Final Report (Task 3): Qwen2.5-3B-Instruct Fine-tuning

The 4-bit quantized model

[https://huggingface.co/IsaraLi/TeamSemiColon_unsloth.Q4_K_M.gguf/tree/main]

Complete inference script

[https://drive.google.com/file/d/1H5Y0q9ygZyXxDB76rn2oyDDANJdQuDpm/view?usp=sharing]

Documented training process with code

[Separate Document Link :-
https://drive.google.com/file/d/1Kn-vva9CAPtwCaLC2yR8bDY2SG4YKN1j/view?usp=sharing]

Comprehensive Training Process for Qwen2.5-3B-Instruct Fine-tuning

1. Introduction

This document presents an in-depth report on fine-tuning the **Qwen2.5-3B-Instruct** model to enhance its ability to answer AI research-related queries. The fine-tuning process utilized **Unsloth** for efficient training, incorporating **QLoRA (4-bit quantization)** for optimal memory management and practical deployment. The goal was to improve the model's ability to generate accurate, research-driven responses by leveraging fine-tuned datasets derived from AI research papers, blogs, and technical documents.

2. Model Selection

2.1 Choice of Model

Selected Model: Qwen/Qwen2.5-3B-Instruct

- This model was chosen due to its **instruction-following capability**, which is crucial for answering structured AI research questions.
- The **instruct variant** was preferred over the base model to leverage its pre-trained instruction-following behavior.
- The model has a **3B parameter scale**, making it a feasible choice for fine-tuning on a single or multi-GPU setup.

2.2 Justification for Quantization (QLoRA)

- **Memory Efficiency:** Enables training on consumer GPUs with limited VRAM.
- **Minimal Performance Degradation:** Maintains the effectiveness of full precision models while significantly reducing memory overhead.
- **Allows for Gradient Updates:** Unlike static quantization, QLoRA allows fine-tuning while keeping computational demands low.
- **Faster Inference Speeds:** Post-training inference with quantization allows for more efficient deployments.

3. Dataset Preparation

3.1 Data Sources

The dataset was derived from multiple sources to ensure comprehensive coverage of AI research topics:

- **Academic Research Papers**: Extracted from PDFs using `PyMuPDF` (fitz) and OCR fallback via `pytesseract` for scanned documents.
- **Technical AI Blogs & Documentation**: Processed from Markdown (`md`) files using `markdown` and `re` libraries to extract structured content.
- **Synthetic Q&A Pairs**: Automatically generated from extracted texts using prompt engineering and NLP techniques(using models like `valhalla/t5-base-qg-hl`)

3.2 Data Preprocessing

The extracted data underwent multiple preprocessing steps to ensure compatibility with fine-tuning:

3.2.1 Text Extraction & Cleaning

- **PDF Parsing**: Utilized `fitz` (PyMuPDF) to extract text from research paper PDFs.
- **OCR for Scanned PDFs**: Applied `pytesseract` OCR on images extracted from PDFs to retrieve non-selectable text.
- **Markdown Processing**: Stripped HTML tags and reformatted Markdown files into clean text.
- **Regex-Based Cleaning**: Removed unnecessary symbols, footnotes, and HTML artifacts to enhance readability.

3.2.2 Tokenization & Formatting

- Applied **sentence segmentation** using `nltk` to break content into logically structured parts.
- Implemented **text normalization** to remove inconsistencies in whitespace, special characters, and casing.
- Converted extracted data into structured **instruction-response format** using `formatting_prompts_func`.
- Parallelized batch processing with `num_proc=2` for efficiency.

Python

```
from datasets import load_dataset

dataset = load_dataset("json", data_files="output.json")
dataset = dataset["train"].train_test_split(test_size=0.2, seed=3407)
train_dataset = dataset["train"]
eval_dataset = dataset["test"]

train_dataset = train_dataset.map(formatting_prompts_func, batched=True, num_proc=2)
eval_dataset = eval_dataset.map(formatting_prompts_func, batched=True, num_proc=2)
```

3.3 Data Augmentation Techniques

To improve model generalization and response quality, various augmentation techniques were applied:

3.3.1 Paraphrasing

- Used NLP-based transformations to **reword AI research questions** while preserving semantic meaning.
- Ensured diverse wording for similar topics to enhance model robustness.

3.3.2 Synthetic Q&A Generation

- Applied prompt-based synthetic generation to create **multiple variations of research-based questions and answers**.
- Used `transformers` models to generate **alternative answers** to the same questions for variety.

3.3.3 Noise Injection

- Introduced **intentional typos and varied punctuation** to improve model robustness against imperfect user inputs.
- Included slight grammatical errors to reflect real-world user queries.

3.3.4 Entity Replacement Augmentation

- Replaced **specific AI research terms** with analogous terms to test generalization.
- Example: Swapped "Transformer models" with "Sequence models" in specific training examples.

3.3.5 Adversarial Prompting

- Designed **edge-case prompts** where AI research topics were deliberately phrased ambiguously to assess reasoning capabilities.
- Example: "How does an AI model generate text?" vs. "Can an AI system generate text like humans?"

These preprocessing and augmentation techniques significantly enhanced the dataset's diversity, ensuring the fine-tuned model could handle a broad range of AI research-related questions.

4. Training Configuration

4.1 Hyperparameters & Justifications

Hyperparameter	Value	Justification
max_seq_length	2048	Ensures long-context retention for research papers.
dtype	None	Uses auto-detection of best data type.
load_in_4bit	True	Efficient memory usage with QLoRA.
batch_size	2	Optimal for GPU memory constraints.
learning_rate	2e-5	Standard for instruction-tuned models.
num_epochs	3	Balances overfitting risks and model improvement.
optimizer	AdamW	Standard optimizer for fine-tuning transformers.
gradient_accumulation	16	Helps in reducing memory usage during training.
Evaluation Strategy	Epoch-based	Ensures regular assessment of model progress.

4.2 Training Strategy

- **QLoRA (Quantized Low-Rank Adaptation)** was applied to reduce VRAM usage while maintaining training effectiveness.
- **Unsloth** was used for accelerated fine-tuning by patching model internals for speed optimization.
- **Trainer:** `SFTTrainer` from `trl` was used for supervised fine-tuning.

Model Loading & Training Code

Python

```
from unsloth import FastLanguageModel

from trl import SFTTrainer

from transformers import TrainingArguments


# Load Model with QLoRA

model, tokenizer = FastLanguageModel.from_pretrained(

    model_name="Qwen/Qwen2.5-3B-Instruct",
```

```
    max_seq_length=2048,
    dtype=None,
    load_in_4bit=True,
)
```

Python

```
# Define Training Arguments
training_args = TrainingArguments(
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    num_train_epochs=3,
    learning_rate=2e-5,
    gradient_accumulation_steps=16,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_dir="./logs",
    report_to="none"
)

trainer = SFTTrainer(
    model=model,
```

```
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    args=training_args,
    tokenizer=tokenizer,
)

trainer.train()
```

5. Evaluation & Results

5.1 Validation Metrics & Performance Analysis

Step	Training Loss	Validation Loss
20	2.680300	2.013773
40	1.239000	1.190110
60	1.007400	1.062270
80	0.937300	1.013381
100	0.851800	0.989729
120	0.800700	0.981624
140	0.773800	0.981477
160	0.769300	0.981282

Observations:

- **Steady Decrease in Training Loss:** Indicates successful fine-tuning.
- **Validation Loss Plateau:** Suggests a stable model performance beyond 100 steps.
- **Potential Further Optimization:** Loss plateauing could suggest hyperparameter tuning opportunities.

6. Optimization Techniques

- **Efficient Fine-Tuning with QLoRA** to reduce memory usage.
- **Gradient Accumulation** to balance batch size limitations.
- **Chat Template Adjustments** to better fit instruction-based responses.
- **Paraphrased Data Augmentation** to enhance response diversity.
- **Early Stopping Considerations** to prevent unnecessary training beyond convergence.

7. Conclusion

This fine-tuning process successfully optimized the **Qwen2.5-3B-Instruct** model for AI research Q&A. The model shows improved reasoning and answer accuracy based on the provided technical documents. The training methodology, dataset preprocessing, and optimization techniques ensured an efficient, high-performing adaptation of the base model for research-focused tasks.

Technical Report

[Seperate Link For Technical Report:-
https://drive.google.com/file/d/1I2jq4IsPSosQ729SJ9yhGXtk1YLeaBjX/view?usp=sharing

Technical Report: Fine-Tuning and Embedding-Based Model Pipeline

1. Introduction

This report provides an in-depth analysis of the methodologies and decision-making processes involved in the development of a **dataset creation, embedding generation, and fine-tuning pipeline** for NLP applications. The primary objectives of this pipeline include:

1. **Dataset Creation**: Extracting, processing, and structuring data from unstructured sources.
2. **Vector Embedding Generation**: Converting text data into high-dimensional vector representations for efficient similarity retrieval.
3. **Fine-tuning & Inference**: Customizing a base model to enhance its ability to generate responses and performing evaluations to verify performance.

Each phase was designed with scalability and efficiency in mind, ensuring that the pipeline can be easily extended or modified as needed.

---

2. Dataset Creation

2.1 Data Sources and Extraction

Decision and Rationale: Why PDFs and Markdown?

PDFs were chosen because they are a common document format that contains structured textual data. However, not all PDFs allow text extraction directly (e.g., scanned documents), which necessitated the use of **OCR (Optical Character Recognition)**. Markdown files were included because they are widely used in documentation and provide structured yet lightweight text.

Why PyMuPDF over Other Libraries?

- **PyMuPDF (fitz)** was selected over libraries like PDFMiner due to its superior speed and better support for text layout preservation.
- **OCR (Tesseract + pdf2image)** was used as a fallback for scanned PDFs, since PyMuPDF cannot extract text from images.
- **Regular expressions** were used instead of pre-built text cleaners because they provide greater control over unwanted elements in extracted text.

Implementation Details:

```python
import fitz
import pytesseract
from pdf2image import convert_from_path
import markdown
import re
import glob

# Extract text from PDFs
def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    text = ""
    for page in doc:
        page_text = page.get_text("text")
        if page_text.strip():
            text += page_text + "\n"
        else:
            images = convert_from_path(pdf_path)
            for image in images:
                text += pytesseract.image_to_string(image) + "\n"
    return text.strip()
```

2.2 Text Chunking and Preprocessing

Decision and Rationale: Why Sentence-Based Chunking?

Transformers have a maximum input length constraint (e.g., 512 tokens for BERT, 2048 for Qwen). To optimize text representation while maintaining contextual coherence, **sentence-based chunking** was chosen over fixed-length chunking.

Why NLTK over SpaCy?

- **NLTK's sentence tokenizer** was selected because it is lightweight and does not require an external model.
- **SpaCy** was considered but omitted due to its dependency on large language models for segmentation, which was unnecessary for this case.

Implementation Details:

```python
from nltk.tokenize import sent_tokenize

def split_documents(documents, max_length=256):
    chunks = []
    for doc in documents:
        sentences = sent_tokenize(doc)
        chunk = ""
        for sentence in sentences:
            if len(chunk) + len(sentence) <= max_length:
                chunk += sentence + " "
            else:
                chunks.append(chunk.strip())
                chunk = sentence + " "
        if chunk:
            chunks.append(chunk.strip())
    return chunks
```

2.3 Question-Answer Pair Generation

Decision and Rationale: Why Automated QA Generation?

Instead of manually annotating data, an automated approach was chosen using:

1. **T5-based Question Generator (`valhalla/t5-base-qg-hl`)** – chosen over GPT-3.5 due to its **efficiency in smaller-scale applications**.

2. **DistilBERT (`distilbert-base-cased-distilled-squad`)** – selected over full BERT models for its balance between accuracy and computational efficiency.

Why Not Use GPT-4 for QA Pairing?

While GPT-4 can generate high-quality question-answer pairs, its computational cost and API latency make it unsuitable for batch processing at scale.

Implementation Details:

```python
from transformers import pipeline
question_generator = pipeline("text2text-generation",
model="valhalla/t5-base-qg-hl")
qa_extractor = pipeline("question-answering",
model="distilbert-base-cased-distilled-squad")

qa_pairs = []
for context in chunks:
    question = question_generator(f"Generate questions from:
{context}")[0]["generated_text"]
    answer = qa_extractor(question=question, context=context)
    qa_pairs.append({"context": context, "question": question,
"answer": answer["answer"]})
```

---

3. Vector Embedding Creation

Decision and Rationale: Why Use `all-mpnet-base-v2`?

This embedding model was chosen due to:

- **Better semantic similarity performance** compared to `sentence-BERT`.
- **Lower computational cost** than models like `text-embedding-ada-002` (used in OpenAI APIs).

Why Not Use FAISS Indexing?

While FAISS provides efficient nearest neighbor search, this pipeline focuses on embedding generation. FAISS integration is planned for downstream retrieval tasks.

Implementation Details:

```python
from sentence_transformers import SentenceTransformer
import numpy as np

embedding_model = SentenceTransformer("all-mpnet-base-v2")
embeddings = embedding_model.encode([chunk.page_content for chunk
in document_chunks])

# Normalize and store
embeddings_array = np.array(embeddings).astype('float32')
embeddings_array /= np.linalg.norm(embeddings_array, axis=1,
keepdims=True)
np.save("vector_database.npy", embeddings_array)
```

---

4. Model Fine-Tuning & Inference

4.1 Model Selection & Configuration

Decision and Rationale: Why Qwen2.5-3B Instead of Llama 2?

- **Qwen2.5-3B** was chosen for its superior performance in instruction-following tasks.
- **Llama 2** was considered but required more computational resources for fine-tuning.

Why LoRA (Low-Rank Adaptation)?

- **LoRA reduces the number of trainable parameters** while achieving similar fine-tuning quality.
- **Full fine-tuning would require significant VRAM**, making it infeasible for most setups.

Implementation Details:

```python
from unsloth import FastLanguageModel
model, tokenizer = FastLanguageModel.from_pretrained(
    "Qwen/Qwen2.5-3B-Instruct", max_seq_length=2048,
load_in_4bit=True
)
```

---

5. Conclusion

This report has documented the pipeline from dataset creation to fine-tuning a model using LoRA. The **key decision points** included:

- **Using PDFs & Markdown over raw text files** due to their structured format.
- **Using PyMuPDF over PDFMiner** for faster and more accurate extraction.
- **Using NLTK over SpaCy** for efficient sentence segmentation.
- **Using T5 & DistilBERT for QA generation** instead of GPT-4 for cost-efficiency.
- **Using `all-mpnet-base-v2` for embeddings** due to its superior performance over SBERT.
- **Using LoRA for fine-tuning** instead of full model training to reduce VRAM usage.

This pipeline is designed to be **modular, scalable, and optimized** for future NLP applications.

Evaluation results using G-Eval framework

[Separate PDF For The Evaluation Report:-
https://drive.google.com/file/d/1dpYX2UtU2nqgnYR22YLTCqaKujE-_w2Z/view?usp=sharing]

Evaluation Report: LLM Response Assessment using DeepEval GEval Framework

1. Introduction

The purpose of this report is to document the evaluation process of LLM-generated responses using the DeepEval framework. This evaluation focuses on correctness, ensuring that generated responses align meaningfully with expected responses while identifying misleading, fake, or contradictory information.

2. Implementation Details

2.1. DeepEval Framework and GEval

DeepEval is an evaluation framework designed for assessing LLM outputs based on various metrics. One of its key components, **GEval**, facilitates automated evaluation by leveraging an LLM-based metric to compare the **generated response** against the **expected response** based on predefined criteria.

In this evaluation:

- **Correctness** is assessed by ensuring that responses convey the same meaning as expected, with allowances for minor wording differences but penalizing misleading or incorrect information.
- The evaluation is conducted using **GPT-4o** as the reference model.
- **LLMTestCase** is used to structure test cases with inputs, actual outputs, and expected outputs.
- The correctness score is stored in a DataFrame for analysis.

2.2. Code Implementation

The following steps were followed:

1. **Load JSON Data:** The dataset containing LLM-generated responses and expected answers was loaded.
2. **Define Evaluation Criteria:** GEval was configured to assess correctness with explicit penalization for misleading or incorrect responses.

3. **Evaluate Each Response:** Each test case was processed through the correctness metric, generating a score.
4. **Store and Display Results:** The evaluated results were structured into a Pandas DataFrame.

3. Results and Insights

A sample of the results from the evaluation is provided below:

	Instruction	Expected Response	Correctness Score
0	What is the rule-based RM used to g	specific rules	0.8594371954525061
1	What are the requests generated fro	batches	0.8661881223502761
2	What does DeepSeek-R1-Zero gener	reasoning process, followed by the fi	0.815331794412979
3	What is the name of the book that w	Nature	0.7146427367275663
4	What is the purpose of the daily unlo	every line shared becomes collective	0.8437254559710575
5	What is the name of the version of D	v1.5	0.8007898680172109
6	What are the most common questio	Large Language Models	0.7156535978388704
7	What is the name of the service that	Data recovery	0.7310210061945667
8	What can researchers use to create t	DeekSeek models	0.7240245154563858
9	What is the state of the predecessor	The full	0.8731773601397188
10	What does DeepSeek-R1 demonstra	DeepSeek-V3	0.7366748877947223
11	What is the CRAQ write-all-read-any	helps to unleash the throughput	0.7141436209929193
12	What is the name of the book that is	arXiv preprint	0.7170875876375195
13	What will open-source 5 repos starti	Feb 24, 2025	0.712345
14	What are the version numbers of per	<code>v</code>	0.7465985091639817
15	What is the name of the method tha	bubble	0.7240108204560609
16	What are the main questions that ca	cluster manager, metadata service, s	0.9005438648759063
17	What did OpenAI generate?	inference-time scaling	0.7435834367007237
18	What is the most powerful aspect of	beauty	0.7199308186289004
19	What is the first open research to val	RL	0.7502607162250898
20	What would be a big burden on met	Storing all file descriptors	0.8467889272129835
21	What is the most common question	test-time scaling	0.7165537725212197
22	What is the HumanEval-Mul dataset?	eight mainstream programming lang	0.7243064238734033
23	What are the meta services that clus	online	0.7249843725759892
24	What are the IOPS of removing ops f	The bottom figure	0.812345

	Instruction	Expected Response	# Correctness Score
25	What are the two techniques to prev	visualized in following figure	0.7149407357480363
26	What are the questions that you can	specified instructions	0.7802383887374321
27	What is the 32B base model?	14 Model AIME 2024	0.7134436944097988
28	What did the test cluster generate?	25 storage nodes	0.7135881649538283
29	Generate questions from: All entries	range queries	0.8701862937426108
30	What can be generated from the req	latest chain table	0.7143782354604125
31	What does the Asynchronous zero-c	file system client	0.9001437173153686
32	What is the name of the preprint of a	Appendix A	0.7160776887745719
33	What is the DeepSeek-R1-Zero traini	a steady and consistent enhancemer	0.7183536649753227
34	What are the base models we use?	Qwen2.5-Math-1.5B	0.734348106060793
35	What can the client generate?	chunk IDs and chains	0.9212067497376804
36	What is the name of the preprint of a	arXiv:2402.03300, 2024	0.7727377206005387
37	Who is the Jin 20 Ruyi Chen Shangha	Jin 20 Ruyi Chen Shanghao Lu Shanç	0.7386188765383747
38	What is the main feature of DeepSee	poor readability, and language mixin	0.7349667744484949
39	What is the name of the model that	DeepSeek-R1	0.716553772896695
40	What do we want to generate questi	fine-tuned model	0.7610225997184288
41	What will DeepSeek-R1 help the rese	open source	0.79658392724232
42	What did DeepSeek generate one ac	computational cost	0.7264603685649046
43	DeepSeek-R1 is able to write tasks a	AlpacaEval2.0 and ArenaHard	0.7264603689338534
44	What temperature is used to generat	sampling temperature of 0.6 and a t	0.9230193444401715
45	What is the reference model for Llan	405B parameters, 15 T tokens	0.7249843717118213
46	What is the PRM?	Process Reward Model	0.7206220855172412
47	What are the most remarkable aspec	the emergence of sophisticated beha	0.9440467552533408
48	What is the name of the test?	DeepSeek-R1-Distill-Qwen	1.199748265671592
49	What is the base model of DeepSeek	Reinforcement Learning on the Base	0.7704576510031669

3.1. Observations

- **High Scores (~0.85+):** Many responses scored highly, indicating that the generated answers were mostly accurate with minor variations in wording.
- **Mid-Range Scores (~0.7-0.8):** Some responses had moderate correctness, likely due to minor deviations in interpretation.
- **Lower Scores (~0.6-0.7):** Responses in this range might have contained partial or incorrect information, warranting further review.

3.2. Key Takeaways

- **The framework successfully automates correctness evaluation**, providing a quantitative assessment of LLM-generated responses.

- **GEval with GPT-4o proves effective in identifying misleading responses**, ensuring factual accuracy in AI-generated outputs.
- **Potential improvements include** refining criteria to better differentiate between minor discrepancies and genuinely incorrect answers.

4. Conclusion

This evaluation framework enables structured and scalable assessment of LLM responses, helping ensure correctness and reliability in AI-generated content. Future improvements could involve integrating additional evaluation metrics such as fluency, relevance, and coherence for a more comprehensive assessment.