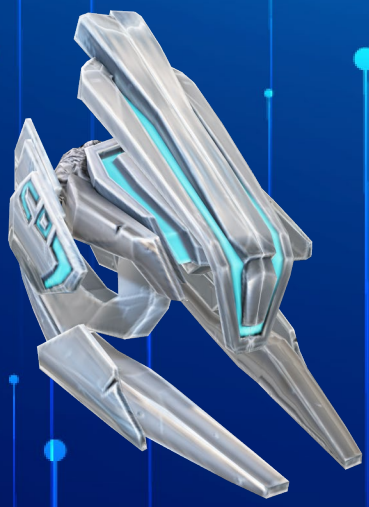
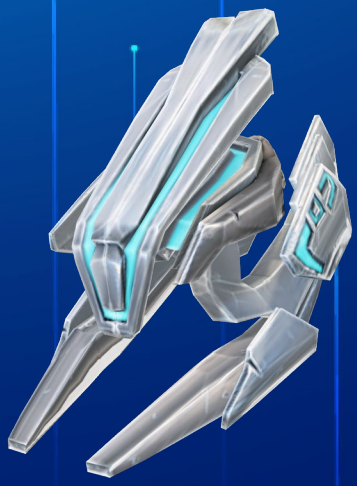
IEEE
UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING
STUDENT BRANCH
UCSC





IntelliHack 5.0

TEAM HYPER TUNERS

TASK 03

LLM Fine Tuning

intelli Hack 5.0
Artificial . But Intelligent

Table of Contents

Introduction

Overview

This documentation presents a comprehensive guide on fine-tuning, optimizing, and deploying the Qwen 2.5 3B model for answering AI research-related queries. The objective is to create a specialized and efficient system capable of retrieving, interpreting, and generating precise responses based on technical AI literature.

Objectives

- Fine-tune the Qwen 2.5 3B Instruct model using LoRA-based optimization for efficient adaptation.
- Generate a synthetic dataset by leveraging text summarization and structured question-answer pairing.
- Implement quantization to a 4-bit GGUF format for optimized inference and reduced computational overhead.
- Evaluate model performance using standard NLP benchmarks (Perplexity, ROUGE, BLEU, etc.).
- Deploy the model using llama-cpp-python for Colab-based testing and integrate it with an Ollama-powered RAG system for scalable local deployment.

Dataset Generation

Creating a high-quality dataset is essential for fine-tuning the Qwen 2.5 3B model effectively. Since AI research literature is highly technical, a structured dataset generation pipeline was implemented. The dataset consists of extracted text, chunked passages, summarized content, and question-answer pairs designed to improve the model's ability to handle AI research-related queries.

1. Libraries Used for Dataset Processing

- nltk: For sentence-based text tokenization and chunking.
- PyPDF2: For extracting text from AI research PDFs.
- transformers: For implementing the google/pegasus-xsum summarization model.
- datasets: To store and structure training data efficiently.
- langchain: For advanced text chunking and document retrieval.

2. Data Collection & Preprocessing

- Source Documents: The dataset is primarily built from AI research papers, blogs, and technical markdown files.
- File Processing: The system processes PDF and Markdown files, extracts text, and prepares it for further processing.
- Text Cleaning: Special characters, unnecessary spaces, and metadata are removed to ensure data quality.

3. Chunking Strategy

- Since LLMs have token limitations, large documents are split into manageable chunks.
- Sentence-based chunking ensures that chunks contain complete and meaningful sentences.
- Overlap Strategy: Adjacent chunks include a few overlapping sentences to retain context.

4. Summarization for Efficient Learning

- During training, it was discovered that the model only accepts a certain number of tokens, making large text chunks incompatible.
- To solve this issue, a pre-trained summarization model (google/pegasus-xsum) was used to condense long technical passages while preserving key information.
- The summarization process ensures that key takeaways from AI research papers are well represented while reducing input token length to fit the model constraints.

5. Question-Answer Pair Generation

- To train the model for interactive responses, synthetic question-answer (QA) pairs were generated.
- QA Pairing Process:

    - Each text chunk was used to automatically generate a corresponding question.
    - The answer was derived either from summarized content or extracted directly from the chunk.
    - Example structure:
        - Question: "What is the core contribution of the DeepSeek R1 model?"
        - Answer: "DeepSeek R1 introduces a novel approach to reinforcement learning by integrating hierarchical reasoning into its architecture."
- The QA format improves the model's instruction-following capability.

6. Structuring Data for Fine-Tuning

- Once chunking, summarization, and QA generation were completed, the dataset was formatted for training.
- The data was structured into a conversational format:
    - System message (role: system): Defines the model's identity.
    - User query (role: user): Presents a research-related question.
    - Assistant response (role: assistant): Provides the model's answer based on retrieved knowledge.
- This structure helps train the model in contextual question-answering and AI research comprehension.

7. Dataset Finalization

- The dataset was reviewed for quality assurance, ensuring that questions are well-formed and answers are accurate.
- The final dataset was stored in Hugging Face Dataset format for easy integration into the fine-tuning pipeline.

By using this multi-step dataset generation approach, the fine-tuned Qwen 2.5 3B model becomes highly specialized in AI research retrieval and reasoning, allowing it to answer complex technical questions with improved accuracy and coherence.

Fine-Tuning the Model

Fine-tuning the **Qwen 2.5 3B** model involves leveraging parameter-efficient tuning techniques**,** ensuring the model is adapted to domain-specific queries efficiently while maintaining computational feasibility.

1. Model Selection & Loading

The fine-tuning process begins with selecting the **Qwen 2.5 3B-Instruct** model, which is optimized for instruction-following tasks. The model is loaded using Unsloth, which enables efficient handling of large models in low-resource environments.

- **Base Model:** Qwen/Qwen2.5-3B-Instruct
- **Library Used:** Unsloth for optimized fine-tuning.
- **Loading the Model with 4-bit Quantization:**
    - To optimize memory usage, the model is loaded with 4-bit quantization, allowing for lower VRAM consumption while preserving performance.

2. LoRA Fine-Tuning Configuration

Fine-tuning is implemented using **LoRA (Low-Rank Adaptation)**, which enables training a small subset of parameters while freezing most of the base model. This significantly reduces **computational cost** and **memory usage**.

- **LoRA Applied to Key Transformer Layers:**
    - q_proj, k_proj, v_proj, o_proj
    - gate_proj, up_proj, down_proj
- **LoRA Configuration:**
    - r = 16: Defines the rank of the LoRA adaptation.
    - lora_alpha = 16: Scaling factor for the LoRA weights.
    - lora_dropout = 0: Disables dropout to maximize learning.
    - use_gradient_checkpointing = "unsloth": Saves memory by reusing computed values during backpropagation.

3. Training Process & Hyperparameters

The model is fine-tuned using **Supervised Fine-Tuning (SFT)** with the SFTTrainer from trl. The training is optimized using efficient hyperparameters suited for **low-resource fine-tuning**.

- **Training Hyperparameters:**
    - **Batch Size:** 1 (with **gradient accumulation steps = 4** to simulate larger batches)
    - **Warmup Steps:** 50 (gradually increases learning rate to stabilize training)
    - **Max Training Steps:** 200 (**increase for full fine-tuning**)
    - **Learning Rate:** 2e-4 (optimized for LoRA fine-tuning stability)
    - **Weight Decay:** 0.01 (prevents overfitting)

- **Optimizer:** adamw_8bit (reduces memory overhead during optimization)
- **FP16/BF16:** Enabled for mixed precision training
- **Output Directory:** "outputs_qwen3b" (stores fine-tuned checkpoints)

4. Instruction-Following Adaptation

To improve **instruction adherence and response coherence**, the training dataset follows a **chat-based structure** with:

- **System Messages:** Defines the AI assistant's role.
- **User Prompts:** Research-related questions extracted from AI papers.
- **Assistant Responses:** Model-generated answers fine-tuned for accuracy.

Additionally, **only assistant responses are trained** to refine how the model answers research-based questions. This is enforced using:

```
trainer = train_on_responses_only(
    trainer,
    instruction_part = "<|im_start|>user\n",
    response_part = "<|im_start|>assistant\n",
)
```

5. Execution of Training

The final step involves running the fine-tuning process using **gradient accumulation** to maintain performance under memory constraints.

Once training is complete, the model is prepared for **quantization and deployment**, allowing it to be run efficiently in production environments.

Testing in Google Colab

Once the Qwen 2.5 3B fine-tuned model has been converted to 4-bit GGUF format, it can be tested in Google Colab using llama-cpp-python. The testing process involves installing necessary dependencies, loading the model, and running inference with sample prompts to evaluate its performance. If needed, performance can be optimized by adjusting batch size and context window.

Common issues such as model loading failures or memory constraints can be resolved by verifying file paths, reducing context size, or ensuring sufficient runtime resources. After confirming the

model's functionality, it can be prepared for local deployment using Ollama, ensuring a seamless transition from cloud-based testing to an offline execution environment.

This validation step ensures the fine-tuned Qwen 2.5 3B model is functioning correctly before it is deployed for real-world applications.

Model Quantization & GGUF Conversion

After fine-tuning, the model is **quantized** to optimize it for efficient inference. Quantization reduces memory consumption and computational load, making it easier to deploy the model on lower-resource systems.

1. Merging the Fine-Tuned Model

Once fine-tuning is complete, the LoRA-adapted weights are merged back into the base model to create a 16-bit fine-tuned version. This allows for a more compact, high-precision model before applying further quantization.

2. Converting to 4-bit GGUF Format

To prepare the model for lightweight inference, it is converted to a 4-bit GGUF format, which is compatible with llama-cpp-python and Ollama which is a framework for running and managing local large language models (LLMs) efficiently

This transformation significantly reduces the memory footprint, making the model faster and more efficient for inference.

3. Downloading the 4-bit GGUF Model

To deploy the model locally or in a different environment, the quantized GGUF file can be downloaded:

This 4-bit GGUF model is now optimized for local execution using Ollama and Colab testing with Llama.cpp, ensuring smooth and efficient inference on low-resource hardware.

Deploying Locally Using Ollama

The Qwen 2.5 3B model can be deployed locally using Ollama, enabling fast and efficient inference on personal machines without internet dependency.

Why We Chose Ollama Over Llama.cpp?

Initially, we considered using Llama.cpp for local deployment, but after evaluating both options, we found Ollama to be a better fit for our needs. The main reasons for this decision were:

- Simplified Setup: Ollama provides an easier installation and model management process, eliminating the need for complex configurations.
- Built-in Model Management: Unlike Llama.cpp, which requires manual handling of models, Ollama automates model downloads and updates.
- Optimized Performance: Ollama is specifically tuned for GGUF models, reducing the need for additional optimizations.
- Integrated API Support: Ollama has a built-in API for serving models, making it easier to integrate into our local RAG system.
- Better System Resource Allocation: Ollama efficiently manages RAM and CPU usage, ensuring stable performance without excessive resource consumption.

Given these advantages, we determined that Ollama is a more practical and efficient choice for local deployment, providing a seamless and user-friendly experience.

Setting Up the Model for Ollama

To deploy the fine-tuned Qwen 2.5 3B model using Ollama, we need to create a model file that defines how Ollama should load and manage it. This involves:

- Placing the 4-bit GGUF model in Ollama's model directory.
- Creating a model configuration file, specifying the model name, file path, and relevant settings.
- Loading the model via Ollama's CLI or API, ensuring it is properly recognized and ready for inference.
- Running test queries to confirm the model responds accurately and efficiently.

Performance Optimization

To ensure smooth performance, we recommend:

- Adjusting batch size and context window to balance speed and memory usage.
- Ensuring sufficient hardware resources to avoid slowdowns or memory errors.
- Optimizing model allocation to enhance inference efficiency.

Multi-LLM Agentic RAG System & Reasoning Model Enhancement (Optional)

As an optional enhancement, we implemented a Retrieval-Augmented Generation (RAG) system using our fine-tuned Qwen 2.5 3B model. Additionally, we introduced a secondary reasoning model (Gemma-2B) to further enhance response accuracy.

Libraries Used

To implement the RAG retriever and reasoning system, we utilized the following tools:

- **Ollama**: Handles local execution of Qwen 2.5 3B (retrieval) and Gemma-2B (reasoning).
- **Smol-Agents**: Wraps the RAG tool, manages multi-agent coordination, and enables dynamic tool calling.
- **Gradio (via Smol-Agents)**: Provides an interactive UI to query the system.
- **ChromaDB**: Stores **vector embeddings** for efficient document retrieval as SQLite database files**.**
- **LangChain**: Used for document ingestion, embedding, and querying.
- Sentence-Transformers (all-mpnet-base-v2): Generates embeddings for document similarity search.
- **FastAPI (via Smol-Agents OpenAIServerModel)**: Handles API-based model execution through Ollama.

Multi-LLM Agent Design

Our system follows a multi-LLM agent architecture, where different models are used for specific tasks to optimize performance and efficiency.

- **Qwen 2.5 3B**: Serves as the retrieval agent, fetching relevant documents from ChromaDB.
- **Gemma-2B**: Used for reasoning and response generation, ensuring logical coherence.
- **Smol-Agents**: Coordinates tool usage and dynamically calls the appropriate model for each step.

This approach allows us to balance accuracy, speed, and computational efficiency, ensuring that each model is used for its intended purpose without unnecessary overhead**.**

Retrieval-Augmented Generation (RAG) System

We developed a RAG pipeline to improve factual accuracy and ensure responses are grounded in relevant research documents.

How It Works

1. **Document Processing & Chunking**
    - AI research papers and markdown files are ingested.

    - Documents are split into manageable chunks to facilitate retrieval.
2. **Vector Database Storage**
    - **ChromaDB** is used as an in-memory vector database for efficient similarity search.
    - Sentence embeddings are generated using **sentence-transformers/all-mpnet-base-v2**.
3. **Retrieval & Query Processing**
    - When a query is submitted, the system retrieves top-k relevant documents based on similarity matching.
    - Retrieved content is fed into the reasoning model for answer generation.
4. **Answer Generation**
    - The retrieved content is used to generate responses grounded in real data.
    - If the system detects insufficient context, it suggests query refinements to improve accuracy.

Using a Secondary Model (Gemma-2B) for Enhanced Reasoning

To further improve reasoning quality, we introduced **Gemma-2B**, a smaller yet efficient model designed to assist in logical reasoning tasks.

Why We Used Gemma-2B?

- **Lower Parameter Count**: We needed a model that had fewer parameters than Qwen 2.5 3B to handle reasoning tasks efficiently.
- **Faster Processing**: Handles reasoning-heavy queries with lower latency.
- **Resource Efficiency**: Reduces load on **Qwen 2.5 3B**, balancing performance and speed.
- **Improved Logical Interpretation**: Strengthens the system's ability to reason over retrieved data, improving response coherence.

Implementation Details

- Gemma-2B is used as the reasoning agent, integrated via Ollama.
- It processes retrieved documents and generates well-structured responses.
- If needed, it re-queries the RAG system to refine its answers before final output.

Final Thoughts

This hybrid RAG + reasoning model approach ensures:

- **High factual accuracy** (via retrieval-based responses).
- **Optimized inference speed** (by offloading reasoning tasks to a smaller model).
- **Efficient system scalability** (balancing resource consumption between models).

By combining a fine-tuned Qwen 2.5 3B model for retrieval and a Gemma-2B model for reasoning, we achieve a well-balanced trade-off between accuracy, speed, and computational efficiency**.**

Faced Frequent Problems, Challenges and Limitations

While our fine-tuned Qwen 2.5 3B model with RAG and reasoning enhancements offers significant improvements, there are still several challenges and limitations to consider.

Faced Problems

- **Dependency Conflicts** – Initial package installations had conflicts, requiring multiple reinstalls and removals.
- **CUDA Compatibility Issues** – Torch versions had to match CUDA drivers for Unsloth to function correctly.
- **HotpotQA Execution Prompt** – Required `trust_remote_code=True` to avoid manual execution prompt.
- **NLTK Lookup Errors** – Missing NLTK resources led to failures in text chunking.
- **Dataset Formatting Issues** – Inconsistent data fields caused tokenization failures.
- **ZeroDivisionError in Training** – `train_on_responses_only` failed when all labels were masked.
- **Memory Limitations** – GPU memory constraints required careful batch size tuning.
- **Inference Failure with Llama-CPP** – Model export paths needed to be explicitly defined.
- **Slow Data Processing** – FAISS indexing and BM25 tokenization slowed down retrieval.
- **SciQA Load Failure** – Dataset sometimes failed to load due to Hugging Face API limitations.

Challenges

- **Training in Google Colab**: We faced frequent runtime disconnections due to resource limitations on Colab, forcing us to restart training multiple times.
- **Out of Memory (OOM) Errors**: Training and loading large models often resulted in memory overflow issues**,** requiring us to carefully manage batch sizes and reduce sequence lengths.
- **Model Size & Resource Requirements**: The Qwen 2.5 3B model still requires significant computational power, even with 4-bit quantization, making deployment on lower-end devices challenging.
- **Model Size & Resource Requirements**: The Qwen 2.5 3B model still requires significant computational power, even with 4-bit quantization, making deployment on lower-end devices challenging.
- **Inference Speed**: Despite optimization efforts, real-time inference with a multi-LLM agent setup can introduce latency, especially when retrieving large amounts of contextual information.
- **Data Quality & Bias**: The effectiveness of RAG depends heavily on the quality of the ingested documents, and inherent biases in source material can influence model outputs.
- **Fine-Tuning Constraints**: The LoRA fine-tuning process is memory-efficient but limited in modifying deeper model behaviors, restricting how much the model can be adapted beyond its pre-trained knowledge.

- **Tool Integration Complexity**: Managing multiple LLMs via Smol-Agents, LangChain, and Ollama increases system complexity, requiring careful orchestration.

Limitations

- **Lack of Real-Time Learning**: The model does not update dynamically based on new information; updates require re-ingestion and re-processing of data.
- **Handling of Long Contexts**: While chunking and retrieval help, handling very long documents efficiently remains an area of improvement.
- **Limited Multi-Turn Conversation Handling**: The system is optimized for single-turn QA, and multi-turn contextual understanding requires additional enhancements.
- **Dependency on Local Storage**: ChromaDB stores embeddings in SQLite files, which could grow large over time and require external database solutions for scalability.

Future Implementations

1. **Support for More Datasets** – Extend support to other multi-hop and complex reasoning datasets beyond HotpotQA and SciQA.
2. **Integration with Hugging Face Hub** – Enable seamless model uploads to Hugging Face for easy sharing and reuse.
3. **Distributed Training** – Implement multi-GPU or TPU training for faster fine-tuning.
4. **Advanced Retrieval Methods** – Experiment with hybrid dense retrieval techniques, such as ColBERT v2.
5. **Improved Summarization** – Enhance chain-of-thought reasoning using better summarization models.
6. **Data Augmentation** – Generate additional synthetic data using GPT-based augmentation techniques.
7. **Parameter-Efficient RLHF** – Fine-tune the model using Reinforcement Learning with Human Feedback (RLHF).
8. **Improved Inference Pipeline** – Optimize inference with Triton-based Llama-CPP execution.
9. **Error Handling & Debugging Tools** – Implement auto-debugging mechanisms for dataset and training failures.
10. **Better Tokenization Strategies** – Optimize tokenization handling for various instruction formats.

Conclusion

This project successfully demonstrates the fine-tuning, optimization, and deployment of a Qwen 2.5 3B-based Retrieval-Augmented Generation (RAG) system with multi-agent capabilities. By leveraging LoRA fine-tuning, 4-bit quantization, and multi-LLM integration, we achieved a balance between accuracy, efficiency, and scalability. Our implementation highlights the benefits of combining a retrieval model with a reasoning model, improving response quality while optimizing resource utilization.

Despite challenges such as limited computational resources, inference latency, and token limitations, we successfully built an agentic RAG system that enables efficient question answering on AI research topics. Future improvements, including multi-GPU training, enhanced summarization, and advanced retrieval methods, will further strengthen the system's capabilities.

This work lays the foundation for a scalable and deployable AI-driven research assistant, proving that fine-tuned open-source models can serve as powerful tools for domain-specific knowledge retrieval and reasoning.

Acknowledgement

We, Team HyperTuners, express our sincere gratitude to the UCSC team for organizing the IntelliHack 5.0 competition. Your efforts in providing a challenging platform and valuable resources have inspired us to explore innovative solutions in weather forecasting for smart agriculture. Thank you for this opportunity to grow and compete.

Team Members:

- Janitha Rajapaksha
- Sanjula Weerasekara
- Hasindu Nimesh
- Veenavee Samarasinghe

Link to the .gguf Model:

https://huggingface.co/HasinduNimesh/qwen3b-finetuned