to "ensure inclusive and equitable quality education and promote lifelong learning opportunities for all." More specifically, it supports Target 4.3, which emphasizes improving access to quality technical and higher education.

By developing the AI-powered TA tailored for EECE 442 Communication Systems, our LLM system enhances the learning experience for students by providing accessible and domain-specific academic support. This not only helps bridge the gap between students' needs and instructor availability, but also makes high-quality educational support more accessible to all students particularly for students who may struggle to keep up with the course's pace or complexity.

In doing so, our project promotes equitable access to learning resources within the American University of Beirut and potentially other institutions, thus contributing to the global movement toward more inclusive, tech-driven higher education.

Moreover, this project serves as a scalable proof of concept that can be adapted to a wide range of courses and educational institutions, including those where academic support and instructional quality are limited, especially in developing countries, thereby narrowing educational disparities and further advancing the global mission of "Quality Education".

Chapter 6

List of Resources and Engineering Tools

The tables below list the tools, software, and services we used during the entire process including experimentation, prototyping and implementation.

Table 6.1: GPU Services and Models Used for Development

Tool or Service Name	**Description**	**Use Case**	**Company**
RunPod	Cloud GPU Platform	Model Fine Tuning, Embedding and Inference	RunPod Inc.
AUB-HPC	University High-Performance Computing cluster	Basic Training and Experimentation	American University of Beirut
Google Colab	Cloud-based Jupyter Notebooks with free GPU access	Prototyping, Testing and Experimenting	Google
Weights and Biases	Experiment tracking and visualization tool	Logging LLM eval metrics and charts and Weights	Weights & Biases Inc.
HuggingFace	Model and Dataset hosting platform	Base and Embedding Models Access	Hugging Face Inc.
Mathstral	Open-Source Mistral Based Model	Base Model used for our main Chatbot	Msitral AI
cross-encoder/ms-marco-MiniLM-L-6-v2	Transformer-based cross-encoder trained for passage reranking on MS MARCO dataset	Used for reranking retrieved chunks in the RAG pipeline	UKP Lab, TU Darmstadt
Qwen	Foundation language model series by Alibaba	Evaluation for fine-tuning benchmarks	Alibaba Group
Stella-400M	Lightweight Transformer model (EN v5)	Selected Embedding Model for RAG	Dun Zhang
Gemini-2.0-flash-thinking-exp	Experimental version of Google's Gemini	Used to help with our public data processing	Google DeepMind
TeleLLM Series	LLMs specialized in telecommunications	Embedding and baseline testing	Ali Maatouk, Kenny Chirino Ampudia, Rex Ying, Leandros Tassiulas
Inf-retriever-v1-1.5b	Retrieval-optimized transformer model	Embedding benchmark in RAG testing	Junhan Yang, Jiahe Wan, Yichen Yao, Wei Chu, Yinghui Xu, Yuan Qi

Table 6.2: Software and Libraries Used for Development

Tool or Service Name	**Description**	**Use Case**	**Company**
Marker	AI-powered PDF-to-Markdown parser using OCR and language models	Converting academic PDF content into Markdown for LLM training	Vik Paruchuri (GitHub)
Gradio	Python library for building interactive UI for ML models	Serving, testing, and demoing the LLM and RAG system via web interface	Hugging Face Inc.
Unsloth	A lightweight fine-tuning optimization library for LLMs	Accelerated fine-tuning of large language models with less memory and compute	Unsloth AI (Open-source)
DeepSpeed	Deep learning optimization library for distributed training	Efficient model training with techniques like ZeRO offloading and mixed precision	Microsoft

Appendix A

Weekly Meeting Minutes

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
1	1/21/25	3:45 pm	60 minutes	○ Advisor ◉ Students	David Abboud

Attendees:

David Abboud, Alex Eid, Alexander Menassa, Sally Choker, Dr. Fadi Zaraket

Briefly summarize the main discussions during the meeting:

- discussed the team's progress on the data collection, as well as our plans for cleaning, division, and treatment of data.

- received guidance on how to use the HPC as well as using department funding on LLAMALabs.

- Miss Choker helped us fix some bugs we had on the HPC and in our code.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- Prof. Zaraket suggested we mask the latex equations, figures, tables, and charts.

- He also suggested we look for Q&A data related to communications online.

- Miss Chokr suggested we use a tool like Markup or Nougat to process the pdf data into Markdown format. It also uses some small LLMs to improve quality.

- We should start figuring out exactly how we want to do fine-tuning.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
Help on finalizing data cleaning and formatting.	David Abboud	Feb 7
Help on finalizing data cleaning and formatting.	Alexander Menassa	Feb 7
Help on finalizing data cleaning and formatting.	Alex Eid	Feb 7

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
2	2/3/25	6:30 pm	60 minutes	○ Advisor ◉ Students	Alexander Menassa

Attendees:

Professor Jihad Fahs, Professor Ibrahim Abou Faycal, Ms. Sally Choker, David Abboud, Alex Eid, Alexander Menassa.

Briefly summarize the main discussions during the meeting:

- RAG: Course overview and using prompting to enhance system performance.

- LLM Selection Experiment: Evaluated models using the TeleQnA dataset. TeleLLM failed to follow format, Mathstral was faster but lacked reasoning, DeepSeek performed well but was slower. Lexicon Score is the best evaluation metric.

- QnA Dataset Challenge: Needed for making the LLM conversational, but no available datasets found.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- TeleLLM needs further analysis. DeepSeek and Mathstral are top candidates, a hybrid approach might improve performance.

- Further evaluation is needed to understand model weaknesses.

- Data collection continues, with a focus on LaTeX for efficiency.

- More sources are needed for QnA data.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
Finish Data Cleaning + Work on RAG + 442 Benchmark	David Abboud	Feb 14
Finish Data Cleaning + Work on RAG + 442 Benchmark	Alexander Menassa	Feb 14
Finish Data Cleaning + Work on RAG + 442 Benchmark	Alex Eid	Feb 14

American University of Beirut
EECE 502 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
3	2/15/25	6 pm	1 hour	○ Advisor ◉ Students	Alexander Menassa

Attendees:

Alexander Menassa- David Abboud - Alex Eid

Briefly summarize the main discussions during the meeting:

How to proceed with QnA translation to conversations for Instruct Training The cleaning process for QnA and factual Data updated. RAG procedure and its evaluation.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

QnA will be generated using problems and quiz questions into metadata and then conversations. Cleaning will prioritzie QnA data after collectiona and then proceed with the factual data RAG implementation is still in progress.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
QnA processing	Alexander Menassa	22 Feb
Cleaning Effort	Alex Eid	22 Feb
RAG embedding and implemention with Eval	David Abboud	22 Feb

American University of Beirut
EECE 502 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
4	2/17/25	6 pm	1 hour	○ Advisor ◉ Students	Alexander Menassa

Attendees:

Alexander Menassa- David Abboud - Alex Eid - Sally Choker - Profressors Jihad Fahs, Ibrahim Abou Faycal and Fadi Zaraket

Briefly summarize the main discussions during the meeting:

We realized that we had diverged slightly from our original plan with the discussions style and other types of conversatinoal training.

We will stick to our original plan and prepare QnA data to use for training our models with lexicon and more.

There are many advanced techniques to train and teach the mdoel to become more of a chatbot.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

QnA will be generated usingthe RAG into questions and answers.

We will continue with the cleaning process of the data.

Setup training environments and train the model with.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
QnA Script generation and dataset completion.	Alexander Menassa	28 Feb
Continue with the cleaning of the data	Alex Eid	28 Feb
Investigate how to train our 7B Model on HPC or Lambda and try a model.	David Abboud	28 Feb

American University of Beirut
EECE 502 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
5	2/24/25	6 pm	40 minutes	○ Advisor ◉ Students	Alex Eid

Attendees:

Alexander Menassa- David Abboud - Alex Eid - Sally Choker - Profressors Jihad Fahs, Ibrahim Abou Faycal and Fadi Zaraket

Briefly summarize the main discussions during the meeting:

There still is some data cleaning to do.
We started chunking the data into appropriate usable sizes.
We looked at Gemini API keys to generate good quality QAs using our chunked data.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

We should continue cleaning the data.
We should generate different levels of questions, easy, medium and hard, to finetune the model at different difficulty stages.
For RAG, we should try out different embeders and pick the best one.
We should come up with a good meta-data stucture to use.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
QnA Script generation and dataset completion.	Alexander Menassa	7 March
Start investigating how to train our 7B Model on HPC or Lambda and try a model.	Alex Eid	7 March
Continue investigating how to train our 7B Model on HPC or Lambda and try a model.	David Abboud	7 March

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
6	3/3/25	7:00 pm	60 minutes	○ Advisor ◉ Students	David Abboud

Attendees:

Professor Jihad Fahs, Professor Ibrahim Abou Faycal, Ms. Sally Choker, David Abboud, Alex Eid, Alexander Menassa.

Briefly summarize the main discussions during the meeting:

- We discussed our progress in the past week regarding the Q&A generation and fine tuning code.

- Miss Choker recommended that we do the fine tuning in stages, and we also include the cleaned chunks of the data in the first stage. These same chunks will also be available as support for the accompanying Information retrieval system.

- We now have over 20,900 Q&A pairs (equally divided among easy, medium, hard), as well as over 2300 Document chunks around 1000 tokens each.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- We need to proceed with fine-tuning our chosen LLM on the Collected and processed data, and with our customized code.

- We have effectively generated our whole dataset so far using the Gemini Flash 2.0 experimental model.

- Dr. Zaraket suggested we try out Lambda labs, and that we carefully evaluate the effectiveness of our system.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
Work on Domain Fine Tuning + Develop the main retriever + Integrate Dr. Abou Faycal's Data	David Abboud	Mar 15
Work on Domain Fine Tuning + Develop the main retriever + Integrate Dr. Abou Faycal's Data	Alexander Menassa	Mar 15
Work on Domain Fine Tuning + Develop the main retriever + Integrate Dr. Abou Faycal's Data	Alex Eid	Mar 15

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
7	3/10/25	7:00 pm	60 minutes	○ Advisor ◉ Students	David Abboud

Attendees:

Professor Jihad Fahs, Professor Ibrahim Abou Faycal, Ms. Sally Choker, David Abboud, Alex Eid, Alexander Menassa.

Briefly summarize the main discussions during the meeting:

- We discussed our progress in the past week regarding the fine tuning code.

- We have now transferred to Runpod for fine tuning after Llambda labs did not work and hpc was time consuming.

- We have successfully finetuned on the 2500 document chunks we have gathered using QLora, BnB Quantization to 8 bits, and over 300 epochs saving the checkpoints locally and by the help of wandb tool (which is free for AUB students).

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- It was suggested by Sally that we try without Quantization to perform a full fine tuning, and for less epochs.

- We can then proceed accordingly to the further fine tuning.

- We were warned about the Catastrophic forgetting phenomenon in LLMs.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
Work on Domain Fine Tuning + Develop the main retriever + Integrate Dr. Abou Faycal's Data	David Abboud	Mar 24
Work on Domain Fine Tuning + Develop the main retriever + Integrate Dr. Abou Faycal's Data	Alexander Menassa	Mar 24
Work on Domain Fine Tuning + Develop the main retriever + Integrate Dr. Abou Faycal's Data	Alex Eid	Mar 24

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
8	3/21/25	12:00 pm	90 minutes	○ Advisor ◉ Students	David Abboud

Attendees:

Ms. Sally Choker, David Abboud, Alex Eid, Alexander Menassa

Briefly summarize the main discussions during the meeting:

We have tested several fine tuning setups:
- single A100 GPU: QLora fine tuning on around 1.2% of the weights of the model, Quantized to 8 bits.
- 4,6,and 8 A40s, using DDP (Distributed data parallelism): can perform Qlora. however, for full fine tuning without quantization, Sally informed us that in DDP the model needs to fit on each individual GPU, thus explaining why it kept giving CUDA out of memory.
- tried to do full fine tuning on a quantized model, but discovered it does not allow directly.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- To proceed, we will follow up on the funding we had received from the department in order to use it on runpod.io.

- We have identified that a GPU like the H200 is suitable, as it fits the model and allows for full fine tuning.

- We could also explore fine tuning on a pre-quantized model from huggingface, like https://huggingface.co/legraphista/mathstral-7B-v0.1-IMat-GGUF.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
Continue working on fine tuning and follow up on funding.	David Abboud	Mar 29
Prepare the Evaluation dataset that we will use to check for improvement before and after each fine tuning phase.	Alexander Menassa	Mar 29
Finish preprocessing Dr. Abou Faycal's data as well as begin optimizing the RAG setup.	Alex Eid	Mar 29

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
9	3/26/25	7:15 pm	90 minutes	◉ Advisor ○ Students	Alexander Menassa

Attendees:

Ms. Sally Choker, David Abboud, Alex Eid, Alexander Menassa, Professors Jihad Fahs and Ibrahim Abou Faycal

Briefly summarize the main discussions during the meeting:

We started developement of Fine Tuning, however we faced size constraints that are in the process of beign solved.

We also started working on the RAG system however we did not complete the developement.

We developped the Evaluatio Dataset that will be used to evaluate the progress of our fine tunign and rag developements.

We starting processign the data from EECE 442 for RAG.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

We should focus on making the RAG system operational and the fine tuning code.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
Continue working on fine tuning and help with RAG developement.	David Abboud	Mar 29
Work on the RAG system code and the comparison of several systems of RAG to pick the best one.	Alexander Menassa	Mar 29
Finish preprocessing Dr. Abou Faycal's data and support the Fine Tuning Effort.	Alex Eid	Mar 29

American University of Beirut
EECE 501 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
10	4/4/25	5:00 pm	45 minutes	○ Advisor ◉ Students	David Abboud

Attendees:

David Abboud, Alex Eid, Alexander Menassa

Briefly summarize the main discussions during the meeting:

- created a GUI for the project using Gradio 5 and integrated it with the model and RAG. Gradio is a state of the art tool compatible with huggingface and langchain.

- our main RAG code is now ready, and we are in agreement on the different configurations that we want to test to have a thorough understanding of the performance.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

- agreed on the upcoming fine tuning task and planned it.

- discussed the new funding we have received and planned its allocation.

- set the target for our RAG improvement, regarding the lost question issue as well as the different configurations we must test to arrive at the best one to be used in the final system.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
- will fix the bugs in the fine tuning code and make sure it is ready to start + assist in interpretting the RAG evaluation.	David Abboud	Apr 12
- in charge of the RAG benchmarking and configuration testing and analysis.	Alexander Menassa	Apr 12
- will lead the fine tuning effort, which will happen over 4 stages.	Alex Eid	Apr 12

American University of Beirut
EECE 502 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
11	7/4/20	6:30 pm	40 minutes	○ Advisor ◉ Students	Alex Eid

Attendees:

Alexander Menassa- David Abboud - Alex Eid - Sally Choker - Profressors Jihad Fahs, Ibrahim Abou Faycal and Fadi Zaraket

Briefly summarize the main discussions during the meeting:

We discussed the progress with the RAG, the results are good.
We had some issues with the finetuning, it is overfitting after a low amount of epochs.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

Dr. Zaraket not to take the overfit model.
Ms. Sally suggested finetuning another model.
Try temperature = 0 for the RAG,
also try packing instead of padding

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
Work on the Paper	Alexander Menassa	16 April
Continue with Finetuning	Alex Eid	16 April
Continue with Finetuning	David Abboud	16 April

American University of Beirut
EECE 502 – Final Year Project
Course Coordinator – Dr. Youssef Tawk
Minutes for the Weekly Group Meeting – Submitted per Group

Meeting #	Date:	Time:	Duration:	Meeting called by:	Minutes Taker:
12	4/15/25	6:30 pm	1:30	(•) Advisor ( ) Students	Alexander Menassa

Attendees:

Alexander Menassa - Alex Eid - David Abboud - Prof. Jihad Fahs - Prof. Ibrahim Abou Faycal - Prof. Fadi Zaraket - Sally Choker

Briefly summarize the main discussions during the meeting:

- We gave updates on our progress: We had completed the work on the GUI that integrates the RAG and the frontend. What is left is plugging in the FT model.
- We shown the evaluation tests we conducted to select the RAG setup that works best for the system.
- We shown the additionla evaluation categories we had added to have a fully rounded eval set.
- We raised the issues we were facign regarding the FT process in terms of DeepSpeed and other blockers.
- The advisors helped us plan and have realistic expectation on the timeline and other deadlines.
- Prof. Zaraket pointed out some potential reasons for our issues and presented us with couple solution to test out.

Briefly summarize the conclusions drawn regarding the above-mentioned discussions:

We have to finisht the FT the fastest possible by applying the possible solutions we were presented with.
Also we have to be swift to meet the deadlines.

Enumerate the assigned tasks by the FYP advisor for each student + the deadline for delivery:

Assigned Task / Per Student	Name of the Student	Deadline
Work on the Fine Tuning Effort to solve the issues	David	Wednesday / Sunday
Work on the Fine Tuning Effort to solve the issues	Alex	Wednesday / Sunday
Continue work on the Report and support the team if needed	Alexander	April 26

Appendix B

FYP Poster

The CommGPT

Students: Alexander Menassa, David Abboud, Alex Eid.
Advisors: Professor Jihad Fahs, Professor Ibrahim Abou Faycal, Professor Fadi Zaraket, Ms. Sally Choker.
Electrical and Computer Engineering Department, Maroun Semaan Faculty of Engineering and Architecture, The American University of Beirut

PROBLEM DEFINITION

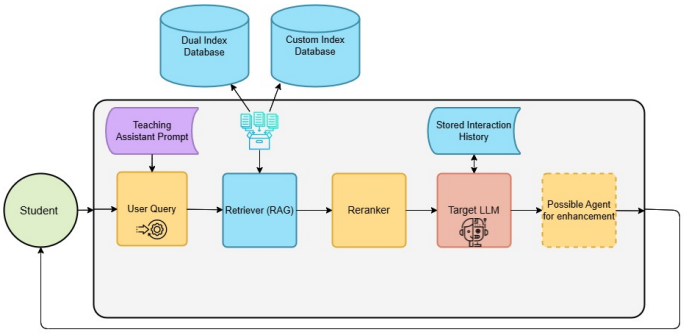
AI tools are an emerging field that is transforming education. In a time when students are increasingly relying on chatbots for support in technical subjects, Professors like to have their own specialized chatbots that can provide guidance within technical fields like Communication Systems, where detailed, domain-specific knowledge is essential. Existing solutions lack the specificity and interactivity required for higher-level technical education. The design aims to provide responses that guide students towards solutions, mirroring the role of a human teaching assistant.

The CommGPT has been proposed to fulfill this need as course specific personalized AI teaching assistant tailored for EECE 442: Communication Systems. Several constraints impact the project, including computational limitations due to reliance on limited resources and technicalities, data security, sufficient accuracy, clarity and coherence of responses, and educational responsibility.

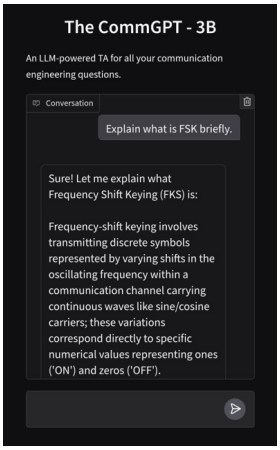
OBJECTIVES

- Deploy a course-specific tutor chatbot for EECE 442 that responds reliably and guides to correct answers.
- Engineer a modular, scalable pipeline whose tools, models, and data can be swapped or ported to any advanced course with minimal refit.
- Maintain an updateable vector knowledge base, a comm-systems dataset, and an MCQ benchmark for fine-tuning, retrieval, and evaluation respectively.

DESIGN AND IMPLEMENTATION



The proposed solution first uses retrieval-augmented generation (RAG) to combine two sets of documents: collected clean chunks, retrieved by comparing the query to specially generated summaries using cosine similarity, and another retrieved from a proprietary database. These documents are then reranked using a cross-encoder. Finally, they are passed to an LLM, fine-tuned successively on easy, medium, and hard question-answer pairs using our domain-specific data. This LLM is given a special prompt to take into consideration a conversation context window, as well as TA stylistic guidelines.



For data processing and creation, we used marker and Gemini.

In the RAG, we use the Langchain FAISS index and semantic chunker, as well as open-source embedding, cross-encoder, and generative models from Huggingface.

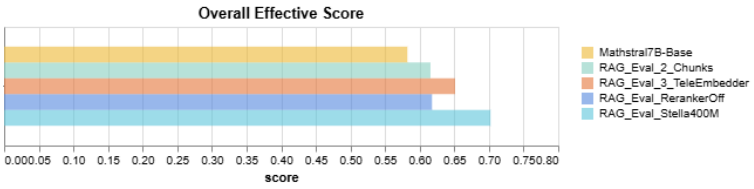
As for Fine tuning, it was done by the help of the Unsloth library on an A100 GPU hosted on Runpod.

Lastly, we developed our UI using Gradio, integrating conversation history, a custom system prompt, and multiple user accommodation.

RESULT ANALYSIS

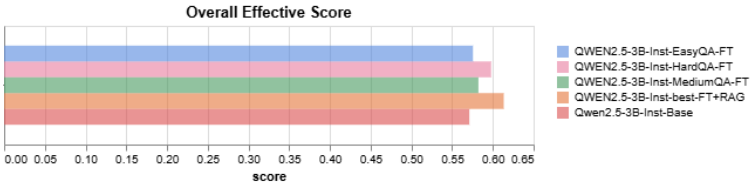
RAG

- Best RAG setup: Stella400M Embedder with a Re-ranker, k = 3 Chunks from both Databases combined.
- Outperformed the developed TeleEmbedder (communications specialized embedder) setup, the same setup with less or more chunks, as well as a Re-ranker-less setup.
- Achieved an overall score of 70.22% compared to a 58.22% with the baseline.



FINE TUNING

- Steady improvement achieved after each fine-tuning stage, from 57.11% to 57.5% to 58.2% to 59.7% going from base to easy to medium to hard respectively.
- Confirmed that the curriculum-style approach effectively imparts domain knowledge and reasoning skills to the base model.
- Best setup: RAG + Fine-tuned setup with Qwen 3B, achieving a 61.33% score compared to a 57.11% baseline.



TESTING STRATEGY

We developed a benchmark consisting of over 450 questions diversely created and gathered questions to gauge the efficacy of our various Fine tuning and RAG configurations.

- 294 MCQ extracted from select chunks of our data containing important concepts by the help Gemini, roughly equally divided over 9 topics of the course.
- 25 manually written Questions about EECE 442.
- 116 questions testing on general Communication lexicon and concepts as guardrails against overfitting.

CONCLUSION AND FUTURE CONSIDERATIONS

- CommGPT, though built for EECE 442, demonstrates a transferable pipeline that other advanced courses can replicate with minimal adaptation.
- Bigger models and richer, course-aligned corpora would further raise answer fidelity, depth, and student satisfaction.
- Introducing RLHF would align responses with pedagogical tone and ethical norms, yielding feedback that better matches diverse learning styles.
- Training on full tutor–student conversations could produce more natural, context-aware guidance than Q-and-A pairs alone.
- Agentic tools (e.g., Wolfram / SymPy) can sharpen multi-step derivations and equation handling.

AMERICAN UNIVERSITY OF BEIRUT
MAROUN SEMAAN FACULTY OF ENGINEERING & ARCHITECTURE

CREATIVE ACHIEVEMENTS

Bibliography

[1] V. Paruchuri, "Marker: A multimodal document parser," 2023. GitHub repository.

[2] G. F. de Castilho, R. Weijers, J.-F. Godbout, R. Rabbany, and K. Pelrine, "Quantifying learning-style adaptation in effectiveness of llm teaching," *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[3] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020.

[4] A. Maatouk, K. Ampudia, and R. Ying, "Tele-llms: A series of specialized large language models for telecommunications," *arXiv preprint arXiv:2409.05314*, 2024.

[5] A. Maatouk, F. Ayed, N. Piovesan, A. D. Domenico, M. Debbah, and Z.-Q. Luo, "Teleqna: A benchmark dataset to assess large language models telecommunications knowledge," 2023.

[6] X. Jiang, Y. Song, Y. Zhang, Y. Wang, and C. Ju, "Opticomm-gpt: A gpt-based versatile research assistant for optical fiber communication systems," *Optics Express*, 2024.

[7] H. Zou, Y. Tian, L. Bariah, and F. Bader, "Telecomgpt: A framework to build telecom-specific large language models," *arXiv preprint arXiv:2407.09424*, 2024.

[8] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversation," *arXiv.org*, October 3 2023.

[9] T. Zhang, S. G. Patil, N. Jain, S. Shen, M. Zaharia, I. Stoica, and J. E. Gonzalez, "Raft: Adapting language model to domain specific rag," Mar. 2024.

[10] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," *arXiv preprint arXiv:2401.08281*, 2024.

[11] D. Han, M. Han, and U. Team, "Unsloth," 2023. Open-source library for efficient LLM fine-tuning.

[12] R. Y. Aminabadi, S. Rajbhandari, M. Zhang, A. A. Awan, C. Li, D. Li, E. Zheng, J. Rasley, S. Smith, O. Ruwase, and Y. He, "Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale," *arXiv preprint arXiv:2207.00032*, 2022. Details DeepSpeed's inference optimizations.

[13] D. Zhang, J. Li, Z. Zeng, and F. Wang, "Jasper and stella: distillation of sota embedding models," 2025.

[14] M. Team, "Mathstral-7b v0.1," 2024.

[15] M. AI, "Llama 3.1-8b," 2024.

[16] Qwen Team, "Qwen2.5 technical report." `https://arxiv.org/abs/2412.15115`, Jan. 2025. `https://arxiv.org/abs/2412.15115`.

[17] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022.

[18] G. N. Group, "Olympic arena leaderboard," 2024.

[19] A. Maatouk, "Llama 3.8b tele," 2024.

[20] K. Wang, N. Thakur, N. Reimers, and I. Gurevych, "Gpl: Generative pseudo labeling for unsupervised domain adaptation of dense retrieval," *arXiv preprint arXiv:2112.07577*, 2022.

[21] ISO/IEC, "Systems and software engineering—systems and software quality requirements and evaluation (square)—system and software quality models," 2011. ISO/IEC 25010:2011.

[22] ISO/IEC, "Information security, cybersecurity and privacy protection—information security management systems—requirements," 2022. ISO/IEC 27001:2022.

[23] IEEE Standards Association, "Ieee standard for data privacy process," 2020. IEEE P7002.

[24] IEEE Standards Association, "Ieee standard for child and student data governance," 2020. IEEE P7004.

[25] B. S. Bloom, *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain*. New York: David McKay Co Inc., 1956.

[26] AI4K12 Initiative, "Ai for k-12 initiative guidelines," 2021. Accessed April 2025.

[27] X. Ma, W. Jiang, and H. Huang, "Problem-solving logic guided curriculum in-context learning for llms complex reasoning," *CoRR*, vol. abs/2502.15401, 2025.

[28] S. Li, H. Vaidya, O. Ruwase, and Y. He, "Pytorch distributed: Experiences on accelerating data parallel training," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3005–3018, 2020.

[29] Y. Zhou, Y. Cui, Q. Yu, X. Li, Z. Yan, K. Sun, *et al.*, "Qwen: Open-source large language models from alibaba cloud," 2023.

[30] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023.

[31] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.