Multimodal Data-Driven LLMs for Personalized Learning: Building a Virtual Assistant for Capstone Course

Master's Thesis submitted to the
Constructor Institute
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science, Software Engineering and Leadership

presented by

Yunus Tijani

under the supervision of

Prof. Manuel Oriol

Julia Kotovich

Second Reviewer:

Prof. Mauro Pezze

May 2025

Statutory Declaration

Family Name, Given/First Name	Tijani, Yunus Olamide
Matriculation number	30007210
What kind of thesis are you submitting: Bachelor-, Master- or PhD-Thesis	Master Thesis

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

This document was neither presented to any other examination board nor has it been published.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.

Yunus Tijani

Yunus Tijani
Schaffhausen,
19th May 2025

Acknowledgement

I am most thankful to God for giving me the strength, perseverance, and grace to finish this thesis. This journey has been one of learning and growth, and I could not have gone through it without the help of others.

To my supervisor, Prof. Manuel Oriol, and Julia Kotovich, thank you for your unwavering support, guidance, and trust in my work. Your advice and encouragement gave me the insight and motivation I needed to keep pushing.

A big thank you to my parents, Baba and Mama Yuyu for their endless love, prayers, and sacrifices. To my siblings, Semira, Aisha. Fatima, Mubarak, Maryam thank you for standing by my side and lifting my spirit when times were tough.

To my wonderful friends and colleagues, Chu, Fabian, Toye, Sayi, Dawud, Mubarak, thank you for the support, and the joy that you infused into this endeavor. You made the most stressful days bearable.

Finally, everyone who had the courage to believe in me, listened to me patiently, and reassured me of the light at the end of the tunnel, thanks.

Table of Contents

List of Tables

List of Equations

List of Figures

Abstract

In today's education institutions, lecturers face challenges in providing personalized support to students particularly in large classrooms. Recent advances in Large Language Models (LLMs) such as ChatGPT has shown its capabilities in providing educational assistance to students, however, these models are typically designed for general use and lack the ability to adapt to specific course content. As a result, students don´t get the tailored guidance needed to fully understand different course concepts and engage with their coursework.

This thesis addresses the gap by developing a Virtual Teaching Assistant (VTA) tailored to the content, and learning outcomes of a course, using the Capstone curriculum taught at Constructor Institute as a case study. The proposed solution involves fine-tuning LLMs on course-specific educational materials such as lecture transcripts and slides enabling the VTA to deliver context-aware support aligned with the curriculum. The system is built using Unsloth, a memory-efficient framework, and is deployed through OpenWeb UI to offer students an interactive learning interface.

This thesis produces a reproducible pipeline for converting course materials into intelligent educational agents, encouraging course teaching assistants with open-source technologies.

Chapter 1

Introduction

1.1 Background

Traditional educational systems often struggle with limitations in resources, particularly concerning student-teacher ratios. Large class sizes make it difficult for lecturers to provide individualized attention and accommodate each student's unique learning style and pace. Many students have a less successful learning experience as a result, and some lag. Even though tools like Learning Management Systems (LMS), have provided some relief with functions like lecture resource distribution and quiz taking, they have not yet addressed tailored learning support at scale for different curriculums that provide students with responsive, context-aware assistance.

The growing gap between student needs and faculty capacity has led to a search for intelligent systems that can bridge the gap between scalable teaching and personalized learning. Lecturers are expected to manage not only in-person teaching but also digital content delivery, assessment, and student support, often with little to no teaching assistance. In the case of large classes, frequently asked questions by students can eat up valuable class time, depriving teachers of other tasks or curriculum innovation.

The progress of artificial intelligence (AI) in the last year has been a major addition to the education sector, among others. Large language models (LLMs), one of the most promising AI developments, are represented by Claude (Anthropic), LLaMA (Meta), and GPT (OpenAI)[1][2][3]. These models have impressive capability for natural language understanding, question answering, and appropriate interaction generation. Their capacity for generalization has made it possible for AI-based learning instruments such as virtual assistants, information summarizers, and tutorial packages. The broadly trained general-purpose LLMs from large online sources often lack the depth and specialization required to serve students in a specific field of study. This reduces their usage as effective teaching aids at the course level for different academic institutions.

To address this problem, this thesis proposes the development of a Virtual Teaching Assistant (VTA) from fined-tuned LLMs trained on multimodal education course slide presentations and lecture video transcripts. The goal is to create a domain-specific assistant that can understand the capstone course concepts, generate semantic question-answer responses, and be deployed to OpenWeb UI[4].

1.2 The Role of Multimodal Educational Data

One of the theoretical foundations for this research is the concept of multimodal educational data. Unlike the term multimedia, which tends to refer to different forms of media (e.g., images, sound, video), multimodality in education refers to different modes of meaning-making and communication like linguistic, visual, spatial, gestural, and temporal representations[5].
In this project, the core data sources are:

- Lecture transcripts: which were generated from video recordings using automatic speech recognition (ASR).
- Course slides: which were extracted from PDF or PowerPoint files.

When the LLM uses text only, the concept of multimodality in the original source is still reflected in the text form, maintaining the structural and semantic flow of multimodal data in multimodal source content, something the LLM was specifically based on in text form, meaning the transcript captures the linguistic and temporal flow of lectures, mirroring how visual data contains structured, instructional cues. As such, the source remains multimodal, even when processed as a text form

Blikstein and Worsley claim that the potential of multimodal learning analytics lies in consolidating different forms of learning expression to represent deeper insights into pedagogy and student understanding[6]. Through using transcripts and slides together, this research embraces multimodality for building a fuller training dataset for fine-tuning the assistant.

1.3 Introducing Unsloth Technology

To carry out LLM fine-tuning on multimodal course materials, this research employs Unsloth, an open-source framework designed to optimize LLM training and inference[7].

Unsloth offers the following:

1. Lightweight Fine-tuning: It offers fine-tuning support for 1 billion (1B) and 3 billion (3B) model sizes optimized to run on limited hardware while providing adequate performance.
2. Conversational Instruction Tuning: It is designed for question and answering when generating conversations, a feature important for creating an interactive educational agent.

3. Academic Flexibility: Unsloth was developed to allow for the integration of domain-specific knowledge easily, accommodating teaching styles or fields of study.

Unlike out-of-the-box LLMs trained on general-purpose internet data, Unsloth allows for the development of context-specific VTAs, utilizing real course content to remain true to the instructor's voice and develop educationally coherent responses to student questions.

1.4 Research Motivation

This thesis is motivated by four main observations:

1. The scalability challenge is present in cases of a large classroom where there is likely not enough personalized feedback or personalization to clarify any misunderstandings. There are few teaching assistants, and instructors do not have an unlimited bandwidth[8].
2. While the characteristics of the LLMs such as GPT-4 are impressive, they ultimately are organized by generalizing over web-scale content. Sometimes they can be too general, and thus their responses cannot accurately reflect depth when responding to the question, especially if the context is course-specific[9].
3. The demand is rising for intelligent tutoring systems that mirror the course's voice, its teaching style, sequencing, and domain-specific terms, as well as general knowledge.
4. Virtual Teaching Assistants can help reduce these common duty workloads and free up time for instructors to focus on higher-order tasks like mentoring and designing curriculums.

This thesis examines how fine-tuning a pretrained LLM on multimodal course content will produce a teaching assistant that understands the course content and helps students with the course.

1.5 Research Aim and Scope

The objective of this work is to design and evaluate a Virtual Teaching Assistant that:

1. Takes in course material in the form of lecture transcripts and text from presentation slides.
2. Provides contextualized, relevant Q&A-style feedback to support student learning.
3. Is deployable as a conversational agent on OpenWeb UI.

The focus is intentionally constrained in the manner that it only deals with textual representations of multimodal data, rather than image or auditory input. The study also limits itself to one course with four years of archived material. This will allow a controlled study of content adaptation and response from the assistant.

1.6 Research Question

How effectively can a fine-tuned Large Language Model, trained in textual representations of multimodal course content, function as a virtual teaching assistant to support personalized learning in Constructor Institute capstone course?

1.7 Contributions

This thesis adds to the new intersection of educational technology, multimodal learning analytics, and LLM-based AI. It exemplifies how multimodal educational data can be enacted upon to produce structured text to be a viable training resource for domain-specific virtual assistants. It also provides a pipeline for converting university course materials into instructional data sets for LLM fine-tuning using Unsloth- which contributes educational alignment to generic AI.

Chapter 2

Literature Review

This literature review aims to produce a comprehensive insight into the research status of Large Language Models (LLMs) in education, through the consideration of existing works and their application in virtual teaching assistants (VTAs) and the performance of Unsloth framework.

2.1 Review of Existing Virtual Teaching Assistants (VTAs)

With the rise of artificial intelligence (AI) in the educational sector, Virtual Teaching Assistants (VTAs) like virtual teacher and student assistants have emerged. VTAs incorporates machine learning, natural language processing (NLP), and lately, large language models (LLMs) to give automatic, smart assistance in an educational environment. These are examples of VTAs present today:

2.1.1 Conversational VTAs

Conversation-style VTAs are employed most often. They perform mainly as chatbots that return responses to the student's query, explain instruction, and mimic one-on-one human tutoring. Khanmigo is an example designed by Khan Academy in conjunction with OpenAI[10]. Khanmigo uses GPT-4 for conversational support, facilitating student guidance, providing step-by-step help through completion of exercises, and providing one-on-one individualized feedback. Its integration into the Khan Academy platform shows the potential of large language models (LLMs) to revive traditional learning environments with human-like, context-aware conversational guidance.

2.1.2 Course-Specific VTAs

Jill Watson, a VTA created at Georgia Institute of Technology, is a quintessential example of what can be achieved through course-specific AI teaching assistants. Jill was trained using data from previous semester’s discussion forums within the Online Master of Science in Computer Science (OMSCS) program with IBM Watson's services[11]. It was implemented to respond to typical student queries raised in online forums. Jill Watson discovered that with structured, past course data, questions could be answered at high accuracy levels. While its domain specificity precludes generalizability, it shows the worth of training VTAs using highly controlled, high-quality datasets for specific academic courses.

2.1.3 Multimodal VTAs

Recent advances in multimodal AI have enabled the development of VTAs capable of processing and generating not just text, but also images, audio, and video. One of the leading models in this domain is Google's PaLM-E[12]. These models are trained to learn instructions from different types of data and produce outputs that correspond to the input type. For example, PaLM-E describes the contents of an image, and answers related questions.

In educational settings, multimodal VTAs are helpful in subjects where understanding is more than text-based and require interpretation of spatial, visual, or auditory information. For instance, in the field of healthcare, a multimodal VTA can help students make sense of X-rays or anatomical diagrams by placing explanations over the image. In architecture, it can help in the analysis of blueprints or 3D models by taking students through elements of the structure visually. Although they are not yet mainstreamed in traditional education, such VTAs show the next cutting edge in virtual assisting, where students can interact with tutors that understand and teach through multiple modalities.

2.1.4 Embedded VTAs in Learning Platforms

VTAs are now seamlessly integrated by a variety of educational technology solutions that have started to natively support them on their platforms. For instance, Duolingo Max uses GPT-4 to mimic natural conversation in the target language and provide real-time explanations upon a student making a mistake[13]. These kinds of tools show how learning environments are becoming more and more individualized, where the VTA dynamically adjusts content delivery.

2.2 Large Language Models (LLMs) in Education

2.2.1 Overview of LLMs

Large Language Models (LLMs) represent a landmark in the evolution of artificial intelligence that converges on decades of progress in linguistics, machine learning, and computational design. The models are trained to comprehend and generate human language enabling several applications in education, science, and healthcare.

LLMs began with machine translation and chat systems which led to IBM-Georgetown Russian-to-English translator and ELIZA projects that demonstrated computers could replicate aspects of human linguistic comprehension[14]. The early systems, though rule-based, drew attention to the potential of automatic linguistic understanding.

During the 1990s and early 2000s, scientists shifted from rule-based systems to statistical methods, employing techniques such as Hidden Markov Models (HMMs), n-grams, and Conditional Random Fields (CRFs) to model word order probability and parts of speech. Building on these foundations, the development of neural networks marked the next significant leap, with models like Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) cells, and Gated Recurrent Units (GRUs) offering improved handling of sequence data such as sentences and paragraphs. These advancements enabled more context-aware applications, including speech recognition and machine translation.

The Transformer architecture´s development allowed parallelized training and deeper context modeling using self-attention mechanisms. This led to the creation of LLMs pre-trained on massive datasets. GPT-3 and GPT-4, developed by OpenAI, show the capabilities of modern LLMs, offering few-shot and zero-shot learning and the ability to engage in meaningful multi-turn conversations[15][16]. Also, Google's BERT improved text classification, summarization, and translation by treating all language problems as a sequence-to-sequence challenge[17].

The open-source community has greatly contributed to enhancing the LLM ecosystem. LLMs such as GPT-Neo of EleutherAI, and LLaMA (Large Language Model Meta AI) have made high-performance accessible LLMs within reach for global academic researchers, educators, and developers[18][19]. It has been crucial in reducing dependence on proprietary black-box models as well as causing transparency, reproducibility, and collaborative innovation.

Open models are mostly used now because of parameter-efficient fine-tuning methods like LoRA (Low-Rank Adaptation) and QLoRA (Quantized LoRA)[14]. These allow researchers to finetune models on limited computational resources. Instead of finetuning all parameters, LoRA finetunes a low-rank, small subset of the model's weights, which speeds up training and lessens the requirement for memory, while QLoRA builds on this with quantized fine-tuning at 4-bit precision without loss of accuracy.

2.3 Fine-Tuning Large Language Models with Unsloth

2.3.1 Fine-Tuning LLMs

Fine-tuning is the process of making a pre-trained Large Language Model (LLM) better aligned with a specific domain or dataset by continuing its training on related, task-specific data. In the academic environment, this commonly involves using lecture transcripts, text from presentation slides, and explained interactions to teach the model how content is written and presented in each course. This enables the model to learn domain-specific terminology, ordering, academic intent, and instructional language. Unlike Retrieval-Augmented Generation (RAG), which relies on external documents during inference time, fine-tuning puts this domain knowledge in the model parameters directly, with answers being faster and more contextually appropriate.

One powerful tool that enables efficient fine-tuning today is Unsloth, an open-source framework that is aimed at improving training performance and memory efficiency for LLMs. Unsloth is optimized for models like LLaMA, and Mistral and offers up to 30x speedup of training times and 60% of memory savings compared to traditional methods. This improvement is achieved through a collection of engineering improvements, such as manual autograd, chain-matrix multiplication optimization, and in-place gradient calculation for LoRA adapters[36].

LoRA (Low-Rank Adaptation) is the central component of Unsloth's method. LoRA works by freezing weights at the core of the base model and only adjusting small rank-decomposed matrices. Unsloth further enhances this through manually computed matrix differentials that are efficiently combined. All of these allow one to fine-tune massive models even with low hardware, opening LLM training to everyone[36].

Unsloth further accommodates QLoRA (Quantized LoRA), a cutting-edge variant that quantizes the model weights to 4-bit. This greatly minimizes the memory requirements, and it is possible to fine-tune 7B or even 13B parameter models on consumer-grade GPUs with a mere 8–12GB VRAM. Such accessibility is invaluable for researchers and instructors who wish to develop domain-specific applications like Virtual Teaching Assistants (VTAs) without needing high-end cloud infrastructure[36].

Unsloth's strategy of fine-tuning is user oriented. It provides notebooks on Google Colab, Kaggle, and local environments to allow users to train models with only 3GB VRAM required. The notebooks simplify model configuration difficulties but expose options like max_seq_length, and quantization levels (load_in_4bit, load_in_8bit, and full_finetuning). Unsloth allows simple data formatting with training data like question-answer pairs, which is appropriately suited for educational use cases[36].

Unsloth´s hyperparameter tuning is designed for avoiding overfitting and underfitting. Key parameters such as learning rate, batch size, epochs, dropout rate, and sequence length can be tuned. Different approaches like gradient accumulation and causal masking are enabled by the framework to stabilize the training. Unsloth supports streamlined evaluations processes. Moreover, it also supports both manual and automatic model evaluation so that users can judge responses through conversational and performance metrics. Models can be saved for inference engine usage after being trained with Ollama and saved in the form of small LoRA adapter files. Post-training support ensures model portability and reusability in a range of deployment environments[36].

Unsloth also shines through the availability of long-context fine-tuning support, enabling full 342,000 token processing per pass—a huge advantage in learning settings where cumulative lecture notes and lengthy transcripts must be held across interactions.

2.3.2 Dataset Preparation in Unsloth

Effective fine-tuning in Unsloth depends a lot on dataset quality and structure. Datasets need to be well-formatted and contextually dense for compliance with tokenizer expectations and model structure. Unsloth supports a diverse range of formats including Alpaca-style instruction formats, ShareGPT multi-turn dialogues, and role-based ChatML messages that are easily parsed by built-in chat templates.

A dataset typically takes the shape of task instructions coupled with expected outputs, which may be in single-turn (instruction → response) or multi-turn (dialogue history) formats. For fine-tuning instructions, Alpaca format:

```
{
  "instruction": "Explain recursion.",
  "input": "Give an example.",
  "output": "Recursion is when a function calls itself. For example..."
}
```

For the development of conversation-based models, multi-turn dialogue datasets accurately reflect the dynamics of real-world educational interactions. Unlike single-turn instruction datasets (e.g., Alpaca format), multi-turn datasets simulate extended interaction that has greater similarity to how real educational dialogue is conducted within classrooms.

ShareGPT-formatted multi-turn datasets generally consist of student and assistant alternating exchanges (VTA). Such a format shows instructional flow, clarification questions, and learning, and is therefore appropriate for modeling pedagogic dialogue. Example of this structure is:

```
{
  "conversations": [
    {
      "role": "user",
      "content": "What is 'the project book p.1 responsibilities p.2 imposed technical choices p.3 personnel characteristics' about?"
    },
    {
      "role": "assistant",
      "content": "This section covers: Page 1 – the responsibilities associated with the project; Page 2 – the technical choices imposed on the team; Page 3 – the characteristics and constraints related to the personnel. Each requirement may be rated for quality on a scale of 0 to 2."
    }
  ]
},
```

To simulate real-life conversations, unsloth supports the conversation_extension parameter where setting this parameter, conversation_extension=3, three random rows from the dataset are picked and merged, allowing the model to learn turn by turn and continuously. This is useful when working with datasets originally designed for single-turn interactions.

Unsloth recommends at least 100 samples for initial fine-tuning, 1,000+ rows as good for stability and scaling. Multi-turn dialogue sets provide the basis for training quality instructional assistants. Unsloth's tooling and format pipeline streamlines the process of preparation to allow researchers to focus on content curation and teaching design without concerns for model compatibility or training efficiency.

2.4 Review of Existing Works

LittleMu, a virtual learning assistant created by Tu et al.[20], was developed to help students in Massive Open Online Courses (MOOCs) by offering conversational support and automated question response. The XuetangX platform was used to implement the system to overcome the scalability issue that human instructors faced in extensive online learning settings[21]. The proposed architecture combined diverse information retrieval and generative prompt to enhance interaction quality. For query retrieval, the process utilized a few sources like video subtitles' concept graph, FAQ documents, out-of-domain web search engines like AMiner band ranked by a concept-aware BM25 algorithm[22]. An ALBERT classifier separates factual questions from general chat. In the generation stage, a "Chain of Teach" prompting method is suggested by the authors, simulating step-by-step reasoning to teach large language models to answer multi-step academic questions[23]. In conversational chat, a knowledge-guided prompt from past context is used for coherence. The system was evaluated both with ROUGE scores and human evaluation on informativeness, coherence, and helpfulness metrics, on which LittleMu outperformed baseline systems. Online metrics post-deployment indicated high user engagement and satisfaction rates following the addition of the generation module. The present work demonstrates the usefulness of integrating LLMs with multi-source retrieval and prompt engineering in developing successful VTAs for large-scale online learning systems[20].

Kortemeyer[24] introduced "Ethel," a virtual teaching assistant (VTA) that was created at ETH Zurich to support new physics students with the addition of Retrieval Augmented Generation (RAG). Ethel analyzes a wide variety of course-level PDFs against OpenAI's LLMs with the dual utilization of LangChain and add embeddings, to ensure questions are answered accurately. More than 1,500 students were enrolled in six courses within the system, many of which incorporated multilingual capabilities through GPT-4, while open source LLMs like LLaMA were trained for English-only environments. Beyond simple question answering, Ethel is used to convert handwritten solutions to LaTeX via Mathpix and GPT-4V/GPT-4o, providing feedback on homework, and grades tests. The feedback feature was initially designed to be human-like but was later altered to be neutral, addressing anthropomorphization issues. The paper does, however, raised pedagogical concerns about the efficacy of RAG as a possible deterrer of deeper learning through excessively verbose answers. The grading function fares well under open-ended answer structures but poorly against numerical correctness and handwriting identification. Financially, Ethel is set at $7.50 per student per course per semester on Azure AI Services, with possible future cost savings using other models[25].

Goel et al. presented the Jill Watson series of Virtual Teaching Assistants (VTAs), which was designed to meet the scalability concerns of human teaching assistants for extensive online courses[11]. JW1, which first functioned under human supervision before eventually becoming unsupervised, used IBM Watson APIs and a memory-based retrieval mechanism trained on previous question-answer pairs. Later releases, JW2 and JW3, added semantic processing to improve the quality and flexibility of dialogue, particularly in application to handling student greetings and answering extensive course questions. The VTAs were deployed to Georgia Tech's Knowledge-Based Artificial Intelligence (KBAI) class via its OMSCS platform, where they supplemented human TAs with forum responses. Performance improved with each iteration, and JW3 model achieved higher in answer accuracy. In comparison to JW1, JW2 and JW3 were built with open-source NLP libraries, which provided modularity and scalability. Response rates and accuracy were used to check its progress, and student´s feedback indicated a very high usage rate. One new aspect of the deployment was hiding the artificial nature of the VTAs initially, to allow for an unbiased analysis of the interactions. The study finds that there is a phase-based approach to development and validates the viability of AI-enhanced teaching assistants to increase participation in MOOCs. Even though the system used text-based representation, its semantic processing and adaptive refinement mechanism provides significance as a reference model for personalized virtual education systems.

A more effective Jill Watson, virtual teaching assistant (VTA) for online classes was introduced by Taneja et al.[26]. This model is not finetuned or trained on training data and targets replication with just ChatGPT and publicly available content. XiaoIce inspired skill-based architecture is used to create the VTA, and student questions are sent to the appropriate skills according to intent classification. One of the basic skills is context-specific response, in which relevant extracts from educational resources, such as courses and slides, are extracted using dense passage retrieval (DPR). These are then passed on to ChatGPT, which generates a response citing exact page numbers and positions. The system performs a textual entailment check to ensure that the response is consistent with the entailed context it extracted, improving the factual coherence. Safety is guaranteed by employing OpenAI's Moderation API and carefully designed prompts to minimize toxic or inappropriate output. The authors compared Jill Watson with a legacy system (LJW) and OpenAI's Assistants (OAI-Assist) and reported better performance in accuracy, relevance, and safety. This research presents a practical yet effective road map for the implementation of secure and scalable VTAs in higher education, with broad implications for more comprehensive LLM-based learning tools.

Sakib et al. [27]presented the development and deployment of a virtual teaching assistant (VTA) bot built using deep learning and natural language processing (NLP) to teach Python programming to undergraduates. Sakib et al. [27]prepared a question-answer dataset from learning websites and formatted it with intent, topic, tag, and question-answer pairs since no appropriate dataset was found. To pre-process the data before training the models, it was passed through an extensive array of preprocessing techniques like stemming, lemmatization, and stop word removal. After training and testing four baseline classifiers (Naive Bayes, Decision Tree, Linear SVM, and Logistic Regression), it was observed that removal of less descriptive tags improved the accuracy and F1-scores. A more complex feed-forward neural network with two hidden layers implemented in PyTorch, with a bag-of-words feature representation, to achieve over 95% accuracy on the cleaned-up dataset. The neural network was deployed using a chatbot interface developed in Flask and JavaScript to simulate actual student interaction. The paper shows the end-to-end pipeline of constructing a domain-specific educational chatbot, from dataset construction to deployment.

Davalos et al. [28]explored the possibility of using Large Language Models (LLMs) to process multimodal student data and generate readable reading assessment reports for teachers. This study used eye-tracking data, learning objectives, and test questions of fifth-grade students using RedForest. K-Means was used instead of GMM and Spectral Clustering in unsupervised clustering, which was used to detect trends in reading activity. These patterns are subsequently processed in an LLM to generate student profiles highlighting strengths, weaknesses, and instructional suggestions. An additional LLM was used for evaluating these reports on interpretability, relevance, and instructional purposes. Teachers were questioned about the interpretability and usability of the reports drawn for the validity of the system. The findings showed that the teachers highly regarded the cluster-based profiles and feedback related to behaviors, confirming the worth of reports drawn by the LLM. However, instructors pointed out gaps like imprecise expressions, lack of proper structure, and more powerful directions being needed. The authors emphasized that a scalable solution for personalized instruction includes the blend of teacher feedback with machine derived insights. The solution has a lot of potential for educational virtual teaching assistants to reason over multimodal data and refines assistance according to student action.

Personalized Multimodal Generation (PMG) proposed by Shen et al.[29], is a system that uses large language models (LLMs) to generate personalized multimodal content. Personalized outputs like movie posters, emoticons, and product images are generated using multimodal conditioning with user preference extractions. User preferences are represented by both explicit keywords and implicit embeddings after being extracted from behavior data, such as clicks and conversations. These target item keywords and preferences provided as input to a multimodal generator such as a diffusion model or an LLM-based model to regulate content generation. A novel weighting mechanism is employed to balance personalization against accuracy by a weighted sum of preference and relevance scores. Baseline comparisons demonstrated that higher picture relevance and personal alignment employ both automatic measurements (Learned Perceptual Image Patch Similarity (LPIPS), Structural Similarity Index Measure (SSIM), and human judgment. Ablation study indicates that words and embedding have better performance collectively than individually. PMG was also tested with in a downstream recommendation task, revealing that generated images improve model accuracy over non-personalized counterparts. Although the method currently handles visual modalities, the framework is sufficiently general to accommodate personalization over multimodal domains. The paper has significant implications for LLM-based personalization and specific implications for the creation of virtual assistants with flexibility for user-specific learning styles.

Bratić et al. [30]proposed a hybrid architecture for centralized access to learning content based on transformer-based natural language processing combined with an LLM/chatbot API. The model solves the issue of isolated databases appearing in education by creating a platform that integrates access to various resources such as scripts, presentations, and articles. Embedded in the system is a highly advanced structure built from a transformer-based model, state-of-the-art word embedding, masked multi-self-attention mechanisms, and module-based layered units like front-end UI, back-end logic, knowledge-based modules, and e-learning interfaces. The hybrid chatbot operates as a blend of rule-based and AI-computed responses for both pre-defined fallback behavior as well as adaptive learning of user input. The implementation supports multi-language processing and PDF document formats, but it currently excludes image, video, and extremely large datasets. Although not fine-tuned, the system uses pre-trained LLMs like OpenAI's GPT models to respond to user inputs with domain-relevant information and to avoid manual scripting and to dynamically interpret context and semantics from user inputs. Bratić et al. [30]emphasized the use of semantic text segmentation and user-specific query processing to ensure high precision and adaptability. Moreover, a comparison of this model with existing rule-based bots highlights its capacity to deliver interactive, scalable, and personalized responses through integrated educational material databases. Using

AI-based reasoning modules, the deployment aims to optimize information retrieval and maintain query accuracy to provide scalable smart educational assistants in e-learning environments.

According to Zheng et al.[31], the functionality of large language models (LLMs) for open-ended conversational tasks remains difficult to assess using conventional benchmarks. To address this issue, the authors propose using robust LLMs, such as GPT-4, as automatic judges or "LLM as a Judge" to evaluate chatbot output. Their strategy focuses on two main components: Chatbot Arena, a crowdsourced platform where users anonymously evaluate chatbot responses, and MT-Bench, a manually selected collection of multi-turn dialogue prompts. The research highlights the main biases of LLM-based adjudication, including position prejudice, verbosity bias, and self-enhancement bias. To solve these biases, strategies including reference-guided grading and response swapping were suggested. GPT-4, as a judge, shares an over 80% agreement rate with human raters, which is on par with the agreement among human evaluators themselves. The authors further demonstrate that GPT-4 boasts solid judgment quality under duress like repetitive responses or mathematical questions, especially when augmented by chain-of-thought or reference responses. It was realized that GPT-4 model is more in line with human preferences when compared to other LLMs like GPT-3, Claude, and LLaMA variations[2][19]. This scalable automated evaluation technique allows quicker chatbot development cycles, and the results validated LLM-as-a-Judge as a viable and trustworthy technique for future LLM testing of conversational AI models.

To help current and potential university students with inquiries about courses, admissions, and campus services, Odede et al. [32]deployed JayBot, an LLM-based chatbot. It was developed on top of OpenAI's GPT-3.5 Turbo and incorporates an embedding model with a vector database (Pinecone) to store and retrieve course-specific data[33]. Prompt engineering was used to guide model behavior, leading to precise and contextually appropriate responses that often include confirmed contact information and links. System architecture includes a Python backend (with Flask and LangChain) and a JavaScript/TypeScript frontend to facilitate a responsive and user-friendly interface. In actual trials at Wolverhampton University, JayBot outperformed traditional live chat solutions for speed, accuracy, and customer satisfaction, with 96% of users having a positive experience. Its ability to maintain natural dialogue flow with the provision of sound, domain-specific answers make it an extensible support system during times of maximum admissions. The retrieval-augmented generation strategy of the chatbot ensures that even with static base LLM, users may be served with real-time information.

Planned future features include real-time integration with university portals for dynamic admissions tracking and provision of multimedia-rich answers.

2.5 Challenges of Generic LLMs in Academic Settings

Large Language Models have performed impressively across various language tasks. However, their immediate deployment in learning spaces has some inadequacies. These challenges have impacted on the model’s reliability and educational accuracy when applied without fine-tuning or domain-specific adaptation.

One of the most significant issues is the phenomenon of hallucination. General LLMs are trained in vast internet-scale data and give probability-based responses rather than verified facts. As a result, they tend to give fluent but factually incorrect statements. In an educational environment, where accuracy and precision are most important, such hallucinations can potentially confuse students or undermine confidence in learning tools[34].

Another limitation is the lack of course context. Generic LLMs do not learn on institution-specific texts such as slides, notes, or curricula. This means that they often fail to detect instructional order, domain-specific words, or the scope of what has been covered in a certain course. Thus, their response can be overly general or misplaced with the actual course content.[30].

Bias and ethical risks contribute to the challenges of applying generic LLMs in educational settings. The models contain societal biases particularly when applied to diverse student populations. Unless well-aligned and filtered, their application would result in ethical problems or even reputational harm to educational institutions[31].

Lastly, integration with institutional infrastructure is a problem. General-purpose LLMs are not natively integrated with learning management systems (LMS), grade books, or analytics software. They are unaware of class schedules, submission deadlines, or personal student performance histories. To function as effective educational assistants, LLMs must be better integrated with existing academic ecosystems.

Chapter 3

Methodology

This section discusses the system architecture used to develop a virtual teaching assistant (VTA) using multimodal data that was collected in capstone courses. The system utilizes data collection, processing, instruction generation, language model refinement, and deployment to perform real-time interaction.

3.1.1 System Architecture

The system follows a pipeline architecture consisting of five major stages: data acquisition, transcription and text extraction, instruction generation, model fine-tuning, deployment and evaluation. Each stage interacts with the next in a unidirectional manner. The pipeline is designed to handle large volumes of data like video lectures, PDF/PPT slides, convert them into dialogue formats, and generate structured conversational datasets for training large language models (LLMs).
This architecture was selected because the system is based on open-source technologies and is executed both locally and in the cloud, enabling reproducibility.

3.1.2 Architecture Diagram

The architecture below shows a high-level view of the entire system where each component within these layers is represented as a discrete block, with arrows indicating the flow of data showing the processing steps and grouping into four layers:

1. Data Gathering Layer
2. Data Processing Layer
3. Model Training Layer
4. Deployment & Interaction Layer

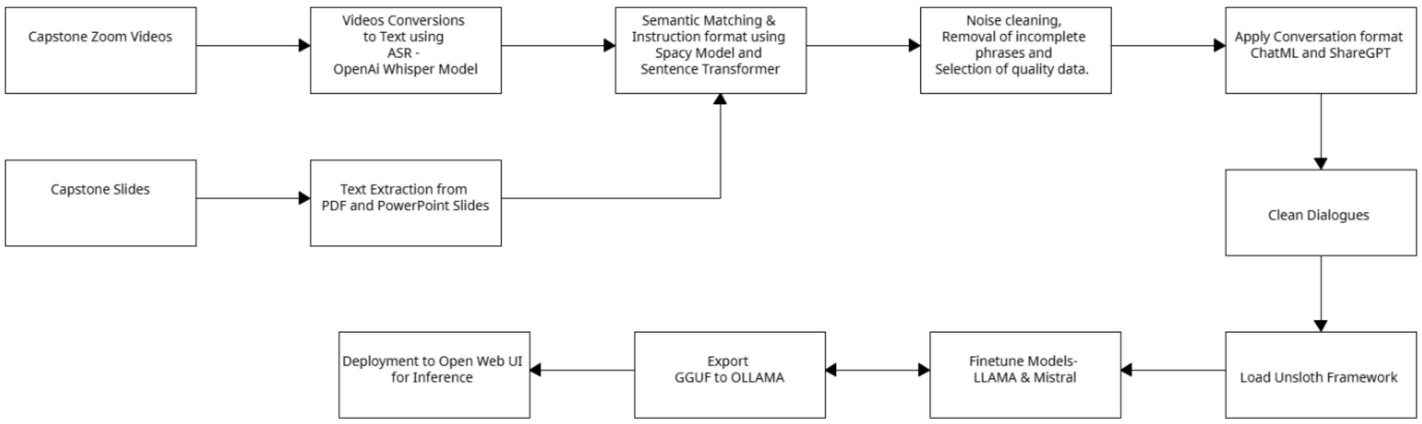

Capstone Zoom Videos
Videos Conversions to Text using ASR - OpenAi Whisper Model
Semantic Matching & Instruction format using Spacy Model and Sentence Transformer
Noise cleaning, Removal of incomplete phrases and Selection of quality data.
Apply Conversation format ChatML and ShareGPT
Capstone Slides
Text Extraction from PDF and PowerPoint Slides
Clean Dialogues
Load Unsloth Framework
Finetune Models- LLAMA & Mistral
Export GGUF to OLLAMA
Deployment to Open Web UI for Inference

Figure 1: High-Level Architecture of the VTA Training and Deployment Pipeline

3.1.3 Component Descriptions

The following is a detailed explanation of the system components shown in the architecture:

1. Data Gathering Layer: This layer is responsible for collecting raw educational data. Course slides were gathered while lecture recordings were downloaded using the Zoom Cloud Recording API with Server-to-Server OAuth application created on Zoom Marketplace[35].

2. Data Processing Layer: After data was acquired, OpenAI's Whisper model was used to transcribe lecture videos, converting speech into text. Audio was chunked into 30-second segments to optimize model performance. The PDF and PPTX files were processed to extract content in text form, which were then corrected for spelling errors, segmented into sentences, and chunked into smaller sections. Keywords were extracted from each transcript and semantic similarity matching was then performed using Sentence Transformers to pair keywords with relevant text spans.

3. Model Training Layer: This layer involves the fine-tuning of instruction-following language models. Two models are used were LLaMA-3.2-3B-Instruct and Mistral-7B-Instruct v0.3. Fine-tuning was performed using the Unsloth library, which allows memory-efficient training in Google Colab. The JSONL instruction datasets are tokenized using ChatML and ShareGPT formats respectively and fine-tuned using SFTTrainer with custom training arguments (batch size, learning rate, warm-up steps, evaluation strategy, etc.)[36].

4. Deployment & Interaction Layer: After training, the LoRA adapters are merged with the base models and exported in GGUF format, which is tested using llama.cpp. Deployment is carried out locally using Ollama, a lightweight CLI tool that loads GGUF models for inference and Open WebUI, a user-friendly web interface connected to Ollama that allows students and faculty members to test the model via a chat interface on capstone course materials.

3.2 Data Acquisition from Zoom

The Capstone course takes place online via Zoom once or twice a week. To download all capstone videos necessary for this thesis, I used Zoom’s Cloud Recording API to retrieve class session recordings stored in the Zoom cloud. I created a Server-to-Server OAuth application via the Zoom Developer Marketplace, using an admin account with cloud recording privileges. This type of authentication allows secure, Application Programmable Interface (API) access to retrieve metadata and downloadable links for stored recordings. This application generated the app ID, client ID, client secret, secret token and verification token which was used later. Through this application, API endpoints on the scope section were generated such as *cloud_recording:read:recording:admin, cloud_recording:read:list_user_recordings:admin, cloud_recording:read:list_account_recordings:admin*, to view recordings, list all recordings for a user to be queried within a specified time frame[37].

I automated the download of all capstone course related Zoom recordings using a Python script was employed based on Ricardo Rodrigues' GitHub open-source project, zoom-recording-downloader. The script used the Zoom API with OAuth Server-to-Server authentication to access and download all available cloud recordings. The script has a configuration file that specified the account credentials, download paths, and filtering options such as start/end dates and time zones. The script retrieves a bearer token, and queries Zoom’s API to list and download all recordings associated with the zoom email address associated with the class recordings[38]. Each recording is saved to the download path on local storage with filenames in this sample format “2024.07.09-04.57-0700-MSC Classroom 2024-Shared Screen with Speaker View (Cc)-a376e3be-a4c0-4f62-a77a-75b2d68b9090” in folders arranged based on date of recording and completed downloads are logged with unique codes to keep track if case of interruption.

The script's design integrates authentication, API interaction, file handling, and error logging effortlessly. This allowed a scalable method of collecting lecture recordings as a first step for dataset preparation. Capstone course recordings from November 2020 till May 2023 academic year were downloaded summing up to 165 videos as a source of dataset. I gathered all lecture slides used in the course from 2020 till 2024 academic year, which summed to 350 slides in PDF and PowerPoint formats.

3.2.1 Transcription of Lecture Recordings Using Automatic Speech Recognition (ASR)

To convert recorded lecture videos into text, they were transcribed with Automatic Speech Recognition (ASR) models using the following steps:

1. **Audio extraction and chunking:** Each video is split into smaller audio segments to manage memory efficiency as the model has limitations on the maximum input length they can process in a single pass. Chunking allowed the script to operate within those limits, particularly on systems with constrained GPU or RAM resources. The OpenAI Whisper model is designed to handle short, fixed-length audio inputs; for instance, Whisper processes 30-second chunks by default[39]. Chunking helped reduce transcription errors by avoiding issues such as input truncation or degradation of output quality over longer sequences. The script used the pydub library to extract and split the audio stream from each video and resample it to 16 kHz as required by the model.

2. **Model selection and transcription:** The OpenAI's Whisper model was selected due to its strong performance in real-world, noisy audio environments, its ability to handle long-form speech, and its robust pretraining on a diverse set of multilingual and multitask audio-text data[39]. Whisper is particularly effective for transcribing academic lecture recordings, which frequently feature multiple speakers, varied accents, and inconsistent audio quality.

   Once the video files were segmented into 30-second audio chunks, each segment was passed through the Whisper model for transcription. The model, which operates using transformer-based encoder-decoder architecture, performs end-to-end speech recognition and returns tokenized text. This text is then decoded into a readable transcript using the model's processor[40].

   The link to the full script is attached to Miscellaneous 2.

3.3 Text Extraction from PDFs and PowerPoint Slides

I used a python script to extract texts from all power point and PDF files used to teach in the capstone course. The script loops files with the.pdf,.pptx, or.ppt extensions and then applies the appropriate logic scripted. The Python-pptx library was utilized to read and extract text from the PowerPoint slides[41]. The script processes a slide at a time, iterates through all the shapes in a slide to locate those containing text. Text from each shape is compiled together to

reassemble the content of the slide. The process guarantees correct capture of all text components, including text within text boxes and placeholders.
It utilized the PyMuPDF library to read PDF documents[42]. Each page of a PDF document is opened successively, and the get_text("text") function is invoked to obtain plain text content. It is an efficient method to obtain textual data along with maintaining the document flow. All extracted texts are merged into one output text file.
The full script is attached to Appendix A.

3.3.1 Text Chunking

To make the processing of texts extracted from previous stage faster, I divided the text files into smaller chunks with a python script. The input path was specified, and the script read the file line by line, ignoring any blank lines. The script divided the filtered lines into segments, each containing 10,000 lines per chunk through a loop that divides the list of lines into consecutive blocks. Each chunk was then written to a separate text file and stored in an output directory.
The full script is attached to Appendix B

3.4 Instruction Generation and Semantic Matching

I used a natural language processing (NLP) pipeline to convert the extracted text from the video transcript and slides into structured question-answer pairs. The script starts processing each transcript using spacy's en_core_web_sm model, and PyTextRank extension which extracts keyword phrases based on their relevance within the sentence. For handling long transcripts, I set the maximum input for the model to handle up to 2.5 million characters. With each keyword extracted, the pipeline marks semantically suitable passages within the transcript. This is achieved by utilizing the all-MiniLM-L6-v2 model in the Sentence Transformers library, which represents keywords and sentences as 384-dimensional vector embeddings. Cosine similarity measurements are applied in matching keywords with the best-fit sentences. This ensures that the answers generated are informative and relevant.

Each keyword is translated into an instructional question, such as:
"What is 'keyword' about?"

The matching answer is within the sentence where the keyword is processed. To maintain quality, only answers comprising at least 10 words are considered. This filtering ensures that the generated pairs are substantial and meaningful. When no suitable keyword-answer pairs are available, the pipeline uses the fallback summarization strategy. It takes the first coherent paragraph of the transcript and carries out general summarization. It keeps doing this until the desired number of keyword-answer pairs is obtained. This ensures that every sentence contributes at least one informative response, and there is consistency within the whole dataset. The produced instruction-answer pairs are serialized to JSON Lines (jsonl) for future analyses. The output of each transcript is kept separately, while another aggregated file unites all generated pairs to make it readily accessible. A sample of the produced output is shown below:
{"instruction": "What is 'big upfront requirements' about?",
"input": "",
"output": "It is recognized as one of the most difficult tasks in software development and it led to the agile revolution of the 2000-2019s, when gathering big upfront requirements was deemed impossible in the general case.”

The full script is attached to Appendix C.

3.5 Conversational Structuring and Dataset Merging

I wrote a python code to combine jsonl files in the different chunks into a single json file in a order to simplify the summation of all instruction data generated above and structure the dataset into a conversational format.
Multiple jsonl files, each containing individual instruction entries were combined into a single consolidated json file. This was achieved by iterating through all jsonl files in a directory, reading each line as a distinct json object, and appending it to a collective list.
Subsequently, the combined dataset was then rephrased in the form of a dialogue to adapt to dialogue-based AI for conversational training and testing. Each instruction-output pair was redesigned as a conversation between an assistant reply and a user message. Specifically, the "instruction" field was converted into the user's message, and the corresponding "output" field the assistant's reply. A sample of the produced output is shown below:

```
{
    "conversations": [
      {
        "role": "user",
        "content": "What is 'team dynamics' about?"
      },
      {
        "role": "assistant",
        "content": "Team dynamics refer to the way teams develop and interact over time. Concepts include the development sequence in small groups and the impact of virtual collaboration, which can lead to more balanced participation but reduces non-verbal communication."
      }
    ]
  },
```

The full script is attached to Appendix D.

3.6 Training Environment Specification

The models were fine-tuned in a high-performance computing environment with the following configuration:

- **Python version**: 3.10
- **CUDA version**: 12.1.1
- **PyTorch version**: 2.1.1
- **Triton version**: 2.10
- **CPU**: 14-core processor
- **RAM**: 147 GB
- **GPU**: NVIDIA A100 (80 GB VRAM)

This configuration allowed for efficient training using 4-bit quantization and LoRA (Low-Rank Adaptation) without encountering memory bottlenecks, even for larger sequence lengths (2048 tokens).

3.7 Model Selection and Fine-tuning using Unsloth

To implement this thesis, two fine-tuned two large language models were utilized: Meta's LLaMA and Mistral. Both models were selected due to their strong performance on open-ended, instruction-tuned tasks and availability in quantized formats, which allow for memory-efficient training and inference. The fine-tuning process was conducted using the Unsloth framework, a lightweight optimization library for efficient LLM adaptation, particularly suited for resource-constrained environments.

3.7.1 Model Selection

Two pretrained models were explored:

1. **LLaMA-3.2-3B-Instruct** (via unsloth/Llama-3.2-3B-Instruct)
2. **Mistral-7B-Instruct v0.3** (via unsloth/mistral-7b-instruct-v0.3-bnb-4bit)

LLaMA, known as Large Language Model Meta AI, is a transformer-based language model released by Meta. LLaMA-3.2-3B-Instruct is accessed via the unsloth/Llama-3.2-3B-Instruct repository. The LLaMA-3.2-3B-Instruct variant is an instruction-tuned version, containing approximately three billion parameters, making it suitable for fine-tuning and inference on mid-range hardware while still providing robust performance in natural language understanding and generation tasks[43]. In addition, Mistral is a highly optimized, open-weight transformer-based language model developed by the Mistral AI team. Mistral-7B-Instruct v0.3, accessed through the Unsloth implementation at unsloth/mistral-7b-instruct-v0.3-bnb-4bit has gained recognition for delivering state-of-the-art performance at a small parameter scale, and it is designed to be efficient during both training and inference. The 7B parameter count refers to the model's size approximately seven billion trainable weights[44].

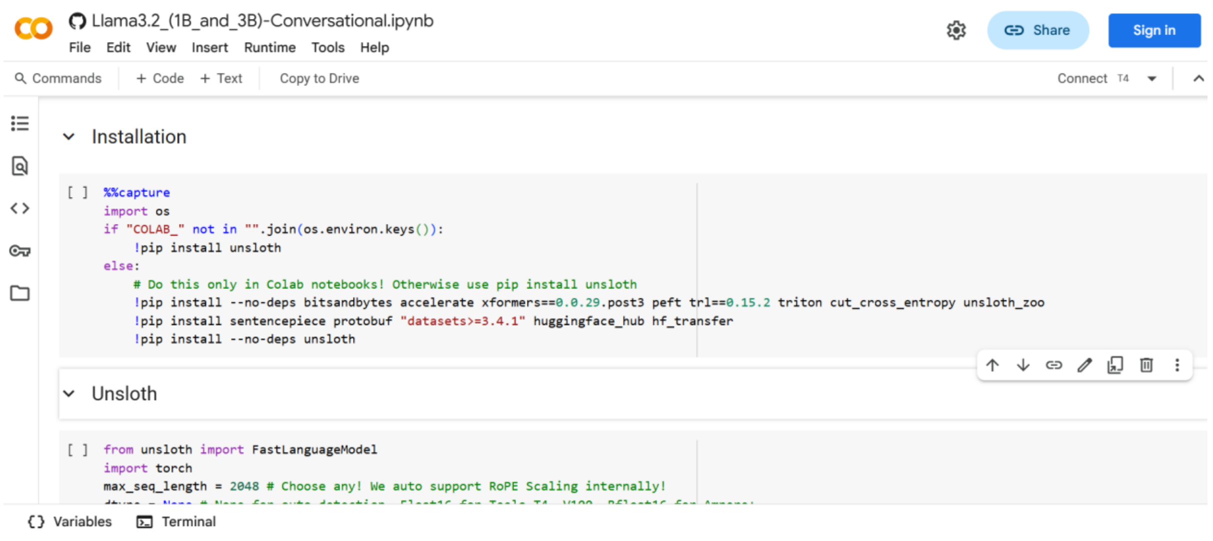
The models were pre-trained on instruction datasets and a general language dataset which contained carefully chosen examples where precise answers were provided with question instructions, enabling the model to be generalized across a variety of user questions and tasks. The instruct fine-tuning stage makes the model particularly well-suited for dialogue agents, and other assistant-style applications where the model must interpret natural language commands and respond.

LLaMA-3.2-3B-Instruct and Mistral-7B-Instruct v0.3 were chosen because of a few considerations. Firstly, their model size is a balance between capacity and computational resources. It is sufficiently small to be finetuned with limited resources such as NVIDIA A100 with 80 GB VRAM but large enough to produce well-formed and contextually relevant output.

Secondly, the models are available in a 4-bit quantized version via the Unsloth library, which dramatically reduces memory usage without significantly degrading performance. This makes it ideal for iterative experimentation and deployment in resource-constrained environments.

3.7.2 Fine-Tuning with LLaMA and Mistral Models

1. **Environment Setup:** Google Colab was used to finetune and train the model, this allowed required dependencies to be installed easily to support fast processing and memory handling.

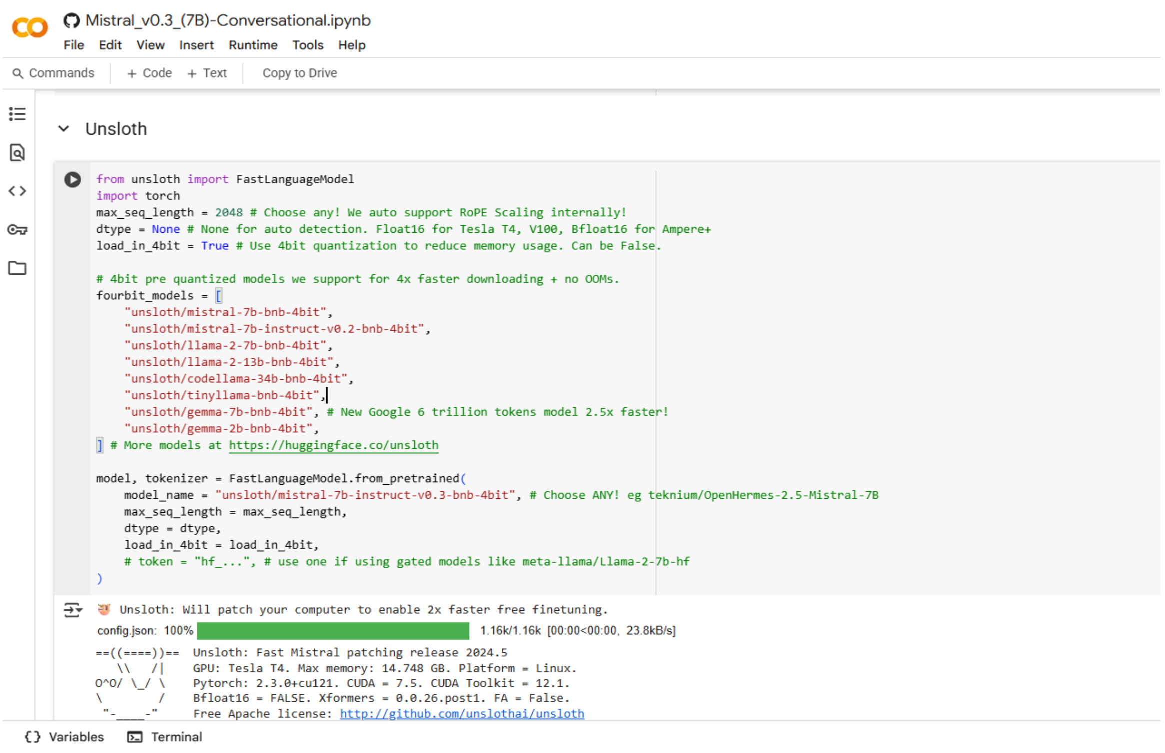


```
%%capture
import os
if "COLAB_" not in "".join(os.environ.keys()):
    !pip install unsloth
else:
    # Do this only in Colab notebooks! Otherwise use pip install unsloth
    !pip install --no-deps bitsandbytes accelerate xformers==0.0.29.post3 peft trl==0.15.2 triton cut_cross_entropy unsloth_zoo
    !pip install sentencepiece protobuf "datasets>=3.4.1" huggingface_hub hf_transfer
    !pip install --no-deps unsloth
```

```
from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
```

Figure 2: Google Colab Notebook used for Finetuning

2. **Model Loading and Configuration**: The LLaMA and Mistral model were loaded using FastLanguageModel with 4-bit quantization, and the datasets were loaded to the trainer with the train_dataset and eval_dataset parameters. LoRA adapters were inserted into key projection layers using get_peft_model () with a rank of 32 and alpha of 64. Both models were set to a sequence length of 2048 to accept long sentences and LoRA parameters were set to r=16, alpha=16, and dropout=0. In both training, only a subset of parameters is updated, reducing training time and compute requirements.


Mistral_v0.3_(7B)-Conversational.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Copy to Drive
Unsloth
Variables Terminal

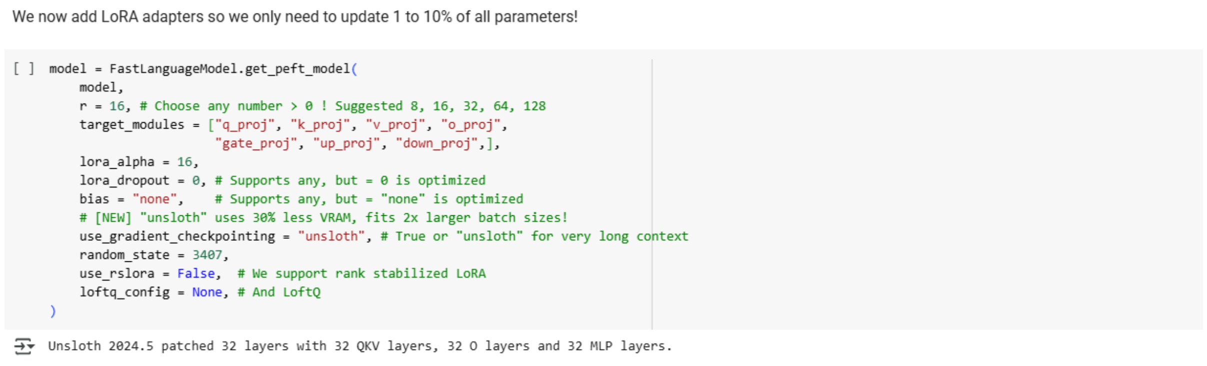
```
from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.

# 4bit pre quantized models we support for 4x faster downloading + no OOMs.
fourbit_models = [
    "unsloth/mistral-7b-bnb-4bit",
    "unsloth/mistral-7b-instruct-v0.2-bnb-4bit",
    "unsloth/llama-2-7b-bnb-4bit",
    "unsloth/llama-2-13b-bnb-4bit",
    "unsloth/codellama-34b-bnb-4bit",
    "unsloth/tinyllama-bnb-4bit",
    "unsloth/gemma-7b-bnb-4bit", # New Google 6 trillion tokens model 2.5x faster!
    "unsloth/gemma-2b-bnb-4bit",
] # More models at https://huggingface.co/unsloth

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/mistral-7b-instruct-v0.3-bnb-4bit", # Choose ANY! eg teknium/OpenHermes-2.5-Mistral-7B
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf_...", # use one if using gated models like meta-llama/Llama-2-7b-hf
)
```

```
Unsloth: Will patch your computer to enable 2x faster free finetuning.
config.json: 100%  1.16k/1.16k [00:00<00:00, 23.8kB/s]
==((====))==  Unsloth: Fast Mistral patching release 2024.5
   \\   /|    GPU: Tesla T4. Max memory: 14.748 GB. Platform = Linux.
O^O/ \_/ \    Pytorch: 2.3.0+cu121. CUDA = 7.5. CUDA Toolkit = 12.1.
\        /    Bfloat16 = FALSE. Xformers = 0.0.26.post1. FA = False.
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
```

Figure 3: Model Loading on Unsloth



We now add LoRA adapters so we only need to update 1 to 10% of all parameters!

```
[ ] model = FastLanguageModel.get_peft_model(
        model,
        r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
        target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                          "gate_proj", "up_proj", "down_proj",],
        lora_alpha = 16,
        lora_dropout = 0, # Supports any, but = 0 is optimized
        bias = "none",    # Supports any, but = "none" is optimized
        # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!
        use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long context
        random_state = 3407,
        use_rslora = False,  # We support rank stabilized LoRA
        loftq_config = None, # And LoftQ
    )
```

```
Unsloth 2024.5 patched 32 layers with 32 QKV layers, 32 O layers and 32 MLP layers.
```

Figure 4: LoRA Adapters on Unsloth

3. **Data Preparation and Formatting**: The dataset used was a JSON file containing multi-turn conversations, formed using a tokenizer template. For LLaMA model, conversations were standardized using standardize_sharegpt(). For Mistral model, the ChatML format was used, with <|im_start|> and <|im_end|> tokens to mark dialogue turns. The tokenization process used Unsloth's get_chat_template function as the chosen template and a role-to-key mapping of {"user": "human", "assistant": "trained model"} to match the dataset format. The response in the assistant section. The Dataset preprocessing was parallelized using two CPU workers (dataset_num_proc = 2) to speed up tokenization and template rendering. Sequence packing

was disabled (packing = False) to ensure that long conversations were processed independently, which led to slightly slower training performance for shorter inputs.

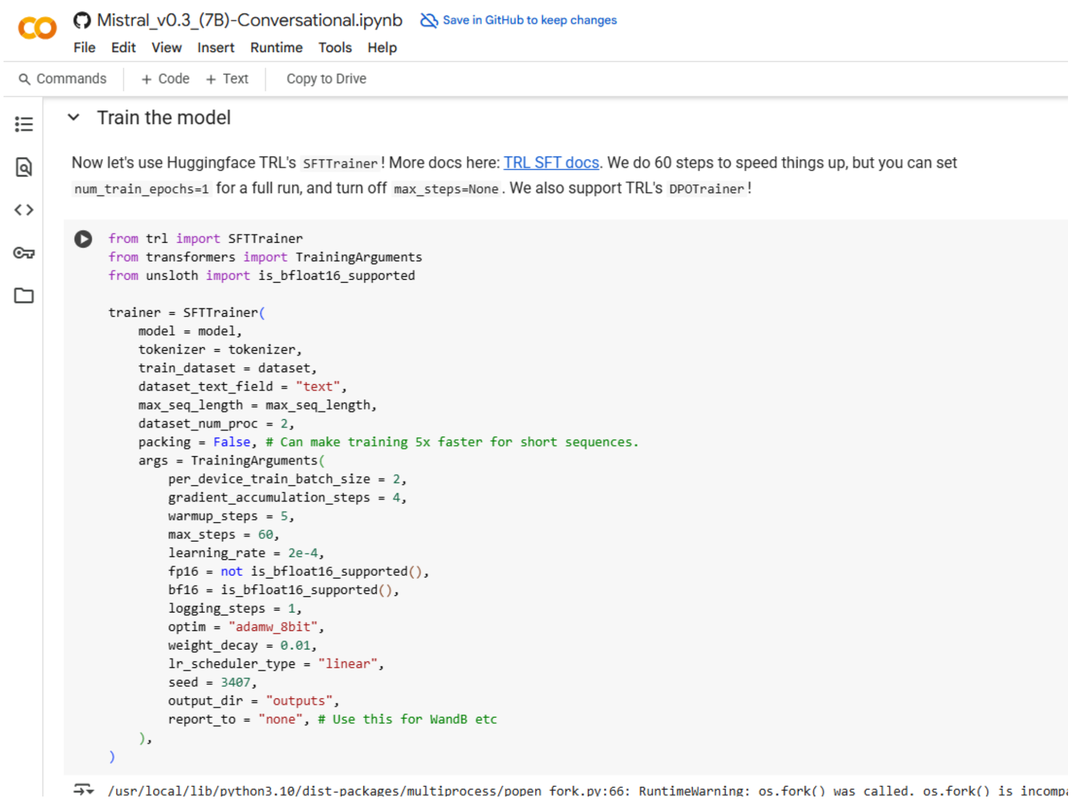
4. **Model Training:** Models were trained using Hugging Face's SFTTrainer, which is efficient for supervising fine-tuning of instruction-tuned models. The training was mainly focused on the assistant responses only using train_on_responses_only () function, ignoring user messages during loss computation. The model's learning process using the training arguments was updated severally until the last choice with less hallucination was made. The per device train batch size was set to 2, which indicates how many samples are propagated through in each forward and backward pass per GPU. To simulate a larger effective batch size without exceeding memory constraints, gradient_accumulation_steps was set to 4. This allowed the model to accumulate 4 mini-batches of gradients before one optimization step, which essentially simulates a batch size of 8.
A warm-up step of 5 was used for the stabilization of early training by incrementally increasing the learning rate from zero. The primary learning rate was set to a conservative value of 2e-5 to prevent overshooting with converged stability. A linear learning rate scheduler was used to decrease the learning rate linearly with respect to time.
To determine how long the training would take, the model was trained for two full epochs (num_train_epochs = 2), allowing it to go through all training samples twice. The max_steps were set to -1 to enable the model train fully according to the number of epochs.
For the best utilization of the GPU, mixed precision training was enabled with either fp16 or bf16, depending on the GPU hardware capabilities. The code would automatically detect the supported precision format. The optimizer employed was "adamw_8bit", an optimized variant of AdamW operating in 8-bit precision with little loss in terms of performance. Also, to improve generalization and avoid overfitting, L2 weight regularization (weight_decay = 0.01) was employed[45][46].
Below are the parameters used in the several retrains.

Arguments	Train 1	Train 5	Train 10	Train 15	Train 20	Final Train
per_device_train_batch_size	2	2	2	4	4	4
gradient_accumulation_steps	4	2	8	4	4	4
warmup_steps	5	5	50	50	100	100
num_train_epochs	None	2	None	None	None	5
max_steps	-1	-1	1000	2000	1000	1000
learning_rate	2e-4	3e-4	2e-4	3e-5	3e-5	3.5e-5
weight_decay	0.01	0.001	0.01	0.01	0.01	0.01
lr_scheduler_type	Linear	Linear	Cosine	Cosine	Linear	Linear

Table 1: Mistral Model Trainer Arguments

Arguments	Train 1	Train 5	Train 10	Train 15	Train 20	Final Train
per_device_train_batch_size	2	2	4	4	4	4
gradient_accumulation_steps	2	4	2	4	4	4
warmup_steps	5	10	50	50	100	100
num_train_epochs	None	2	None	2	4	5
max_steps	-1	1000	-1	1000	1000	1000
learning_rate	2e-4	3e-4	4e-5	1.5e-5	1e-5	2e-5
weight_decay	0.01	0.01	0.001	0.001	0.01	0.01
lr_scheduler_type	Linear	Linear	Cosine	Cosine	Cosine_ with _restarts	cosine

Table 2: LLAMA Model Trainer Arguments


Mistral_v0.3_(7B)-Conversational.ipynb
Save in GitHub to keep changes
File Edit View Insert Runtime Tools Help
Commands + Code + Text Copy to Drive

Train the model

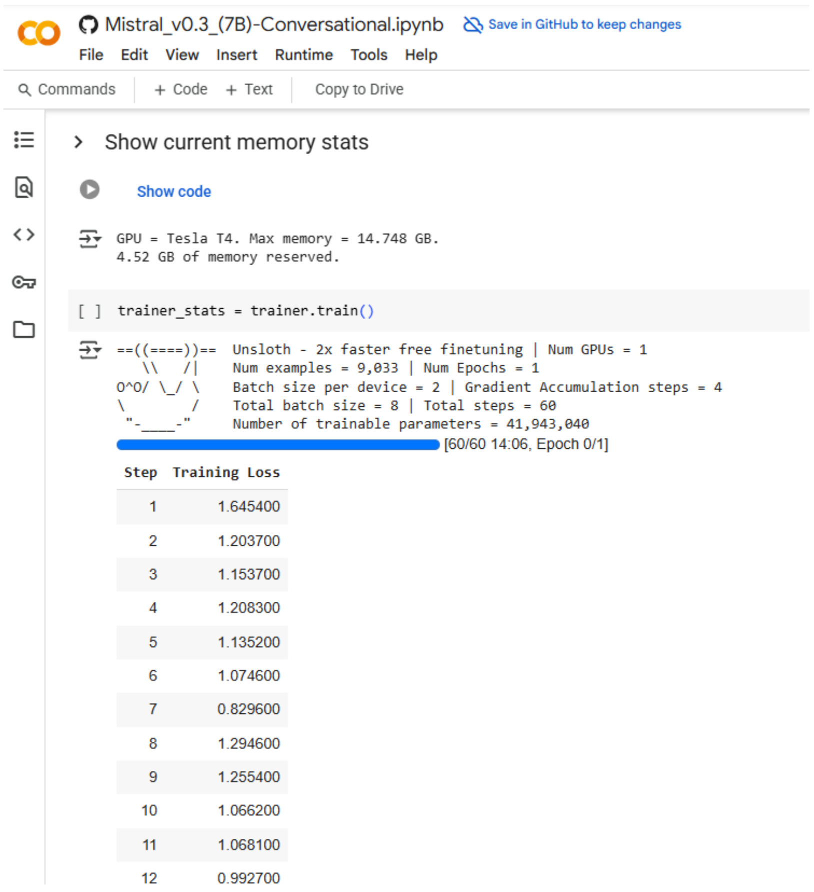
Now let's use Huggingface TRL's `SFTTrainer`! More docs here: TRL SFT docs. We do 60 steps to speed things up, but you can set `num_train_epochs=1` for a full run, and turn off `max_steps=None`. We also support TRL's `DPOTrainer`!

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        max_steps = 60,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
        report_to = "none", # Use this for WandB etc
    ),
)
```

```
/usr/local/lib/python3.10/dist-packages/multiprocess/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompa
```

Figure 5: Setting Arguments on Unsloth


Mistral_v0.3_(7B)-Conversational.ipynb
Save in GitHub to keep changes
File Edit View Insert Runtime Tools Help
Commands + Code + Text Copy to Drive
Show current memory stats
Show code

```
GPU = Tesla T4. Max memory = 14.748 GB.
4.52 GB of memory reserved.
```

```
[ ] trainer_stats = trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 9,033 | Num Epochs = 1
O^O/ \_/ \    Batch size per device = 2 | Gradient Accumulation steps = 4
\        /    Total batch size = 8 | Total steps = 60
 "-____-"     Number of trainable parameters = 41,943,040
[60/60 14:06, Epoch 0/1]
```

Step	Training Loss
1	1.645400
2	1.203700
3	1.153700
4	1.208300
5	1.135200
6	1.074600
7	0.829600
8	1.294600
9	1.255400
10	1.066200
11	1.068100
12	0.992700

Figure 6: Model Training showing loss

5. **Evaluation:** During training, weights & biases (WandB) logging was configured to capture loss, learning rate, epoch, global step and grad norm metrics every 10-training step for real-time visualization. After both models were trained, test questions were asked regarding the capstone course and the responses generated were streamed token-by-token using a text streamer. However, it was noticed the models were providing some correct answers yet, hallucinating some answers not in the training data. This led to continuous argument update on the SFTT Trainer until the model with less hallucination was chosen.

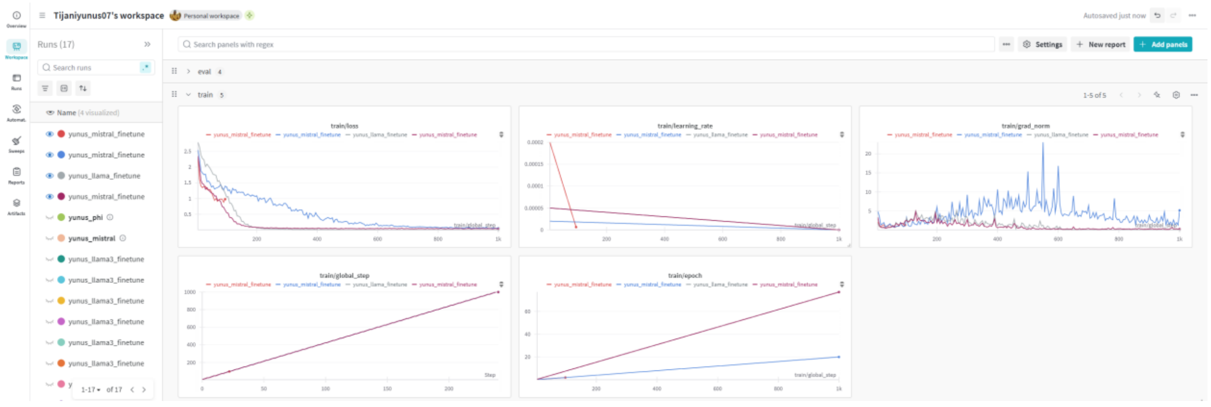
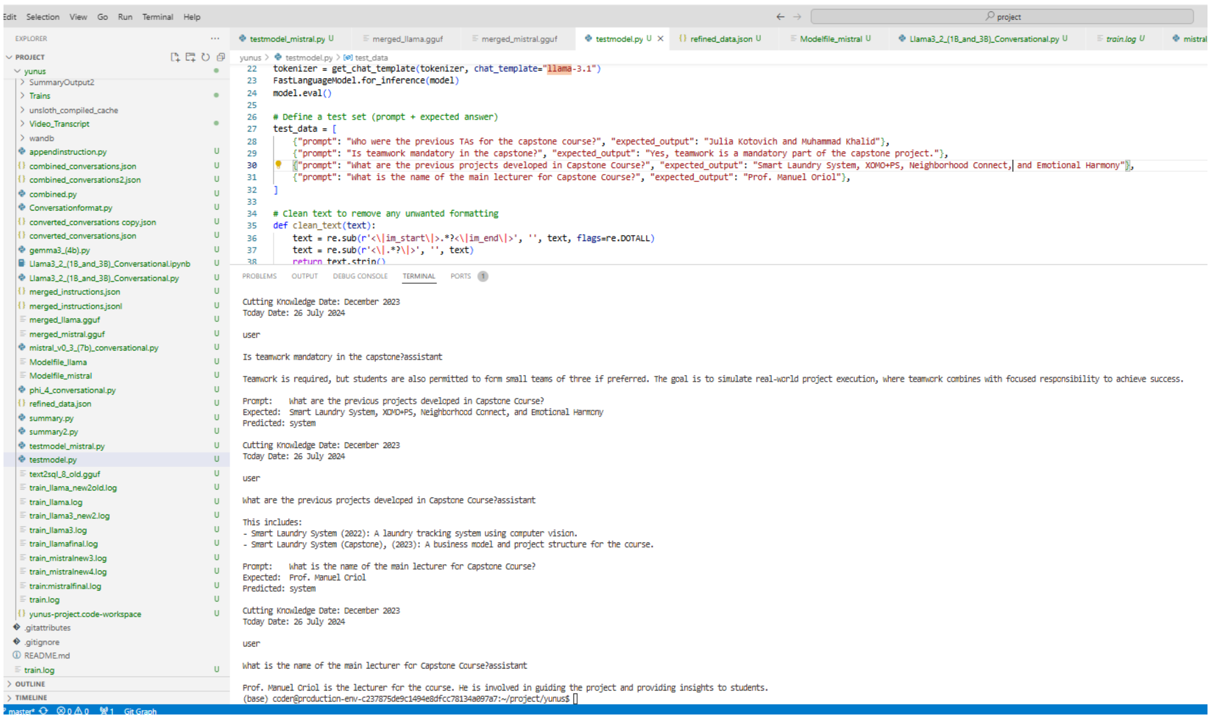


Figure 7: Monitoring Trainings on Weights and Bias (WandB)



```python
tokenizer = get_chat_template(tokenizer, chat_template="llama-3.1")
FastLanguageModel.for_inference(model)
model.eval()

# Define a test set (prompt + expected answer)
test_data = [
    {"prompt": "Who were the previous TAs for the capstone course?", "expected_output": "Julia Kotovich and Muhammad Khalid"},
    {"prompt": "Is teamwork mandatory in the capstone?", "expected_output": "Yes, teamwork is a mandatory part of the capstone project."},
    {"prompt": "What are the previous projects developed in Capstone Course?", "expected_output": "Smart Laundry System, XOMO+PS, Neighborhood Connect, and Emotional Harmony"},
    {"prompt": "What is the name of the main lecturer for Capstone Course?", "expected_output": "Prof. Manuel Oriol"},
]

# Clean text to remove any unwanted formatting
def clean_text(text):
    text = re.sub(r'<\|im_start\|>.*?<\|im_end\|>', '', text, flags=re.DOTALL)
    text = re.sub(r'<\|.*?\|>', '', text)
    return text.strip()
```

```
Cutting Knowledge Date: December 2023
Today Date: 26 July 2024

user

Is teamwork mandatory in the capstone?assistant

Teamwork is required, but students are also permitted to form small teams of three if preferred. The goal is to simulate real-world project execution, where teamwork combines with focused responsibility to achieve success.

Prompt:    What are the previous projects developed in Capstone Course?
Expected:  Smart Laundry System, XOMO+PS, Neighborhood Connect, and Emotional Harmony
Predicted: system

Cutting Knowledge Date: December 2023
Today Date: 26 July 2024

user

What are the previous projects developed in Capstone Course?assistant

This includes:
- Smart Laundry System (2022): A laundry tracking system using computer vision.
- Smart Laundry System (Capstone), (2023): A business model and project structure for the course.

Prompt:    What is the name of the main lecturer for Capstone Course?
Expected:  Prof. Manuel Oriol
Predicted: system

Cutting Knowledge Date: December 2023
Today Date: 26 July 2024

user

What is the name of the main lecturer for Capstone Course?assistant

Prof. Manuel Oriol is the lecturer for the course. He is involved in guiding the project and providing insights to students.
(base) coder@production-env-c237875de9c1494e8dfcc78134a097a7:~/project/yunus$ 
```

Figure 8: Evaluating Model by asking test questions

6. **Saving Fine-Tuned Models:** Fine-tuned Low-Rank Adaptation (LoRA) adapters generated after training phase were saved using save_pretrained() function in float16 format allowing deployment in lightweight inference engines like llama.cpp, and Ollama.

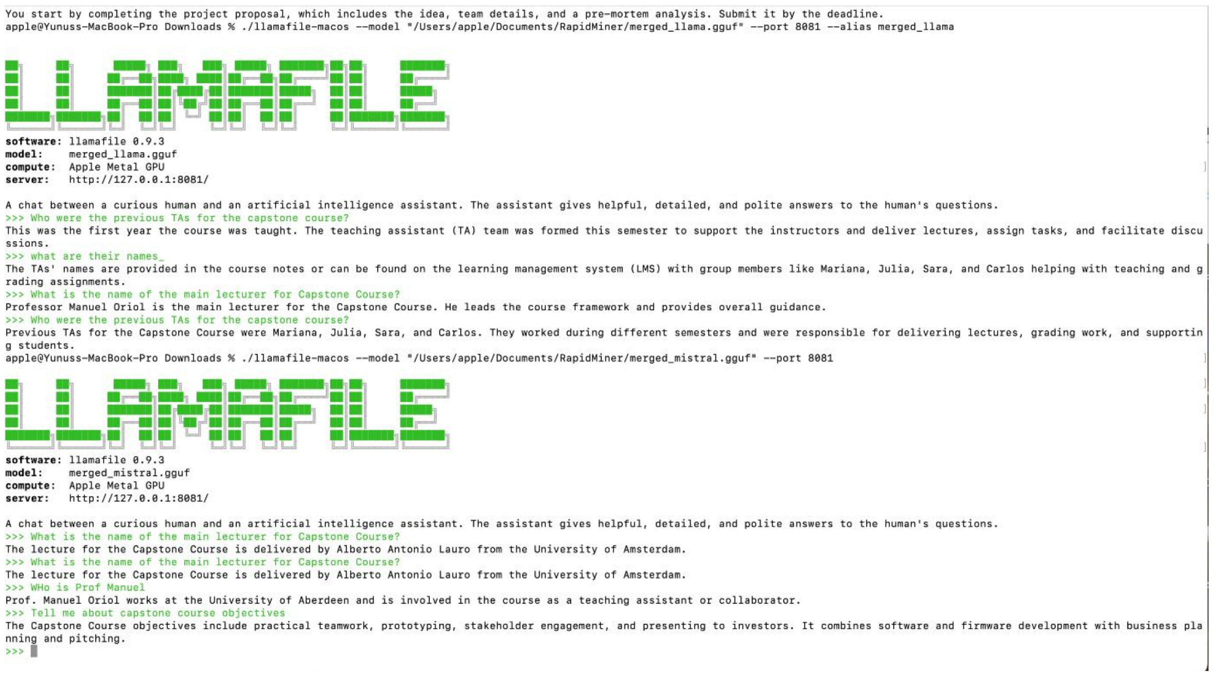
3.8 Deployment of Fine-Tuned Model to OpenWeb UI

3.8.1 LoRA Adapter Conversion and Deployment via LlamaFile

After the LoRA adapters were exported out from all the model training, merged base model and LoRA weights were exported through llama.cpp. The llama.cpp repository was cloned and all its dependencies with the git clone command, compiling the code with CUDA support.

After the environment was set up, the combined model was then quantized to the General Unified Format (GGUF) format by the script convert_hf_to_gguf.py. The function command specified the directory where the combined_model was located, the output file to be "merged_llama.gguf" and "merged_mistral.gguf", with the quantization method (q8_0) for reduced model size as well as accelerated testing without performance loss.

Deployment was done using LLamaFile, on the command line interface (CLI). The GGUF model was invoked in the context of a Modelfile, a configuration file for Ollama. This contained the base model path which registered the model in the local Ollama registry and supported model testing through direct invocation via CLI[47].



```
You start by completing the project proposal, which includes the idea, team details, and a pre-mortem analysis. Submit it by the deadline.
apple@Yunuss-MacBook-Pro Downloads % ./llamafile-macos --model "/Users/apple/Documents/RapidMiner/merged_llama.gguf" --port 8081 --alias merged_llama

LLAMAFILE

software: llamafile 0.9.3
model:    merged_llama.gguf
compute:  Apple Metal GPU
server:   http://127.0.0.1:8081/

A chat between a curious human and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the human's questions.
>>> Who were the previous TAs for the capstone course?
This was the first year the course was taught. The teaching assistant (TA) team was formed this semester to support the instructors and deliver lectures, assign tasks, and facilitate discu
ssions.
>>> what are their names_
The TAs' names are provided in the course notes or can be found on the learning management system (LMS) with group members like Mariana, Julia, Sara, and Carlos helping with teaching and g
rading assignments.
>>> What is the name of the main lecturer for Capstone Course?
Professor Manuel Oriol is the main lecturer for the Capstone Course. He leads the course framework and provides overall guidance.
>>> Who were the previous TAs for the capstone course?
Previous TAs for the Capstone Course were Mariana, Julia, Sara, and Carlos. They worked during different semesters and were responsible for delivering lectures, grading work, and supportin
g students.
apple@Yunuss-MacBook-Pro Downloads % ./llamafile-macos --model "/Users/apple/Documents/RapidMiner/merged_mistral.gguf" --port 8081

LLAMAFILE

software: llamafile 0.9.3
model:    merged_mistral.gguf
compute:  Apple Metal GPU
server:   http://127.0.0.1:8081/

A chat between a curious human and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the human's questions.
>>> What is the name of the main lecturer for Capstone Course?
The lecture for the Capstone Course is delivered by Alberto Antonio Lauro from the University of Amsterdam.
>>> What is the name of the main lecturer for Capstone Course?
The lecture for the Capstone Course is delivered by Alberto Antonio Lauro from the University of Amsterdam.
>>> WHo is Prof Manuel
Prof. Manuel Oriol works at the University of Aberdeen and is involved in the course as a teaching assistant or collaborator.
>>> Tell me about capstone course objectives
The Capstone Course objectives include practical teamwork, prototyping, stakeholder engagement, and presenting to investors. It combines software and firmware development with business pla
nning and pitching.
>>> 
```

Figure 9: Deployment via LLamaFile

3.8.2 Deployment of Ollama Model to Open WebUI

Open WebUI is an open-source, self-hosted web interface designed to interact with large language models (LLMs) providing a friendly chat experience with the model. Open WebUI was downloaded via docker and it is designed to automatically detect and connect to a running llama.cpp instance running on the machine using the default configuration (http://127.0.0.1:8081) for backend and (http://127.0.0.1:3000) for the frontend. From the Open WebUI dashboard, I navigated connections on admin settings, and I configured my selected model.

With the fine-tuned model successfully loaded, I conducted tests allowing other students to interact with the system through a familiar chat interface, making it suitable for question and answering of capstone course[47].

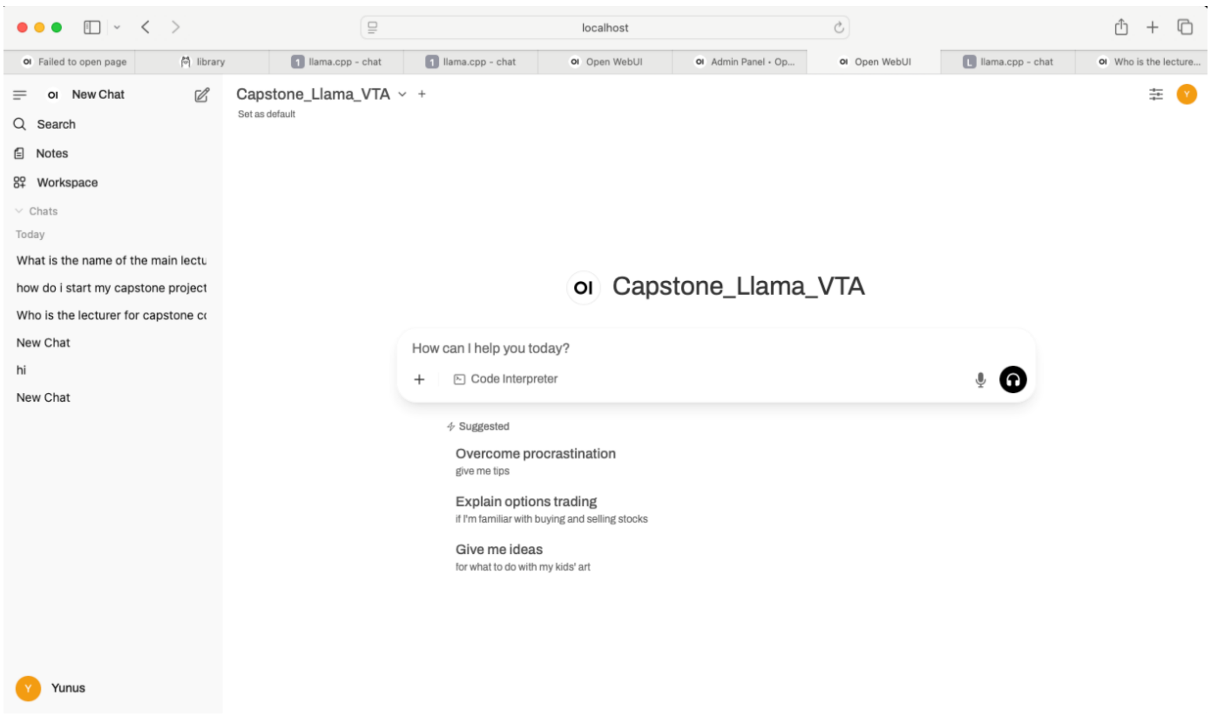


Figure 10: Deployment to OpenWeb UI

Chapter 4

Evaluation and Results

This chapter discusses the results of the fine-tuned models after finding the least hallucinated version and the evaluation metrics used to check its precision, recall and accuracy.

4.1 Testing Overview

I used a python script to test both models each time they were trained to check the model's response. During training, several checkpoints are generated based on number of epochs or maximum step in the local adapter to save the model locally on the system. This checkpoint is loaded and tested using Chat ML and LLaMA 3.1 templates for both models. Four test questions and their expected answers related to the capstone course were loaded into the script and the response determines the model efficiency, along with the visualization from Weighs and Bias (WandB), checking if the model was overfitted or underfitted.
Each question is passed to the model, and the generated output is decoded, cleaned of special tokens, and printed out. These generated predictions are then compared with the expected response. The evaluation of the model was split into two categories, quantitative and qualitative which is discussed below. Also, the evaluation code is found in Appendix E.

4.2 Training Progress Analysis

The several trainings done on both models were logged on Weights and Bias (WandB) to monitor their training behaviors and performance trajectory based on each training logs with visualization tools. These logs provided insights into how the model evolved throughout the number of steps, or full epoch training covering key aspects such as loss reduction, learning rate decay, gradient stability, and evaluation consistency.
The training loss curves measures how well the model minimized its prediction error during both training. A steady and consistent decline in training loss indicates effective learning from the training data. The training loss decreased sharply at the beginning and gradually flattened, suggesting that the model learned efficiently and approached convergence.

The learning rate curve shows the learning schedule applied during training. The plotted curve for both models shows an initial rise followed by a controlled decrease, confirming that the learning rate was well-regulated throughout the process.
Gradient norm plots show the stability of the training. Rough spikes in gradient norms indicate unstable learning. In both models, fluctuations were observed, but the gradient norm gradually stabilized as training progressed, indicating that the model's parameter updates became more consistent toward the end of training.
Global step and epoch plots provide a timeline of the training process. With approximately 1,000 global steps completed and more than 40 epochs covered, the models were trained long enough to train the dataset multiple times.

Below are the train diagrams for both models:

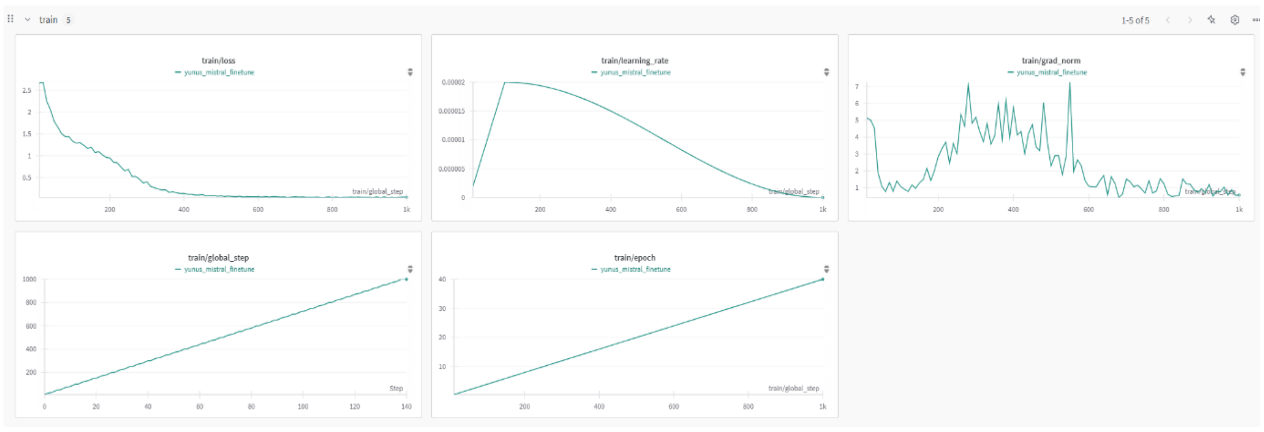
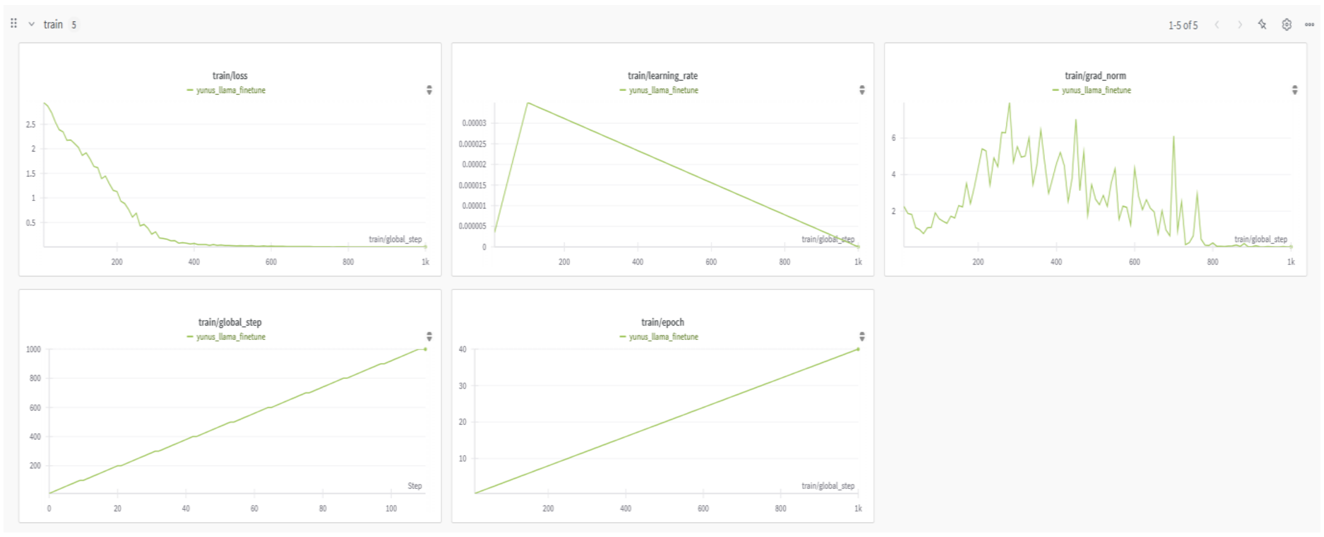


Figure 11: Mistral Training Curves on WandB



12: LLAMA Training Curves on WandB

4.3 Quantitative Evaluation

The model's performance was measured using metrics like BLEU, ROUGE, and BERTScore. These metrics evaluate different aspects such as token overlap, fluency, recall, coherence, and semantic similarity. Below, these metrics will be discussed in detail:

1. BLEU (Bilingual Evaluation Understudy)

BLEU is a precision-based metric used for machine translation tasks. It compares n-grams (up to 4-grams) in the predicted output against one or more reference texts. It measures how many n-gram overlaps exist between the generated text and the reference and penalizes repetition by using a clipped count of n-grams[48]. The metric outputs a score between 0 and 1, with higher values indicating closer matches to the reference. However, BLEU does not evaluate grammatical correctness.

Formula:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Equation 1: BLEU Formula

where:

- p_n = modified precision for n-grams
- w_n = weight for each n-gram level (usually uniform)
- BP = brevity penalty

Metrics	**Score**	**Result**
BLEU	0.1412	Low score with reasonable token match.

Table 3: BLEU Score for Final Finetuned Mistral Model

After several series of retrains, the model with the least hallucinations achieved a BLEU score of 0.1412, which is the highest that was gotten and it acceptable for conversational and open-ended question answering tasks. The score indicates that the model produces relevant answers that partially match the expected token sequences but may differ due to paraphrasing.

Metrics	**Score**	**Result**
BLEU	0.0304	Very low score with little to no token overlap.

Table 4: BLEU Score for Final Finetuned LLAMA Model

After multiple training iterations, the LLaMA model achieved a BLEU score of 0.0304, which is considered very poor. This score indicates that the generated responses had low token overlap with the reference answers. However, in the context of open-ended and conversational tasks, a low BLEU score does not imply poor performance. It often reflects the model's tendency to paraphrase rather than replicate the exact wording of the reference.

2. ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE is a recall-based metric often used in text summarization tasks. It measures overlap between generated summaries and reference summaries in terms of n-grams, longest common subsequence, and skip bigrams[49]. The metric outputs a score between 0 and 1, with higher values indicating a high overlap to the reference.

ROUGE-1: It measures the overlap of unigrams (single words) between the generated and reference text. It evaluates whether the key individual words in the reference are present in the generated output. A high score indicates good keyword coverage.

Formula:

$$\text{ROUGE-1} = \frac{\text{Number of overlapping unigrams}}{\text{Total unigrams in the reference}}$$

Equation 2: ROUGE-1 Formula

ROUGE-2: It measures the overlap of bigrams (two-word sequences) between the generated and reference texts. This Evaluates fluency and the ability to preserve short phrases and local word order. A high score shows phrase coherence.

Formula:

$$\text{ROUGE-2} = \frac{\text{Number of overlapping bigrams}}{\text{Total bigrams in the reference}}$$

Equation 3: ROUGE-2 Formula

ROUGE-L: It measures the Longest Common Subsequence (LCS) between the candidate and reference text. This captures the structure and sentence-level similarity by identifying the longest ordered sequence of words shared by both texts. A high score shows how well the structure is preserved, not just putting lots of words together.

Formula:

$$\text{ROUGE-L} = \frac{\text{Length of LCS}}{\text{Length of reference sentence}}$$

Equation 4: ROUGE-L Formula

ROUGE-Lsum: This measures the summary of multiple sentences by computing LCS across sentences, rather than treating the whole text as one long sequence. It used for evaluating multi-turn responses, where each sentence matters individually. This is like ROUGE-L, but more accurate in summarization-style tasks.

Metrics	**Score**	**Result**
ROUGE-1	0.3142	Good unigram overlap (keywords preserved).
ROUGE-2	0.2583	Moderate bigram overlaps (some fluency captured)
ROUGE-L	0.3028	Captures good sequence similarity in a sentence.
ROUGE-Lsum	0.3142	Captures good sequence similarity in multiturn conversation

Table 5: ROUGE Score for Final Finetuned Mistral Model

After several model retrains, the ROUGE evaluation scores obtained for the final fine-tuned Mistral model indicate a good coverage of content relevance and structural coherence in its generated responses. The ROUGE-1 score of 0.3142 reflects a good number of unigram (word-level) overlap between the predicted and reference texts, suggesting that the model successfully captures key vocabulary and factual elements from the original responses. Meanwhile, the ROUGE-2 score of 0.2583 shows that the model maintains a moderate level of fluency and phrase-level consistency, as it measures the overlap of word pairs. The ROUGE-L score of 0.3028, which considers the longest common subsequence between predicted and reference responses, indicates that the model preserves the structure of the reference answers. Similarly, the ROUGE-Lsum score of 0.3142 shows that this structural coherence is good enough even when evaluating multi-sentence outputs. These scores suggest that the model produces good and relevant responses but could be higher with more trainings.

Metrics	**Score**	**Result**
ROUGE-1	0.1365	Low unigram overlap
ROUGE-2	0.0805	Very poor bigram overlap
ROUGE-L	0.1274	Captures low sequence similarity in a sentence.
ROUGE-Lsum	0.1365	Captures low sequence similarity in multiturn conversation

Table 6: ROUGE Score for Final Finetuned LLAMA Model

The ROUGE evaluation scores for the fine-tuned LLaMA model indicate a modest level of content relevance and structural alignment in its generated responses. The ROUGE-1 score of 0.1365 reflects a low unigram overlap, suggesting that the model isn't doing great to capture key words from the reference responses. ROUGE-2 score of 0.0805 shows that the model maintains a very poor level of fluency and phrase-level consistency. The ROUGE-L score of 0.1274, lacks consistent sentence-level coherence. Similarly, the ROUGE-Lsum score of 0.1365 shows that multi turn sentences perform very poorly. These scores suggest that the model produces poor responses when compared to Mistral Model but could be higher with more trainings.

3. BERTScore

BERTScore evaluates how semantically similar a generated sentence is to a reference sentence by using word embeddings from pre-trained transformer models like BERT[50]. It used cosine similarity to compare the meaning of words in context, recognizes paraphrases and semantically equivalent phrases that BLEU and ROUGE would miss. It computes Precision, Recall, and F1 based on matching vectors instead of token overlap. The Precision measures how much of the predicted sentence is semantically covered by the reference, while Recall measures how much of the reference sentence is semantically recovered in the prediction and the F1 Score measures the mean of Precision and Recall. The metric outputs a score between 0 and 1, with higher values indicating a high semantic match to the reference.

Formula:

$$\text{Precision} = \frac{1}{|y|} \sum_{y_i \in y} \max_{x_j \in x} \cos(y_i, x_j)$$

$$\text{Recall} = \frac{1}{|x|} \sum_{x_j \in x} \max_{y_i \in y} \cos(x_j, y_i)$$

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Equation 5: BertScore Formula

Metrics	**Score**	**Result**
Precision	0.843	The answer generated is relevant to reference.
Recall	0.9261	Generated answers contain key ideas present in reference.
F1-Score	0.8824	Very good semantic match.

Table 7: BertScore for Final Finetuned Mistral Model

The BERT Scores gotten from the several retrains of the fine-tuned model reveal a high score of semantic similarity between the predictions and the reference answers. The Precision score of 0.843, shows that the content generated is meaningful when compared to the reference. The Recall score of 0.9261, shows a very high score meaning the model is very good at capturing key information present in the reference responses. The F1 Score of 0.8824, shows a strong overall semantic alignment. This score confirms that the model generates accurate and semantically correct responses.

Metrics	**Score**	**Result**
Precision	0.8095	The answers generated are relevant to reference.
Recall	0.8815	Most generated answers contained key ideas present in reference.
F1-Score	0.8437	Good semantic match.

Table 8: BertScore for Final Finetuned LLAMA Model

The BERT Score result above is achieved after multiple retraining with the Precision score of 0.8095 indicating that the content produced by the model is relevant when compared to the expected output. The Recall score of 0.8815 suggests that the model is effective at using key words present in the reference. Meanwhile, the F1 Score of 0.8437, which balances both precision and recall, confirms a very good overall semantic match.

4.4 Qualitative Evaluation

In addition to numerical metrics, a qualitative analysis of several sample outputs was looked at to evaluate both model's fluency, accuracy, and adherence to Capstone knowledge.

Prompt	**Expected**	**Predicted**	**Result**
Who were the previous TAs for the capstone course?	Julia Kotovich and Muhammad Khalid	The previous TAs were Julia Kotovich and Muhammad Khalid. Julia was	Excellent response. Went further to giving details.

		responsible for grading deliverables and providing feedback, while Muhammad focused on mentoring students and supporting software development.	
Is teamwork mandatory in the capstone?	Yes, teamwork is a mandatory part of the capstone project.	Yes, teamwork is a core part of the Capstone Project. Students are expected to work in teams, collaborate on deliverables, and present their work to stakeholders.	A very accurate and clear response
What are the previous projects developed in Capstone Course?	Smart Laundry System, XOMO+PS, Neighborhood Connect, and Emotional Harmony	The previous projects developed include Smart Laundry System, XOMO+PS, Neighborhood Connect, and Emotional Harmony. Students build on these experiences when proposing project ideas.	Excellent and accurate recall of projects names from dataset.
What is the name of the main lecturer for Capstone Course?	Prof. Manuel Oriol	The main lecturer is Prof. Manuel Oriol, he is responsible for guiding the team's product vision, maintaining the backlog, and facilitating sprint reviews and planning sessions every two weeks.	Very detailed response based on how it was trained.

Table 9: Output for Mistral Model Testing

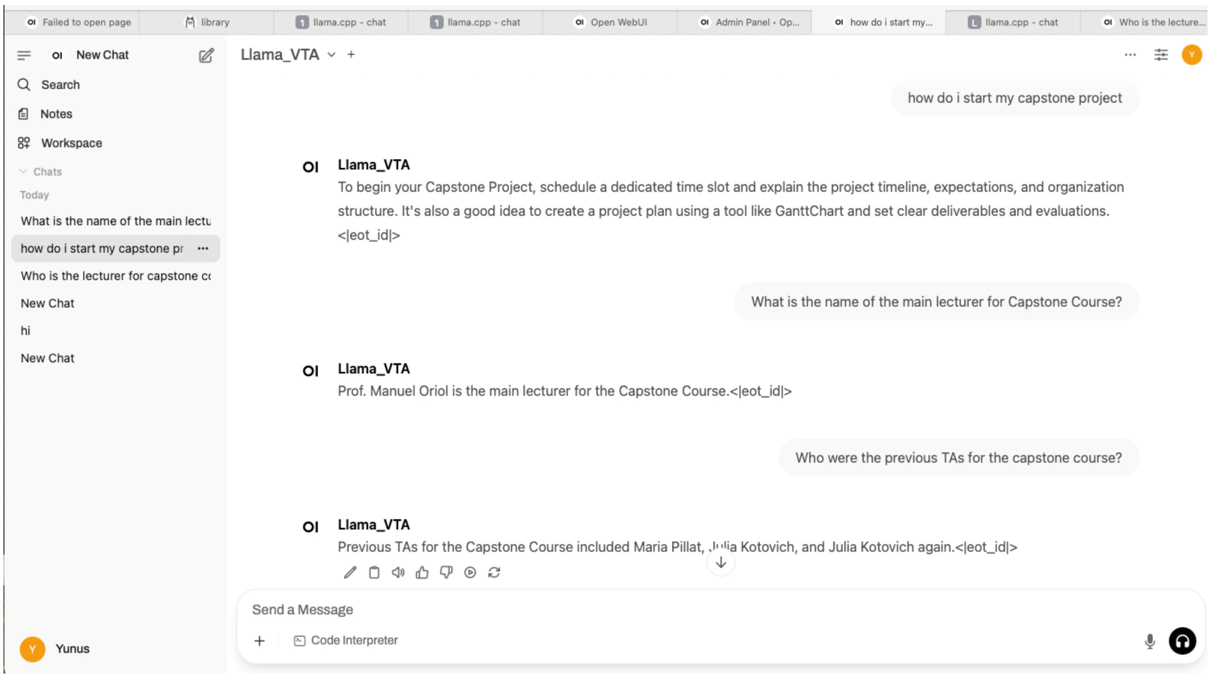
The responses generated for the four test questions were accurate. For instance, the model correctly identified *"Julia Kotovich and Muhammad Khalid"* and talked more on their TAs roles. This response showed the models ability to generalize from training data. The responses show that the model does not replicate memorized answers but uses relevant information to form a sound response. Also, all the responses generated were fluent and grammatical correct. Sentences were well structured, beginning responses with direct answers and following up with explanatory details. The responses showed a strong understanding of capstone course terminologies such as *"sprint reviews", and "stakeholders"*. This indicates vocabulary and instructional flow of the Capstone course was well adapted.

Prompt	**Expected**	**Predicted**	**Result**
Who were the previous TAs for the capstone course?	Julia Kotovich and Muhammad Khalid	The previous TAs were Julia Kotovich and Irina Sudzika.	Partial correct answer.
Is teamwork mandatory in the capstone?	Yes, teamwork is a mandatory part of the capstone project.	Teamwork is required, but students are initially grouped by teams. They can later self-assess their team dynamics and provide feedback on collaboration structure and effectiveness.	Fair response with an understand of teamwork dynamics in a team for capstone.
What are the previous projects developed in Capstone Course?	Smart Laundry System, XOMO+PS, Neighborhood Connect, and Emotional Harmony	This project includes core components such as team structure, project planning, and the creation of a high-quality product using Python. Students are expected to build a real-world application or system	Poor response. No names of projects designed.

		over multiple months, demonstrating technical, technical communication, and project management skills.	
What is the name of the main lecturer for Capstone Course?	Prof. Manuel Oriol	Prof. Manuel Oriol is the lecturer for the course. He is involved in guiding the project work and providing insights through external sessions.	Good response.

Table 10: Output for LLAMA Model Testing

The responses generated by the fine-tuned LLaMA model for the four test questions shows partial accuracy and a moderate grasp of domain context. For instance, the model correctly identified Julia Kotovich but hallucinated Irina Sudzika. This indicates the model ability to provided wrong details. Also, the model lacks factual recall and struggles with retrieving specific names from memory as shown in the test questions regarding previous projects.



how do i start my capstone project

Llama_VTA
To begin your Capstone Project, schedule a dedicated time slot and explain the project timeline, expectations, and organization structure. It's also a good idea to create a project plan using a tool like GanttChart and set clear deliverables and evaluations. <|eot_id|>

What is the name of the main lecturer for Capstone Course?

Llama_VTA
Prof. Manuel Oriol is the main lecturer for the Capstone Course.<|eot_id|>

Who were the previous TAs for the capstone course?

Llama_VTA
Previous TAs for the Capstone Course included Maria Pillat, Julia Kotovich, and Julia Kotovich again.<|eot_id|>

Figure 13: Outputs on OpenWeb UI

4.5 Error Analysis and Hallucination Examples

In the process of testing both models, several recurring patterns of error were identified. These errors mostly relate to factual inaccuracy, low recall performance, hallucinations, and semantic deviation from reference answers.

Lack of factual recall refers to a model's inability to retrieve or reproduce specific information that it has either seen during training or that is expected in each context.

Factual inaccuracy refers to any output that is incorrect or inconsistent with the truth. It occurs when the model gives a response that is wrong based on the known facts, ground truth, or reference material.

Hallucination is a specific type of factual inaccuracy where the model "makes things up". It generates information that sounds plausible but has no basis in the training data or reality.

Below are a few examples that show the types of errors encountered with both models:

Prompt	**Expected**	**Predicted**	**Result**
Who were the previous TAs for the capstone course?	Julia Kotovich and Muhammad Khalid	I'm unable to identify the previous TAs for the capstone course.	Lack of factual recall
		Previous tutors included Thomas Okker and Julia Kotovich, who helped guide and develop the students' project work.	Hallucination
Is teamwork mandatory in the capstone?	Yes, teamwork is a mandatory part of the capstone project.	Teamwork is not mandatory in the traditional sense, as students can choose to work independently. However, the requirement encourages collaboration and supports virtual communication tools, such as Slack, for team coordination.	Factual Inaccuracy.

		Teamwork is required, but students are also allowed to work independently. The choice between collaborating or going solo is up to each student's team dynamics and project planning.	Factual Inaccuracy.
What are the previous projects developed in Capstone Course?	Smart Laundry System, XOMO+PS, Neighborhood Connect, and Emotional Harmony	This course has seen several project editions. Previous versions included the Capstone Project 2018/2019 and Capstone Project 2019/2020. The Capstone Project course builds on the foundation of two earlier Capstone courses, Capstone I and Capstone II. These projects introduce real-world development experience using agile methodologies and customer-centric design. This includes Smart Laundry System, Virtual Flight Planner, and Emart. Each project was focused on different fields such as technology (flight planning), industry (laundry), and services (online marketing), exploring various functionalities and collaborations among team members.	Lack of factual recall Lack of factual recall Hallucinations

What is the name of the main lecturer for Capstone Course?	Prof. Manuel Oriol	Prof. Manuel Oriol is the lecturer for the course. He is available at Manuel.Oriol@cs.tu.darmstadt.de or @oriolmanuel on Xoomer or Twitter. The main lecturer for the Capstone Course is Dr. Rachel Brown.	Partial accuracy plus hallucinations. Hallucinations

Table 11: Error Analysis on both models

Chapter 5

Conclusion

I successfully finetuned and deployed two large language models named Mistral and LLaMA on the multimodal data gotten from the capstone course and tested it on OpenWeb UI. The data used video transcripts and slides from the capstone course, and it was processed by transcribing the videos and formatting all data into the required conversational format. I used Unsloth framework to finetune the data using Mistral and LLaMA model to test which model work best to provide contextually accurate, and grammatically fluent responses based on the course to act as a virtual teaching assistant.

The fine-tuning process was done severally by training the process repeatedly by updating the trainer arguments until I achieved the least hallucinated version for both models. The training process was monitored using Weights and Bias (WandB) to check the train loss curves, learning rate and gradient norm graphs to indicate if the trained model is underfitted or overfitted. The evaluation used metrics, including BLEU, ROUGE, and BERT Score, to measure the fluency, recall, precision and sentence similarity performance of the responses generated from the trained model. The responses generated was also reviewed to check for errors like factual inaccuracies and hallucinations. The metrics gotten from the several evaluations yielded several scores for both models but after comparison, it was clear that Mistral model is the best model for the project with a higher metric score, better accuracy and recall.

In summary, this thesis shows that fine-tuning open-source language models can be adapted with Unsloth framework for domain-specific assistance in Constructor Institute. Although challenges such as hallucinations and factual inaccuracies were noted in some responses, but the fluency and generalization of the generated responses show the potential of these models as a virtual teaching assistant in academic environments. Future work will continue to refine these systems toward greater accuracy, reliability, and educational value.

5.1 Future Works

Even though the results of this research are promising, several areas need improvement. Firstly, expanding the dataset and formatting it in different diversity to include more varied question styles, multi-turn dialogues, and data sources from multiple semesters or instructors. This would help the model generalize better and reduce overfitting.

Secondly, integrating the fine-tuned model into the university learning management system with feedback mechanisms would provide additional training data to the model and more support to the lecturers and students. This platform would allow students to interact with the model, giving researchers valuable insights into model limitations, and opportunities for continual learning.
Finally, integrating an attendance and homework management system would help keep track of student's participation, assignment submissions and deadline reminders. This would enable the lecturers have time to focus on curriculum while the VTA monitor students' attendance and even answers clarifying questions related to the assignment.

Miscellaneous

1. Github Link for Zoom Video Download - https://github.com/ricardorodrigues-ca/zoom-recording-downloader

2. Github Link to transcribe speech-based video files into clean text - https://github.com/pszemraj/vid2cleantxt/tree/master

3. Unsloth Google Colab Notebook for LLAMA model - https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Llama3.2_(1B_and_3B)-Conversational.ipynb

4. Unsloth Google Colab Notebook for Mistral model - https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Mistral_v0.3_(7B)-Conversational.ipynb

Appendix A

Document Text Extraction from PDFs and PowerPoint Presentations

```
import os
import fitz  # PyMuPDF for PDFs
from pptx import Presentation

def extract_text_from_pptx(file_path):
    """Extract text from a PowerPoint file."""
    prs = Presentation(file_path)
    text_data = []

    for slide in prs.slides:
        slide_text = []
        for shape in slide.shapes:
            if hasattr(shape, "text"):
                slide_text.append(shape.text.strip())
        text_data.append("\n".join(slide_text))

    return "\n\n".join(text_data)

def extract_text_from_pdf(file_path):
    """Extract text from a PDF file."""
    text_data = []

    with fitz.open(file_path) as doc:
        for page in doc:
            text_data.append(page.get_text("text"))  # Extract text from each page

    return "\n\n".join(text_data)
```

```
# Folder containing PPTX & PDF files
file_folder = r"C:\Users\yt\Downloads\CapstoneSlide"
output_file = "combined_extracted_text.txt"

# Open output file in write mode
with open(output_file, "w", encoding="utf-8") as outfile:
    for filename in os.listdir(file_folder):
        file_path = os.path.join(file_folder, filename)

        if filename.endswith(".pptx") or filename.endswith(".ppt"):
            print(f"Processing PowerPoint: {filename}")
            extracted_text = extract_text_from_pptx(file_path)
            file_type = "PowerPoint"

        elif filename.endswith(".pdf"):
            print(f"Processing PDF: {filename}")
            extracted_text = extract_text_from_pdf(file_path)
            file_type = "PDF"

        else:
            continue  # Skip non-PDF/PPTX files

        # Append extracted text to the output file
        outfile.write(f"--- Extracted from {file_type}: {filename} ---\n")
        outfile.write(extracted_text + "\n\n")

print(f"Text extraction complete! All extracted text saved in {output_file}")
```

Appendix B

Text Chunking for Scalable Processing

```
import os

input_path = "path/to/your/input.txt"
output_dir = "path/to/output/chunks"
lines_per_chunk = 10000

# Make sure the output directory exists
os.makedirs(output_dir, exist_ok=True)

# Read all non-blank lines
with open(input_path, 'r', encoding='utf-8') as infile:
    lines = [line for line in infile if line.strip() != ""]

# Split into chunks and write each to a new file
for i in range(0, len(lines), lines_per_chunk):
    chunk = lines[i:i + lines_per_chunk]
    chunk_filename = os.path.join(output_dir, f"chunk_{i // lines_per_chunk + 1}.txt")
    with open(chunk_filename, 'w', encoding='utf-8') as outfile:
        outfile.writelines(chunk)

print(f"Split into {len(lines) // lines_per_chunk + 1} chunk(s) successfully.")
```

Appendix C

Python Script for Instruction Generation and Semantic Matching.

```
import os
import json
import spacy
import pytextrank
from pathlib import Path
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import SentenceTransformer

# --- Configuration ---
INPUT_DIR = r"/home/coder/project/Video_Transcript/chunks"   # Folder with raw .txt
transcripts
OUTPUT_DIR = r"/home/coder/project/SummaryOutput"  # Folder to save processed
instruction data

# --- Ensure folders exist ---
os.makedirs(INPUT_DIR, exist_ok=True)
os.makedirs(OUTPUT_DIR, exist_ok=True)

# --- Load NLP + Embedding Models ---
nlp = spacy.load("en_core_web_sm")
nlp.max_length = 2_500_000  # Updated to handle long transcripts
nlp.add_pipe("textrank")
embedder = SentenceTransformer("all-MiniLM-L6-v2")  # Lightweight, fast model

# --- Utility Functions ---
def get_semantic_passage(phrase, text, top_k=2):
    """Find semantically similar passage for the given phrase."""
    sentences = [s.strip() for s in text.split(".") if len(s.strip().split()) > 5]
    if not sentences:
        return ""
    phrase_vec = embedder.encode([phrase])
    sent_vecs = embedder.encode(sentences)
    sims = cosine_similarity(phrase_vec, sent_vecs)[0]
    top_indices = sims.argsort()[::-1][:top_k]
    return ". ".join(sentences[i] for i in top_indices)

def clean_phrase(phrase):
    return ' '.join(phrase.strip().lower().replace("\n", " ").split())

def is_valid_phrase(phrase):
    phrase = clean_phrase(phrase)
    return len(phrase.split()) >= 2 and phrase.isascii() and not phrase.startswith("you")

# --- Processing ---
def process_transcript(filepath):
    with open(filepath, "r", encoding="utf-8") as f:
        text = f.read()

    doc = nlp(text)
    seen = set()
    examples = []

    for phrase in doc._.phrases:
        keyword = clean_phrase(phrase.text)
        if keyword in seen or not is_valid_phrase(keyword):
            continue
        seen.add(keyword)

        instruction = f"What is '{keyword}' about?"
        output = get_semantic_passage(keyword, text)

        if len(output.strip().split()) < 10:
            continue  # skip too short answers
        print(f"Valid phrase: {keyword} → {len(output.strip().split())} words in match")

        examples.append({
            "instruction": instruction,
            "input": "",
            "output": output.strip()
        })

    # Fallback if nothing found
    if not examples:
        fallback = text.strip().split("\n\n")[0]
        if len(fallback.split()) > 10:
            examples.append({
                "instruction": "Summarize the main idea of this segment.",
                "input": "",
                "output": fallback
            })

    return examples

# --- Run over folder ---
all_data = []
input_files = sorted(Path(INPUT_DIR).glob("*.txt"))

for file in tqdm(input_files, desc="Processing transcripts"):
    try:
        data = process_transcript(file)
        all_data.extend(data)

        print(f" Processed {file.name}: {len(data)} examples")

        # Save per file
        output_file = Path(OUTPUT_DIR) / f"{file.stem}_qa.jsonl"
        with open(output_file, "w", encoding="utf-8") as out:
            for item in data:
                json.dump(item, out)
                out.write("\n")
    except Exception as e:
        print(f" Error processing {file.name}: {e}")

# Optional: Save all combined
with open(Path(OUTPUT_DIR) / "all_instructions.jsonl", "w", encoding="utf-8") as out:
    for item in all_data:
        json.dump(item, out)
        out.write("\n")

print(f"Done! {len(all_data)} instructions written to {OUTPUT_DIR}")
```

Appendix D

<u>Merging Datasets and Conversional Structuring</u>

```
import os
import json

def convert_folder_to_conversational(input_folder, output_file):
    all_conversations = []
    file_count = 0
    entry_count = 0

    for filename in os.listdir(input_folder):
        if filename.endswith(".jsonl"):
            file_count += 1
            file_path = os.path.join(input_folder, filename)
            with open(file_path, "r", encoding="utf-8") as f:
                for line in f:
                    try:
                        item = json.loads(line.strip())
                        instruction = item.get("instruction", "")
                        input_text = item.get("input", "").strip()
                        user_msg = instruction if not input_text else f"{instruction}\n\n{input_text}"

                        convo = {
                            "conversations": [
                                {"role": "user", "content": user_msg.strip()},
                                {"role": "assistant", "content": item["output"].strip()}
                            ]
                        }
                        all_conversations.append(convo)
                        entry_count += 1
                    except Exception as e:
```

```
                    print(f" Skipping malformed line in {filename}: {e}")

    # Save all combined conversations
    with open(output_file, "w", encoding="utf-8") as out_file:
        json.dump(all_conversations, out_file, indent=2)

    print(f"Converted {entry_count} entries from {file_count} files.")
    print(f" Output saved to: {output_file}")

convert_folder_to_conversational(r"/home/coder/project/SummaryOutput/Processed",
"converted_conversations.json")
```

Appendix E

Evaluation Script

```
# !pip install bert_score

from unsloth import FastLanguageModel
from transformers import TextStreamer
from unsloth.chat_templates import get_chat_template
from tqdm import tqdm
import torch
import evaluate
import re
from bert_score import score as bertscore

# Load the fine-tuned LLaMA model and tokenizer
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "./outputs_llama3/checkpoint-1000",  # Local path to LLaMA checkpoint
    max_seq_length = 2048,
    dtype = None,
    load_in_4bit = True,
)

# Use LLaMA 3.1 chat template
tokenizer = get_chat_template(tokenizer, chat_template="llama-3.1")
FastLanguageModel.for_inference(model)
model.eval()

# Define a test set (prompt + expected answer)
test_data = [
    {"prompt": "Who were the previous TAs for the capstone course?", "expected_output":
"Julia Kotovich and Muhammad Khalid"},
    {"prompt": "Is teamwork mandatory in the capstone?", "expected_output": "Yes, teamwork
is a mandatory part of the capstone project."},
        {"prompt": "What are the previous projects developed in Capstone Course?",
"expected_output": "Smart Laundry System, XOMO+PS, Neighborhood Connect, and
Emotional Harmony"},
        {"prompt": "What is the name of the main lecturer for Capstone Course?",
"expected_output": "Prof. Manuel Oriol"},
]

# Clean text to remove any unwanted formatting
def clean_text(text):
    text = re.sub(r'<\|im_start\|>.*?<\|im_end\|>', '', text, flags=re.DOTALL)
    text = re.sub(r'<\|.*?\|>', '', text)
    return text.strip()

# Inference loop
predictions = []
references = []

for sample in tqdm(test_data):
    messages = [{"role": "user", "content": sample["prompt"]}]
    input_ids = tokenizer.apply_chat_template(
        messages, tokenize=True, add_generation_prompt=True, return_tensors="pt"
    ).to("cuda")

    with torch.no_grad():
        outputs = model.generate(input_ids=input_ids, max_new_tokens=128)
        decoded = tokenizer.decode(outputs[0], skip_special_tokens=True).strip()
        cleaned = clean_text(decoded)

    predictions.append(cleaned)
    references.append(sample["expected_output"])

# Load metrics
```

```
bleu = evaluate.load("bleu")
rouge = evaluate.load("rouge")

# Compute BLEU
bleu_result = bleu.compute(
    predictions=predictions,
    references=[[r] for r in references]
)

# Compute ROUGE
rouge_result = rouge.compute(
    predictions=predictions,
    references=references,
    use_stemmer=True
)

# Compute BERTScore
P_clean = [p.strip() for p in predictions]
R_clean = [r.strip() for r in references]
P_, R_, F1 = bertscore(P_clean, R_clean, lang="en", verbose=True)

# Display results
print("\nEvaluation Metrics")
print("-" * 40)
print("BLEU:", round(bleu_result["bleu"], 4))
print("ROUGE-1:", round(rouge_result["rouge1"], 4))
print("ROUGE-2:", round(rouge_result["rouge2"], 4))
print("ROUGE-L:", round(rouge_result["rougeL"], 4))
print("ROUGE-Lsum:", round(rouge_result["rougeLsum"], 4))
print("\n BERTScore:")
print("Precision:", round(P_.mean().item(), 4))
print("Recall:    ", round(R_.mean().item(), 4))
print("F1 Score: ", round(F1.mean().item(), 4))
```

```
# Show sample outputs
print("\n Sample Outputs")
print("-" * 40)
for i in range(len(test_data)):
    print(f"\nPrompt:    {test_data[i]['prompt']}")
    print(f"Expected:  {references[i]}")
    print(f"Predicted: {predictions[i]}")
```

Bibliography

[1] OpenAI, “GPT-4 Technical Report,” 2023. [Online]. Available: https://arxiv.org/abs/2303.08774

[2] Anthropic, “Building with Claude,” 2025. https://docs.anthropic.com/en/docs/overview

[3] H. Touvron *et al.*, “LLaMA: Open and Efficient Foundation Language Models,” 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[4] “Open WebUI.” Accessed: May 18, 2025. [Online]. Available: https://openwebui.com/

[5] G. Kress and T. van Leeuwen, *Multimodal Discourse: The Modes and Media of Contemporary Communication*. Arnold Publishers, 2001.

[6] P. Blikstein and M. Worsley, “Multimodal Learning Analytics and Education Data Mining: using computational technologies to measure complex learning tasks,” *Journal of Learning Analytics*, vol. 3, no. 2, 2016, doi: 10.18608/jla.2016.32.11.

[7] “Unsloth AI - Open-Source Fine-Tuning for LLMs.” Accessed: May 18, 2025. [Online]. Available: https://www.unsloth.ai/

[8] K. VanLehn, “The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems,” *Educ Psychol*, vol. 46, no. 4, pp. 197–221, 2011.

[9] V. K. Chaudhri, A. Aiken, M. Bauer, and T. Bhansali, “Teaching Assistants Powered by Large Language Models: Promises and Pitfalls,” *arXiv preprint arXiv:2303.16771*, 2023.

[10] “Meet Khanmigo: Khan Academy’s AI-powered teaching assistant & tutor.” Accessed: May 18, 2025. [Online]. Available: https://www.khanmigo.ai/

[11] A. K. Goel and L. Polepeddi, “Jill Watson: A Virtual Teaching Assistant for Online Education,” in *Learning Engineering for Online Education: Theoretical Contexts and Design-Based Examples*, C. Dede, J. Richards, and B. Saxberg, Eds., Routledge, 2018, pp. 120–143. [Online]. Available: http://hdl.handle.net/1853/59104

[12] “PaLM-E: An embodied multimodal language model,” 2023. [Online]. Available: https://research.google/blog/palm-e-an-embodied-multimodal-language-model/

[13] D. Team, “Duolingo Max uses OpenAI’s GPT-4 for new learning features,” 2025. [Online]. Available: https://blog.duolingo.com/duolingo-max/

[14] A. Cabello, “The Evolution of Language Models: a journey through time,” 2023. [Online]. Available: https://medium.com/@adria.cabello/the-evolution-of-language-models-a-journey-through-time-3179f72ae7eb

[15] “GPT-3 powers the next generation of apps.” https://openai.com/index/gpt-3-apps/

[16] "GPT-4." https://openai.com/index/gpt-4/

[17] S. AI, "BERT for Dummies: State-of-the-art Model from Google." https://medium.com/@skillcate/bert-for-dummies-state-of-the-art-model-from-google-42639953e769

[18] "GPT-Neo." https://www.eleuther.ai/artifacts/gpt-neo

[19] "Llama." https://www.llama.com/

[20] S. Tu *et al.*, "LittleMu: Deploying an Online Virtual Teaching Assistant via Heterogeneous Sources Integration and Chain of Teach Prompts," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23)*, ACM, 2023, pp. 1–7. doi: 10.1145/3583780.3615484.

[21] "XuetangX: Online Courses from Top Universities." https://www.xuetangx.com/

[22] "AMiner - AI赋能科技情报挖掘-学术搜索-论文检索-论文专利-文献追踪-学者画像." https://www.aminer.cn/

[23] K. Team, "Keras documentation: AlbertTextClassifier model." https://keras.io/keras_hub/api/models/albert/albert_text_classifier/

[24] G. Kortemeyer, "Ethel: A Virtual Teaching Assistant," *ArXiv*, 2024, [Online]. Available: https://arxiv.org/abs/2407.19452

[25] "AI Services | Microsoft Azure." https://azure.microsoft.com/en-us/products/ai-services

[26] K. Taneja, P. Maiti, S. Kakar, P. Guruprasad, S. Rao, and A. K. Goel, "Jill Watson: A Virtual Teaching Assistant powered by ChatGPT," *ArXiv*, 2024, [Online]. Available: https://arxiv.org/abs/2405.11070

[27] S. J. Sakib, B. K. Joy, Z. Rydha, M. Nuruzzaman, and A. A. Rasel, "Virtual teaching assistant for undergraduate students using natural language processing & deep learning," in *AIP Conference Proceedings*, 2024, p. 30042. doi: 10.1063/5.0192090.

[28] E. Davalos *et al.*, "LLMs as Educational Analysts: Transforming Multimodal Data Traces into Actionable Reading Assessment Reports," *ArXiv*, 2025, [Online]. Available: https://arxiv.org/abs/2503.02099

[29] Y. Shen *et al.*, "PMG: Personalized Multimodal Generation with Large Language Models," *arXiv preprint arXiv:2402.13075*, 2024, [Online]. Available: https://arxiv.org/abs/2404.08677

[30] D. Bratić, M. Šapina, D. Jurečić, and J. Žiljak Gršić, "Centralized Database Access: Transformer Framework and LLM/Chatbot Integration-Based Hybrid Model," *Applied System Innovation*, vol. 7, no. 1, p. 17, 2024, doi: 10.3390/asi7010017.

[31] L. Zheng *et al.*, "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," *arXiv preprint arXiv:2306.05685*, 2023, doi: 10.48550/arXiv.2306.05685.

[32] J. Odede and I. Frommholz, "JayBot: Aiding University Students and Admission with an LLM-based Chatbot," in *Proceedings of the 2024 Conference on Human Information Interaction and Retrieval (CHIIR '24)*, Association for Computing Machinery, 2024, pp. 391–395. doi: 10.1145/3627508.3638293.

[33] "The vector database to build knowledgeable AI | Pinecone." https://www.pinecone.io/

[34] S. Meyer, S. Singh, B. Tam, C. Ton, and A. Ren, "A Comparison of LLM Fine-tuning Methods & Evaluation Metrics with Travel Chatbot Use Case," *ArXiv*, 2024, [Online]. Available: https://arxiv.org/abs/2408.03562

[35] "App marketplace." [Online]. Available: https://marketplace.zoom.us/user/build

[36] "Fine-tuning Guide | UnSloth Documentation," 2025. [Online]. Available: https://docs.unsloth.ai/get-started/fine-tuning-guide

[37] Z. D. Forum, "How can I get cloud Recordings video using zoom Api," 2023. [Online]. Available: https://devforum.zoom.us/t/how-can-i-get-cloud-recordings-video-using-zoom-api/94828

[38] R. Rodrigues, "Downloads and organizes all cloud recordings from your Zoom Business account," 2025. [Online]. Available: https://github.com/ricardorodrigues-ca/zoom-recording-downloader

[39] "OpenAI/whisper-base.en · Hugging Face," 2023. [Online]. Available: https://huggingface.co/openai/whisper-base.en

[40] Pszemraj, "vid2cleantxt: Python API & command-line tool to easily transcribe speech-based video files into clean text," 2022. [Online]. Available: https://github.com/pszemraj/vid2cleantxt/tree/master

[41] "python-pptx — python-pptx 1.0.0 documentation." [Online]. Available: https://python-pptx.readthedocs.io/en/latest/

[42] Artifex, "PyMuPDF 1.25.5 documentation." [Online]. Available: https://pymupdf.readthedocs.io/en/latest/

[43] "unsloth/Llama-3.2-3B-Instruct · Hugging Face," 2024. [Online]. Available: https://huggingface.co/unsloth/Llama-3.2-3B-Instruct

[44] "Mistral-7b-Instruct-V0.3-Bnb-4bit." [Online]. Available: https://www.promptlayer.com/models/mistral-7b-instruct-v03-bnb-4bit

[45] C. H. U., “Fine-tuning LLAMA-3.2–3B-Instruct model using ‘Unsloth’ and ‘LORA,’” 2024. [Online]. Available: https://charanhu.medium.com/fine-tuning-llama-3-2-3b-instruct-model-using-unsloth-and-lora-adb9f9277917

[46] D. Research, “Finetuning mistral 7b using Unsloth,” 2024. [Online]. Available: https://dbrpl.medium.com/finetuning-mistral-7b-using-unsloth-2cf554159a03

[47] “Tutorial: How to finetune Llama-3 and use in Ollama | Unsloth Documentation.” [Online]. Available: https://docs.unsloth.ai/basics/tutorial-how-to-finetune-llama-3-and-use-in-ollama#id-5.-parameters-for-finetuning

[48] Papineni, K., Jr., Roukos, S., Ward, T., Zhu, W.-J., & IBM T. J. Watson Research Center. (2002). BLEU: a Method for Automatic Evaluation of Machine Translation. In *IBM T. J. Watson Research Center*. https://aclanthology.org/P02-1040.pdf

[49] Lin, C. (2004). *ROUGE: a package for automatic evaluation of summaries*. ACL Anthology. https://aclanthology.org/W04-1013/

[50] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2019, April 21). *BERTScore: Evaluating Text Generation with BERT*. arXiv.org. https://arxiv.org/abs/1904.09675