**Data set creation**

We initially attempted to use RAGAS to create a training dataset but encountered limitations due to the free tiers provided by AWS, OpenAI, Vertex, and Google, which were insufficient for our needs. As we lack the financial resources to upgrade, we tried downloading Hugging Face models locally to use with RAGAS, but found that RAGAS does not support them.

Next, we used the Hugging Face inference API, but quickly ran out of quota, and were prompted to purchase Hugging Face Pro. As a result, we decided to upload each PDF to Deep Seek Chat and ask it to generate a training data JSON file. However, we only managed to process one file and spent a significant amount of time troubleshooting why RAGAS was not performing correctly. Ultimately, we were only able to generate five data entries using RAGAS and the AWS free tier.

1)Tried to use amazon free tier

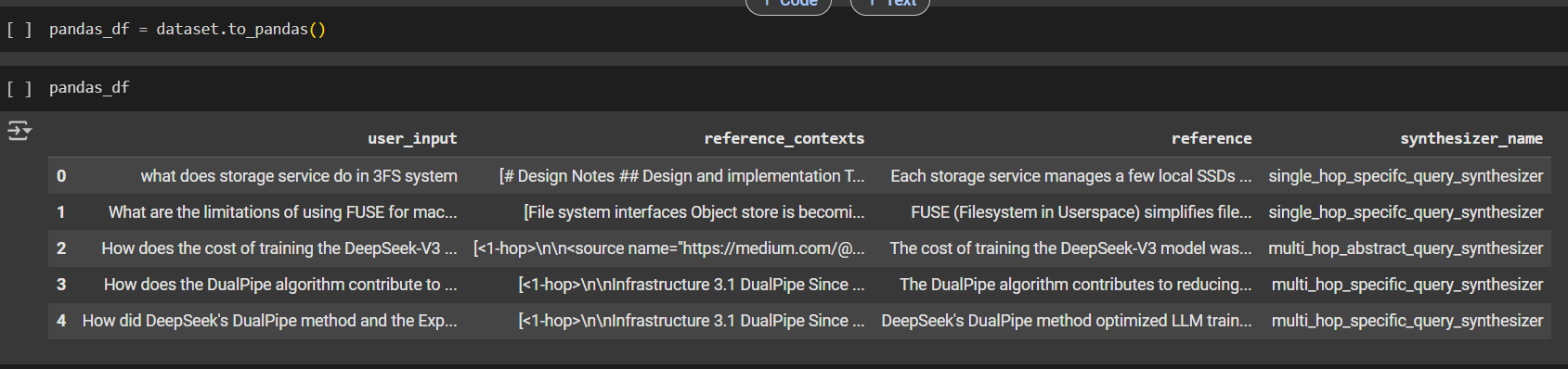
A computer code with text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Unfortunately, we were only able to create 5 rows for the dataset because we exceeded the free tier limits.



2) Try to download huggingface free models locally and try to generate dataset

We tried using a set of locally hosted Hugging Face models, but it didn't succeed because RAGAS implementation does not support these models. While we could use models from the Hugging Face inference API, we again ran out of the free quota and were prompted to purchase Hugging Face Pro.

A screen shot of a computer code

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

3) we uploaded the PDFs to DeepSeek and set up the dataset that way. We asked DeepSeek to generate the training data in a JSON format, but due to limitations, we were only able to process one file.

A screenshot of a computer

AI-generated content may be incorrect.

**Fine Tuning the model**

1. Setup and installation

* datasets: For loading and processing datasets.
* transformers: For working with pre-trained models and tokenizers.
* peft: For parameter-efficient fine-tuning (e.g., LoRA).
* trl: For supervised fine-tuning (SFT) of LLMs.
* bitsandbytes: For 4-bit quantization (QLoRA).
* accelerate: For distributed training.
* tensorboard: For logging training metrics.

### **Why Use LoRA Over Full Fine-Tuning?**

Full fine-tuning requires a significant amount of resources and time. Google Colab's free tier does not have the capabilities to handle such workloads. While full fine-tuning can increase accuracy compared to QLoRA, when considering the cost, time constraints, and resources available in a competition setting, QLoRA becomes a much better option. It offers a good balance between **accuracy**, **efficiency**, and **resource constraints**.

A screenshot of a computer

AI-generated content may be incorrect.

1. Model and Dataset Preparation

**A computer screen with white text

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

1. Fine-Tuning with LoRA and QLoRA

A screen shot of a computer program

AI-generated content may be incorrect.

**Justification of parameters used**

### **1. Max Sequence Length (max\_seq\_length = 40):**

* **Reason for Setting 40**: The choice of a max\_seq\_length of 40 is likely a trade-off between limiting input sequence length for faster processing and still being able to capture enough information for the task at hand.
* **Speed Consideration**: Shorter sequence lengths allow the model to process data faster. When sequences are limited to 40 tokens, the model spends less time processing each batch, speeding up training. This is particularly important when you're aiming for **faster training** without compromising too much on performance.
* **Memory Efficiency**: Shorter sequences use less GPU memory. This means you can fit more examples into each batch or train larger models on the same hardware. In situations where time and resources are limited, this helps you scale the training process.
* **Justification in Time-Crunch**: Since you didn’t have much time to build the model and wanted faster training, reducing sequence length can be an effective shortcut. While a longer sequence might capture more context, cutting down to 40 tokens will allow the model to process more batches in less time, allowing for more iterations over the data and faster overall training.

### **2. Packing (packing = True):**

* **Efficiency with Short Sequences**: Packing is a technique where smaller sequences (that are less than max\_seq\_length) are combined together into one batch to improve throughput. This reduces idle time for GPU resources and can increase training efficiency, especially when you have multiple small examples.
* **Faster Training with Packing**: Since you're limiting sequence length to 40, packing will allow multiple short sequences to be processed in parallel, improving the effective batch size and speeding up training.

### **3. Other Parameters:**

* **LoRA Parameters (lora\_r = 4, lora\_alpha = 16, lora\_dropout = 0.1)**:  
  + These parameters are designed to make the model more efficient by reducing the number of parameters while still achieving good performance. LoRA (Low-Rank Adaptation) allows the model to adapt without requiring full re-training, helping to train more efficiently.
  + **Reasoning**: These values are conservative, likely chosen to balance between model performance and training speed. The dropout value (0.1) helps prevent overfitting while still being small enough to allow the model to learn effectively.
* **Bitsandbytes Parameters (e.g., use\_4bit = True)**:  
  + **Reason for Setting**: Using 4-bit quantization reduces memory usage and speeds up training without significantly harming performance. This is particularly useful when time and memory constraints are tight.
  + **Impact on Training**: By enabling 4-bit precision for weight storage, memory requirements are drastically reduced, allowing you to train with larger models or more data within the same hardware constraints. This results in faster training iterations.
* **Training Parameters (e.g., per\_device\_train\_batch\_size = 4, learning\_rate = 2e-4, num\_train\_epochs = 10)**:  
  + **Reasoning for Lower Batch Size**: A batch size of 4 allows the model to fit within the GPU memory, especially with 4-bit quantization. This keeps the training feasible and reduces time spent on handling larger batches.
  + **Learning Rate**: The learning rate of 2e-4 is a typical starting point for fine-tuning large models. It’s chosen conservatively to avoid issues with convergence, especially with limited time for tuning.
  + **Epochs**: 10 epochs is a reasonable balance between training long enough to capture meaningful patterns and keeping training time manageable.
* **Other Choices (e.g., max\_grad\_norm = 0.3, warmup\_ratio = 0.03, save\_steps = 25)**:  
  + These settings are designed to help stabilize training and prevent overfitting while also improving efficiency. The use of gradient clipping (max\_grad\_norm) helps ensure the gradients don’t explode, and the learning rate warm-up helps avoid issues when the model is first starting to train.

### **4. Justification of the Approach for Faster Training:**

* **Faster Training**: The overall goal of these parameters is to **accelerate training** by reducing memory consumption, speeding up the processing of smaller sequences, and making training more efficient with lower precision (e.g., 4-bit).
* **Limited Resources**: You may not have had access to a large amount of time or computational power, so these parameters allow you to balance speed and efficiency. By using a smaller max\_seq\_length (40), packing, 4-bit quantization, and smaller batch sizes, you're able to train the model faster without needing vast amounts of memory.
* **Trade-Off Between Time and Accuracy**: While the choice of a small sequence length and other optimizations may lead to some loss in the ability to capture long-term dependencies, it's a reasonable trade-off when time is limited and you're aiming for faster iterations with the current resources.

A screenshot of a computer program

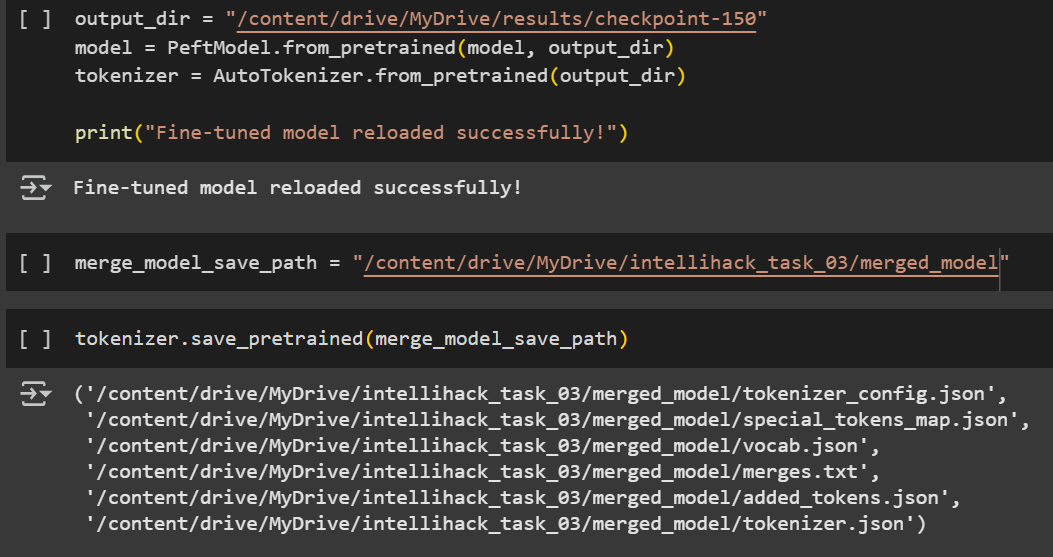
AI-generated content may be incorrect.

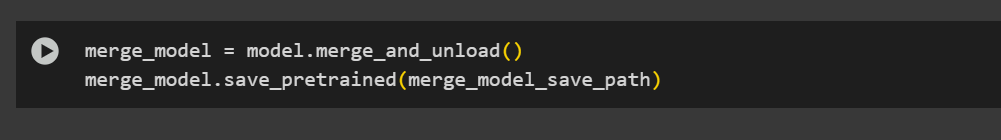
A screen shot of a computer

AI-generated content may be incorrect.

4.Model Merging and Quantization

When we use trainer.model.save\_pretrained(output\_dir), it only saves the adapter weights. As a result, the saved model cannot be directly converted to the .gguf format. To convert it to .gguf, we need to combine the base model with the adapter weights first.





5. Quantization

For quantization, we first used lama.cpp to quantize the model to float16. After that, we further quantized it to 4-bit. We followed this approach because we encountered errors when trying to directly convert the HF model to 4-bit.

6.Inference with Fine-Tuned Model

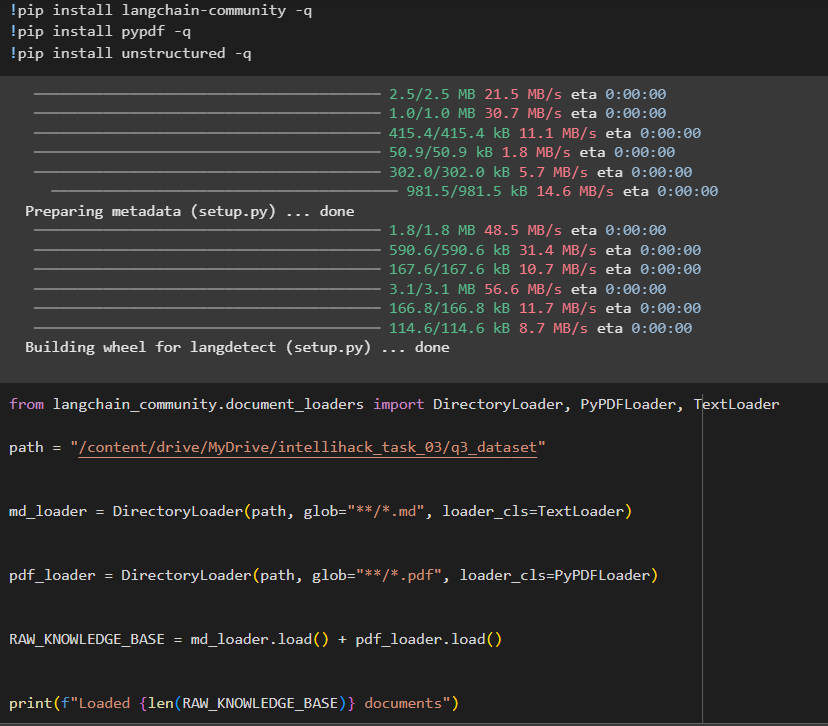
A screenshot of a computer

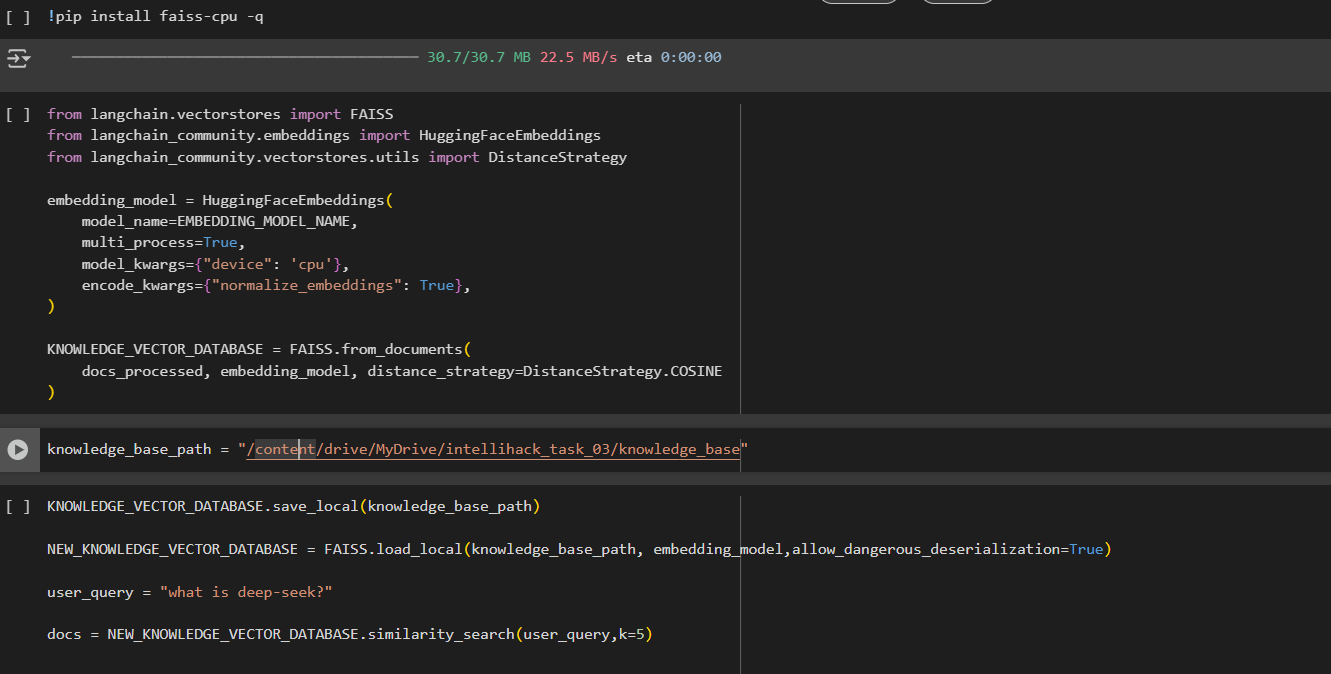
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

6. Retrieval-Augmented Generation (RAG) Pipeline





**Tools and Libraries Used**

* **Hugging Face Transformers**: For model loading and fine-tuning.
* **PEFT**: For LoRA-based fine-tuning.
* **TRL**: For supervised fine-tuning.
* **Bitsandbytes**: For 4-bit quantization.
* **Llama.cpp**: For model quantization and inference.
* **LangChain**: For document loading, splitting, and RAG pipeline.
* **FAISS**: For vector similarity search.

**How to increase accuracy**

**Use a Larger Dataset**:

* **Impact**: More data leads to better generalization. We used around 100 examples due to time constraints, but using a larger dataset can improve accuracy.

**Increase Max Sequence Length (max\_seq\_length)**:

* **Impact**: Longer sequences capture more context, improving accuracy, especially for tasks with long-range dependencies.
* **Suggestion**: Increase it from 40 to 128, 256, or higher.