Figr Design Assignment

Submitted by: Parteek Kamboj email: 22f1000482@ds.study.iitm.ac.in

Task 1: Dataset Preparation

- I accessed the dataset using the provided Hugging Face link. Since the dataset is very large, I used the streaming method to handle it efficiently. For this task, I focused on the columns 'text' and 'llm_generated_idea' to prepare the data for fine-tuning. I limited the extraction to 20,000 rows to manage the dataset size effectively.

- To prepare the data for fine-tuning, I used Beginning-of-Sequence (BOS) and End-of-Sequence (EOS) tokens. The specific tokens used for BOS and EOS were applied to structure the data properly. Each entry in the 'text' column was transformed by concatenating a fixed prompt with the content from the 'llm_generated_idea' column and the original 'text'. This step ensures that the input data is standardized, providing a consistent format for the model to generate code based on the given instructions.

- After processing, I saved the transformed data into a CSV file named 'final_data.csv'. This file contains the formatted data ready for use in training the Stable Code Instruct model.

  Link of the final data: https://huggingface.co/datasets/keetrap/data4tune

Task 2: Find a suitable model to Fine tune or Build a Rag

- **LLM Model Selected:** Stabilityai/stable-code-instruct-3b
  This model is specially designed for coding purposes and performs better than some 7B LLMs.

- **Fine-Tuning vs. Building a RAG:** I chose to fine-tune the model rather than build a RAG because RAG might not be as effective in generating highly specialized content like HTML and CSS code, as it depends on external data. Fine-tuning, on the other hand, focuses specifically on the task at hand, making it more effective for generating precise and accurate HTML and CSS code based on prompts. By training on examples of website code generation, the model learns the nuances and specifics required for this task, leading to more accurate and relevant output. Fine-tuning embeds knowledge directly into the model, minimizing reliance on external sources and improving output control.

- **Tokenization and Quantization:** To prepare the text data for model input, the padding token is set to match the end-of-sequence **(EOS**) token to maintain consistency throughout the sequences. The text data is then tokenized, with all sequences padded

or truncated to a maximum length of 2048 tokens. Finally, the tokenized data is converted into PyTorch tensors, making it ready for processing by the model.

Using a dataloader, the tokenized data is loaded with a batch size of 2 (GPU memory does not permit a batch size greater than 2).

To optimize model performance and memory usage, the model is configured to load in **4-bit** precision using the **BitsAndBytesConfig**. The configuration employs **nf4** quantization for improved precision and uses the bfloat16 data type for computations. The model is then loaded with this quantization configuration, with the GPU automatically mapped.

- **Low-Rank Adaptation and Model Optimization:** To enhance the model's efficiency, a LoRA (Low-Rank Adaptation) configuration is applied specifically for causal language modeling. This configuration adjusts the model by targeting key components such as **query, key, value**, and output projections with a low-rank adaptation technique, which is controlled by scaling and dropout parameters. The model is then fine-tuned using the **AdamW** optimizer with a **learning rate of 2e-5**, and its learning rate is adjusted over time using a step scheduler that gradually reduces the rate to improve convergence.

- **Single Epoch Training and Model Saving:** Due to memory and time constraints, the model is trained over a single epoch using 10,000 rows of data (which takes almost 8 hours). During each iteration, the model processes input prompts and corresponding code labels, computing the loss. The optimizer is then updated to minimize this loss, with a learning rate scheduler adjusting the learning rate to ensure stable training. After completing the epoch, the model's performance is evaluated based on the average loss, and the fine-tuned model is saved for future use.

- **Link for the tunned model:** https://huggingface.co/keetrap/final_tune_model

- **Results and Possible Improvements:** After one epoch, the average loss is about 18.27, which is **quite high.** However, the model was trained over a single epoch. If we try to train it for more epochs and over more data, the loss will likely decrease. I generated a few web pages, and the results look acceptable but not perfect. Screenshots of the generated web pages are attached in the images directory of the shared GitHub repo link.
  There is much scope for improvement. First of all, we should choose a model with more parameters, train over a larger dataset, and consider using **DPO (Direct**

Figr Design Assignment

Submitted by: Parteek Kamboj email: 22f1000482@ds.study.iitm.ac.in

Preference Optimization) for fine-tuning, as it has several advantages over typical fine-tuning methods.

- **Experience About the Assignment:** I had a great experience doing the assignment. There was so much to learn from this task. As the assignment mentioned not to use any wrappers for fine-tuning, I initially thought about how this could be done without fine-tuning libraries. I tried to search online for example code but found nothing because everyone is using the library. I do not know how correct I am with the fine-tuning method I used in this assignment, but somehow, I was able to do it.

  Thanks for this opportunity.