OptiMinders – Task 03

Technical Report

GitHub repository Link: https://github.com/OptiMinders-IntelliHack5/Intellihack_OptiMinders_Task3 (Saved model couldn't be uploaded)

HuggingFace model Link: Nipuni100/OptiMinders_LLMModel

1. Introduction

This report documents the process of fine-tuning the Qwen 2.5 3B model to answer questions based on recent AI research papers, blogs, and documents. The objective was to create a specialized model capable of retrieving, interpreting, and generating accurate responses from technical AI literature. This report includes all methodologies, attempts, successes, and failures encountered during the challenge.

---

2. Approach and Methodology

2.1 Dataset Creation

- **Synthetic Dataset Generation**
    - Used requests, xml.etree.ElementTree libraries to generate the papers.
    - Employed techniques such as manual QA pair creation
    - Preprocessed documents by tokenizing, removing stopwords, and converting text into QA pairs.
    - Saved the data as a Jason file. **arxiv_qa_dataset.json**
- **Dataset Implementation (from Jupyter Notebook)**:
    - Applied tokenization on the questions and answers using the Qwen tokenizer to ensure compatibility with the model.
    - Split the dataset into training and validation subsets for structured evaluation.

Approaches used:

1. **Pytorch data set - FAILED**

The Torkenzied data didn't fit with the pytorch data set.

2. **Tensorflow data set – PASSED**

Went forward with this approach.

Took keys for both questions and answers.

2.2 Model Selection

- **Base Model**:
    - Chose Qwen/Qwen2.5-3B-Instruct from Hugging Face due to its optimized architecture for instruction-based tasks.
- **Rationale**:
    - Its instruction fine-tuned variant aligns with the goal of AI research QA.
    - Its manageable size allows for efficient fine-tuning and deployment.

---

3. Fine-Tuning Process

- Created a virtual environment to ensure dependency isolation.
- Installed essential libraries including PyTorch, transformers, and tokenizer packages.
- Used a GPU-backed Google Colab environment and a Jupyter notebook for computational efficiency.

4.Save the Model

```
from transformers import AutoModelForCausalLM, AutoTokenizer

# Load the tokenizer and model
model_name = "Qwen/Qwen2.5-3B-Instruct"  # Replace with the actual model name you are using
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
```

```
Sliding Window Attention is enabled but not implemented for `sdpa`; unexpected results may be encountered.
Loading checkpoint shards: 100%|██████████| 2/2 [01:08<00:00, 34.50s/it]
```

```
# Save the model and tokenizer locally
model.save_pretrained("./saved_model")
tokenizer.save_pretrained("./saved_model")
```

5. Quantization Efforts

- Produce a 4-bit quantized version of the fine-tuned model in .gguf format for efficient inference.

Quantization Libraries Explored

- **BitsAndBytes**:
    - Installed successfully but encountered PackageNotFoundError for dependency issues.
    - Tried resolving by reinstalling in a new virtual environment, but the issue persisted.
- **AutoGPTQ**:
    - Installed auto-gptq==0.2.0 after multiple version conflicts.
    - Encountered ModuleNotFoundError due to incomplete installation or missing dependencies.
- **Transformers Quantization**:
    - Attempted 4-bit quantization using transformers library.
    - Error: Unsupported quantization configuration for Qwen 2.5 3B.

Virtual Environment Isolation

- Created a virtual environment (venv) to isolate dependencies.

- Successfully installed base dependencies but faced unresolved issues with the required quantization libraries.

Summary of Failures

- Dependency conflicts and incomplete installations were the primary barriers.
- Lack of official support for .gguf format conversion from Qwen 2.5 3B.
- Computational limitations hindered debugging efforts.

Final Outcome

- Successfully fine-tuned and saved the model using
- model.save_pretrained("./saved_model")
- tokenizer.save_pretrained("./saved_model")
- Could not produce the 4-bit quantized model in .gguf format despite multiple attempts using different libraries and setups.

---

6. Possible Causes for Quantization Failures

1. **Dependency Conflicts**
    - Conflicting versions of PyTorch and CUDA libraries required by quantization tools.
2. **Incomplete Library Support**
    - Limited support for Qwen models in quantization tools such as BitsAndBytes and AutoGPTQ.
3. **Unsupported Format**
    - Lack of direct support for .gguf format within existing quantization libraries.
4. **Virtual Environment Issues**
    - Isolated environments did not resolve all dependency requirements.

---

7. Evaluation

- Evaluated the fine-tuned model using a held-out test set of QA pairs.
- Achieved a notable improvement in accuracy for technical questions compared to the base model.
- Due to the inability to quantize the model, inference efficiency was not tested as planned.
