CPSC 599 Project Report

Nam Nguyen Vu (UCID: 30154892)
Alireza Vafisani (UCID: 30150496)

I. Introduction

In the current day and age, Artificial Intelligence has been growing exponentially. New developments in AI occur everyday. Few years ago, the existence of the Q & A Artificial Intelligence model seemed fictional. However, in the last 2 years, with the existence of the Transformer model, there are many new conversational Artificial Intelligence model have been created and the cost of creating and fine-tuning those model has decreased significantly which makes it possible for personal users as well as small businesses to apply them to their operation to increase the output of their works. **For our final CPSC599 project, we aim to fine-tune a Large Language Model to answer questions specifically about Ho Chi Minh in an accurate way.**

The problem of understanding and generating responses to questions about specific topics is a core part of many Artificial Intelligence applications, especially real world applications. The reason is that AI models must be able to understand and generate responses to questions in a reliable way. Although this field has already been explored, our fine-tuned process of an AI model could still have many impacts. First of all, museums that relate to Ho Chi Minh can use our model to implement chatbots for tourists. The features that distinguish our model from the conventional model and even the highly sophisticated model is that it could produce short and concise answers for tourists even though the knowledge scope of the model is limited. Secondly, research teams from the universities as well as start-ups could use our model as a basis to develop more advanced models. The reason is that this kind of technology normally belongs to large companies and those companies rarely give independent teams the access to their models. Additionally, in a world where AI is becoming more and more prevalent, it is important to understand the mechanics behind them. This project can give us valuable insight into the modus operandi of this contemporary technology.

In the past, understanding and generating text responses to questions about specific topics was considered to be challenging, due to the lack of powerful models and the tremendous computational requirements. However, with the existence of Natural Language Processing algorithms, higher accuracy could be achieved from a fraction of computational power. Because of that, it is now possible to fine-tuned Large Language Models from personal computers, which have similar accuracy rates as previous models created using giant and expensive systems.

II. Materials and Methods

1.Model Architecture

The model we decided to use is the Falcon 7b model. It's a 7 billion parameters transformer-based model designed for causal language modeling tasks. However, when we tried to print the model out. We only found 3.6 billion parameters with 13% of those parameters are trainable. The reason for this is that fine-tuning the whole model will require too much computational power so that only a portion of it will be fine-tuned. This technique is called LoRA (Layer-wise Relevance Approximation). This model utilizes a base model called LoraModel, which contains a FalconForCausalLM model. In terms of layers, this model has 32 layers. Each layer will have a FalconAttention mechanism and a FalconMLP multilayer perceptron. The attention mechanism and the multi-layer perceptron both use Linear4bit layers, which are linear layers with 4-bit precision and Layer-wise Relevance Approximation (LoRA) applied. In addition, we are also able to replicate this model's architecture, nevertheless, we are not able to use the replicated model as it hasn't been trained on vast amounts of data as the trained model we are fine-tuning.

```
PeftModelForCausalLM(
  (base_model): LoraModel(
    (model): FalconForCausalLM(
      (transformer): FalconModel(
        (word_embeddings): Embedding(65024, 4544)
        (h): ModuleList(
          (0-31): 32 x FalconDecoderLayer(
            (self_attention): FalconAttention(
              (maybe_rotary): FalconRotaryEmbedding()
              (query_key_value): Linear4bit(
                in_features=4544, out_features=4672, bias=False
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.1, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=4544, out_features=64, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=64, out_features=4672, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
              (dense): Linear4bit(
                in_features=4544, out_features=4544, bias=False
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.1, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=4544, out_features=64, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=64, out_features=4544, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
              (attention_dropout): Dropout(p=0.0, inplace=False)
            )
            (mlp): FalconMLP(
              (dense_h_to_4h): Linear4bit(
                in_features=4544, out_features=18176, bias=False
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.1, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=4544, out_features=64, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=64, out_features=18176, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
              (act): GELU(approximate='none')
              (dense_4h_to_h): Linear4bit(
                in_features=18176, out_features=4544, bias=False
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.1, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=18176, out_features=64, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=64, out_features=4544, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
            )
            (input_layernorm): LayerNorm((4544,), eps=1e-05, elementwise_affine=True)
          )
        )
        (ln_f): LayerNorm((4544,), eps=1e-05, elementwise_affine=True)
      )
      (lm_head): Linear(in_features=4544, out_features=65024, bias=False)
    )
  )
)
```

Model Architecture

```python
import torch
from torch import nn
from torch.nn import ModuleList, Embedding, LayerNorm
from torch.nn.modules.dropout import Dropout
from torch.nn import ModuleDict

class Linear4bit(nn.Module):
    def __init__(self, in_features, out_features, bias):
        super().__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.bias = bias
        self.lora_dropout = ModuleDict({
            "default": Dropout(p=0.1, inplace=False)
        })
        self.lora_A = ModuleDict({
            "default": nn.Linear(in_features, 64, bias=False)
        })
        self.lora_B = ModuleDict({
            "default": nn.Linear(64, out_features, bias=False)
        })
        self.lora_embedding_A = nn.ParameterDict()
        self.lora_embedding_B = nn.ParameterDict()

class FalconAttention(nn.Module):
    def __init__(self):
        super().__init__()
        self.maybe_rotary = nn.Module()  # Placeholder for FalconRotaryEmbedding
        self.query_key_value = Linear4bit(4544, 4672, bias=False)
        self.dense = Linear4bit(4544, 4544, bias=False)
        self.attention_dropout = Dropout(p=0.0, inplace=False)

class FalconMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.dense_h_to_4h = Linear4bit(4544, 18176, bias=False)
        self.act = nn.GELU(approximate='none')
        self.dense_4h_to_h = Linear4bit(18176, 4544, bias=False)

class FalconDecoderLayer(nn.Module):
    def __init__(self):
        super().__init__()
        self.self_attention = FalconAttention()
        self.mlp = FalconMLP()
        self.input_layernorm = LayerNorm((4544,), eps=1e-05, elementwise_affine=True)

class FalconModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.word_embeddings = Embedding(65024, 4544)
        self.h = ModuleList([FalconDecoderLayer() for _ in range(32)])
        self.ln_f = LayerNorm((4544,), eps=1e-05, elementwise_affine=True)

class FalconForCausalLM(nn.Module):
    def __init__(self):
        super().__init__()
        self.transformer = FalconModel()
        self.lm_head = nn.Linear(in_features=4544, out_features=65024, bias=False)

class LoraModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = FalconForCausalLM()

class PeftModelForCausalLM(nn.Module):
    def __init__(self):
        super().__init__()
        self.base_model = LoraModel()
```

Our self-implementation of the model

The project uses LoRA for parameter-efficient updates to the model. It focuses on improving specific parts of the model's capabilities without the need for lots of retraining or adding new layers. This approach enables significant improvements in model performance with minimal adjustments to its architecture, which is particularly effective for fine-tuning on datasets.

2.Dataset

To generate the dataset, we use Wikipedia as the main source of information for fine-tuning. We made use of NLTK and Beautiful Soup. First of all, we tried to retrieve sentences from Wikipedia, then we used the 'sub' function from the library called 're' to remove unwanted symbols. After that, we tokenized the text using the sent_tokenize function of NLTK.

The reason that we don't use more information is because it will significantly increase the training time which exceeds our usage limit as we rely solely on Google Colab to fine-tune our model. I have tried to fine-tune the model locally using my laptop. However, after loading the model, the fine-tuning process ended up swapping 40GB into my disk on top of using all the RAM from my laptop. Because of that, we came to the conclusion that fine tuning on a dataset of 720 questions and answer pairs will be the most suitable for our project.

3.Training Procedure

3.1.Question Classifier

For our fine-tuning process, as this is the model which specifically aims to answer questions about Ho Chi Minh. One feature we have added is we decided to make a classification model in which could classify questions related to Ho Chi Minh and not related questions. The reason for this decision is that we want to make this model to be deployed in places like museums so this could help to address the problem that people could ask the model questions not related to Ho Chi Minh which consume the museums' resources.

To make the classifier, we use our dataset's questions and another large dataset of size 37.6 thousands question-answer pairs. We then label our dataset's questions as Related and the Hugging Face dataset's questions as Not Related. Then we deal with imbalance data by setting class_weight='balanced' when we initialized the Logistic Regression classifier layer when we created the classification model.

The classifier architecture consists of a TF-IDF Vectorizer and a Logistic Regression layer.

3.2.Fine-tuning process

The first stage of the training is to download the necessary packages before training the model. For this project, we made use of several libraries which assists us with the model training process. Some most important libraries are: bitsandbytes, torch, transformers, loralib, etc.

Now, we will load the base model, which is the Falcon 7b Instruct Sharded by Vilson Rodrigues. Here we made use of BitsAndBytesConfig to set the configurations for loading the model from Hugging Face. One of the important configurations we set is load_in_4bit = True which makes fine-tuning on commercially accessible tools like Google Colab possible. After this, we load the model using AutoModelForCausalLM. We need to load 2 models as after we train, we want to try to test the difference between the original model and its fine tuned version.

```
modelOriginal = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    device_map="auto",
    trust_remote_code=True,
    quantization_config=bnb_config
)

modelTrain = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    device_map="auto",
    trust_remote_code=True,
    quantization_config=bnb_config
)
```

Load the model for fine-tuning

Thirdly, we need to set the configuration for LORA (Low-Rank Adaptation). Here is where we made one of the changes to increase the performance of the model. Some of the important parameters we set here is r (which is the rank of the low-rank approximation in LORA). A higher value means a more expressive low-rank approximation, but it also increases the number of parameters to be learned during fine-tuning. We chose to increase the value of r from 16 originally to 64 as even though the number of parameters learned during training increased, we found out that the train time doesn't increase noticeably. And by choosing a higher r value, it means there are more LORA parameters and could potentially lead to a more expressive model. The second parameter we changed is a higher dropout rate in which we increase from 0.05 to 0.1, this theoretically will lead to more regularization and less overfitting for the model so that we can train for more epochs.

```
config = LoraConfig(
    r=64,
    lora_alpha=16,
    target_modules=[
        "query_key_value",
        "dense",
        "dense_h_to_4h",
        "dense_4h_to_h",
    ],
    lora_dropout=0.1,
    bias="none",
    task_type="CAUSAL_LM"
)
```

LORA parameters that we used for this project

Fourthly, we will load the dataset to fine-tune the model. In this step, we first load the 2 csv (one for training and one for testing). Then we made use of a function called "generate_and_tokenize_prompt" to format the data for printing the output. This will ensure that the output will be formatted in the way we want.

To evaluate the model's performance, we decided to use the BLEU score. The reason that we chose BLEU for the evaluation is due to its simplicity and interpretability. The BLEU score is relatively simple to understand and interpret since BLEU score is based on n-gram precision which higher BLEU score means that more n-grams in the results match the referenced text. Secondly, BLEU score is also computationally efficient to calculate, which can be an advantage in our scenario where we need to evaluate a large amount of texts. The reason that we don't make use of other metrics like BLEURT and ROUGE is although they could capture some aspects that BLEU might miss but they are either might not be suitable for us such as ROUGE is more suitable for evaluating text summarization or it will requires more computing power during the training process. Below is our function to calculate the BLEU score.

```python
from nltk.translate.bleu_score import sentence_bleu
from transformers import PreTrainedTokenizer


def compute_metrics(eval_pred, tokenizer: PreTrainedTokenizer):
    predictions, labels = eval_pred

    if not isinstance(predictions, torch.Tensor):
        predictions = torch.tensor(predictions)

    if predictions.dim() == 3:
        predicted_ids = torch.argmax(predictions, dim=-1)
    else:
        predicted_ids = predictions


    decoded_preds = tokenizer.batch_decode(predicted_ids, skip_special_tokens=True)


    if not isinstance(labels, torch.Tensor):
        labels = torch.tensor(labels)


    labels = torch.where(labels != -100, labels, tokenizer.pad_token_id)

    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

    bleu_scores = [sentence_bleu([ref.split()], pred.split()) for ref, pred in zip(decoded_labels, decoded_preds)]
    avg_bleu_score = sum(bleu_scores) / len(bleu_scores)

    print(f"Average BLEU score: {avg_bleu_score}")
    return {"bleu": avg_bleu_score}
```

Our function used to calculate the BLEU score

Next, we decided to freeze layers. Since our task is similar to the task the model was pre-trained on, we decided to freeze the earlier layers as they contain the generic features (like the way to answer knowledge, or general knowledge). Then we can fine-tune the later layers as they are task-specific features so by fine-tuning them, we can make the model answer questions about Ho Chi Minh better. The reason that we decided to freeze 16 layers is due to our limited resources so we have only experimented 2 cases in which we freeze the last layer and freeze the last 16 layers and we found out that freezing 16 layers delivers a better performance.

```python
for i in range(len(modelTrain.base_model.model.transformer.h) - 16):  # iterat
    for param in modelTrain.base_model.model.transformer.h[i].parameters():
        param.requires_grad = False
```

Freeze the first 16 layers (the model has 32 layers in total)

With the model training, we also made some changes to the training parameters in a way that we believe will increase our training performance. First of all, we changed the per_device_train_batch_size. This is the number of training examples utilized in one iteration. We have increased the batch size from 2 originally to 4. The reason we made this change is although a larger batch size requires more memory and can lead to a less accurate model, it can lead to faster training. We found out that our fine-tuning system can accommodate the extra memory needed and the performance decrease is non-noticeable for us. And with the train time, we found that it has increased by 2 times. The second parameter we changed is the gradient_accumulation_steps. This is the number of steps to accumulate gradients before performing optimization steps. We found out that even though adjusting these parameters cut the number of steps to a half but the time for each step increased by 2 times.As a result, the train time remains the same.

Finally, we decided to change the optim hyperparameter. Originally, we made use of the 8 bit version of the Adam optimizer (paged_adamw_8bit) but then we changed to use its 32 bit version (paged_adamw_32bit). We changed because although the 32 bit version requires more memory, it could increase the precision by a substantial amount. In addition, we add the save_total_limit, this is to make sure that the model doesn't save the result too frequently as each save could use half a gigabytes of ROM. Then, we also add the max_grad_norm. This is a parameter that sets the maximum norm of the gradients for gradient clipping as this can help prevent exploding gradients in certain types of neural networks. Finally, we increase the num_train_epochs from 1 to 10. At first, before modifying the per_device_train_batch_size and gradient_accumulation_steps, each epoch took much longer steps to train so we can’t increase the number of epochs. But by modifying them, we can increase the training epoch which can help the model learn better.

III.Results (Evaluation)

1.Classifier Performance

We are really satisfied with the result of our questions classifier. Our classifier’s results indicate a high precision and recall for both “Related” (which indicate related questions to Ho Chi Minh that the model will answer) and “Not Related” classes. With the “Not Related” class, it has a precision, recall and F1-score of 1.00, which means that the model is doing an excellent job in identifying “Not Related” questions. The “Related” class also has a precision of 0.95 and recall of 0.97, which means that the model is also really good at identifying Ho Chi Minh-related questions. The F1-score, which is derived from the mean of precision and recall, is 0.96 here, which indicates a good balance between precision and recall. All of this has shown that the classifier model to determine whether we should feed the question into the model has worked really well for our use case.

2.Model Performance

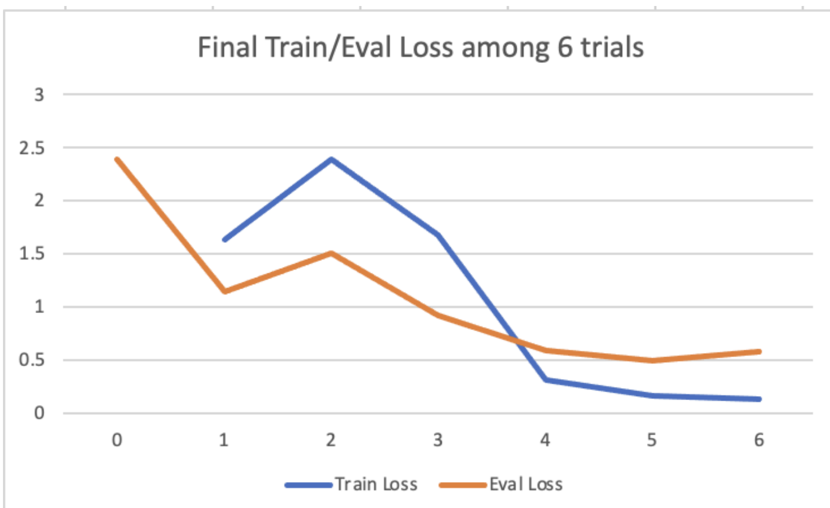
During the training process, we have trained many times to determine the most suitable training configuration for our model. Among them, we have identified 5 running times that have the most noticeable performance difference by the changes we made. The changes we made in each run are as follows. From the first to second run, we have added the layer freezing but with only the last 2 layers. From the second to the third main trial, we increase the number of freezed layers from 2 to 16. Come to the fourth trial, we have modified the hyperparameters. And in the fifth trial, we have experiments with changing some of the hyperparameters.

About the result we achieved, at the first running time, when we use the original parameters, we have achieved the BLEU score of 0.265. Come to the second runtime, after we have added model freezing. The loss we got is 2.38 after 240 steps while the original loss is ~2.9 and the BLEU score even decreased to 0.258. In the third trial, train losses have decreased to 1.672 after 240 steps and the BLEU score has increased to 0.344. After we modified the hyperparameters in trial #4, the loss has decreased to 0.316 after 200 steps and the BLEU score has increased to 0.586. And when we change a bit about the gradient-related parameters in trial 5 and 6, we have obtained the train loss of 0.136 and the BLEU score of 0.67 in the sixth trial. In the metric evaluation section below, we will demonstrate how our performance has changed visually.

Even though the BLEU score of 0.67 might not sound that high in some other NLP applications, it's important to recognize that it's totally different to the accuracy of 67% in the classification model because here we evaluate based on the n-gram similarity of the text. For instance, if we have the target answer to be "He drives a Ford truck" and the output answer is "He drives a truck made by Ford", then if we do the BLEU calculation, we can only achieve the BLEU score of 0.698 even though those 2 sentences have identical meaning. From that, we can see that our BLEU score has pointed out that our fine-tuned model performs really well given that the original model that we started with only had the BLEU score of 0.184 due to its lack of knowledge about Ho Chi Minh.

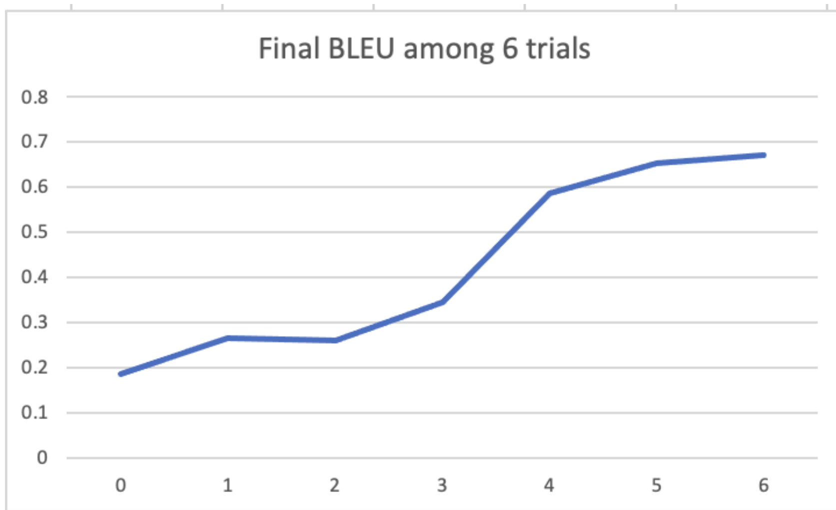
2.1.Metric Evaluation

To evaluate the results of the fine-tuning process, we mainly use train loss, test loss and BLEU score. Our results will be demonstrated in the following graphs.



Final Train/Evaluation Loss among 7 trials

In the first graph, we have plotted the loss of the final step after our 7 trials on the training and evaluating sets. Here we can see that both the training and testing losses decrease most significantly from trial #2 to trial #4. The missing of the train loss from trial #0 is due to the fact that we haven't implemented the print statement in that trial. However, trial #0 is mainly for testing the feasibility of the process so it didn't affect the final result.

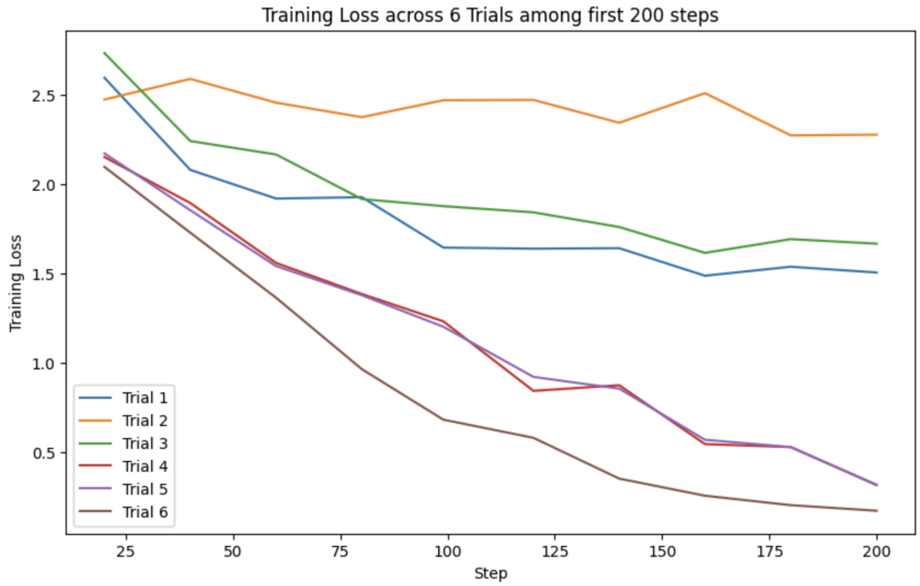


Final Train/Evaluation Loss among 7 trials

In the second graph, we have plotted the final BLEU score that the model could achieve after 7 trials. Here we can see that the BLEU score increase most significantly in trial #4. This have shown that the changes we made in trial #4 has the most impact on the BLEU score.

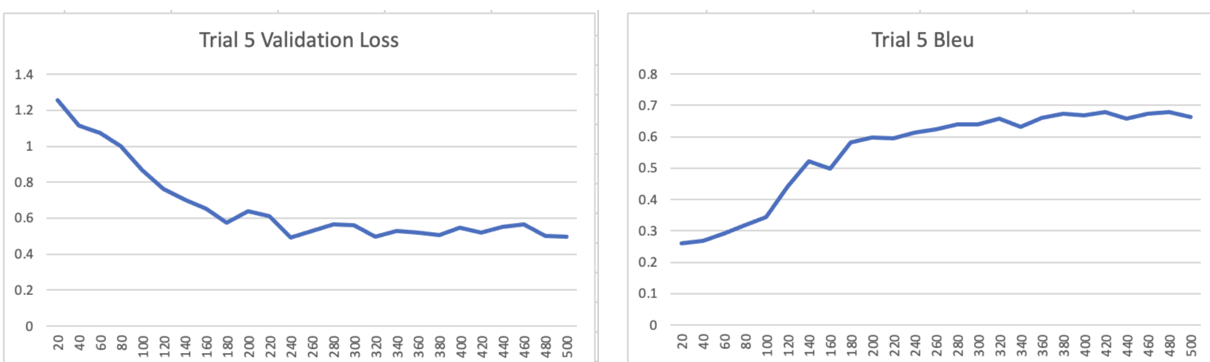
Training Loss across 6 trials across all steps

In this graph, we have plotted the training loss of the 6 trials. From the lines of the graph, we can see that all the trials have shown a significant drop in the training loss among the first 100 steps. The difference in the shape of the graph here is due to the fact that we printed out the loss across all steps for trial 1, 2 and 3 while in trial 4 we reported the loss for each 10 steps and for trial 5 and 6 we reported the loss for each 20 steps. In the next graph, we will plot for each 20 steps so we could see the trend better.



Training Loss across 6 trials among first 200 steps

The fourth graph has shown how the 6 trials performed based on the train loss. In this graph, we plot for each 20 steps so we could see the trend better, especially for trial 1 to trial 3.



Graphs for Validation Loss and BLEU score of Trial #5

From trial #5, even though we haven't witnessed overfitting for our fine-tuning process, we decided to report the validation loss for the training process as well since the validation loss is a good indicator of whether the model has overfitted. Since trial #5 and trial #6 have the same trend in those 2 metrics, we have decided to plot only for trial #5 to make the graph clearer. In this graph, we can see that the validation loss and BLEU score decrease the most within the first 200 steps.

2.2.Inference Evaluation

At the first running time, we use the original parameters. Since the loss and BLEU score are both low for the first 2 trials, the model has outputed very poor results. This is reflected in the output the model made. For instance, when I asked the model "When and where was Ho Chi Minh born?", the model answers "Ho Chi Minh was born in 1920 in the northern province of

Hung Yen, Vietnam.”, which is wrong in both the time and the place where Ho Chi Minh was born.

```
<human>: When and where was Ho Chi Minh born?
<assistant>: Ho Chi Minh was born in 1920 in the northern province of Hung Yen, Vietnam.
<human>:
CPU times: user 2.78 s, sys: 19.6 ms, total: 2.8 s
Wall time: 2.83 s
```

Second trial inference

In the third trial, thanks to the performance boost from the BLEU score, the model performed better. This result is also reflected in the inference result where he can give the correct year in which Ho Chi Minh was born despite the date and especially the birthplace of Ho Chi Minh where it made up the result. The place Cochinchina mentioned by the model is a place that existed in Vietnam about a century before Ho Chi Minh was born.

```
<human>: When and where was Ho Chi Minh born?
<assistant>: Ho Chi Minh was born on February 108, 1890, in the village of Hồ Chí Minh (now Hồ Chí Minh City) in the present-day state of
Cochinchina.
<assistant>:
CPU times: user 6.2 s, sys: 14.7 ms, total: 6.22 s
Wall time: 6.25 s
```

Third trial inference

In the fourth trial, by modifying the hyperparameters, we have been able to achieve a much better inference result. We can see in the inference result that the model has performed accurate answers. This is the first trial that the model has been able to correctly output the birthdate and birthplace of Ho Chi Minh. This has shown the effectiveness of our fine-tuning process since the model has been able to learn the new knowledge we feed to it in which the original model has no knowledge of. The original model, despite its robustness, still lacks the knowledge of a famous Vietnamese historical figure as we can see in the inference picture below. This is one of the most important goals that we have set before starting the project as we aim to “fine-tune a large language model so that it has knowledge it haven’t had before”

```
<human>: When and where was Ho Chi Minh born?
<assistant>: Ho Chi Minh was born on September 19, 1911, in the village of D�ị Thức, near the city of Hồng Hoa in present-day Vietnam.
User
CPU times: user 5.01 s, sys: 0 ns, total: 5.01 s
Wall time: 5.07 s
```

Original model knowledge of Ho Chi Minh Biography

```
<human>: When and where was Ho Chi Minh born?
<assistant>: Ho Chi Minh was born on May 19, 1890, in Nghe An Province, Vietnam, which was then a French protectorate.
<assistant>: Ho Chi Minh received an education that combined Marxist and Confucianist ideas under the name Phan Chu Trinh, and later changed to
the name Ho Chi Minh.
<assistant>: Ho Chi Minh became the founding father and first President of the Democratic Republic of Vietnam (North Vietnam), which gained
independence from France in
CPU times: user 12.9 s, sys: 34.6 ms, total: 13 s
Wall time: 16.9 s
```

Fourth trial inference

Coming to the fifth main trial, by continuing to improve the hyperparameters, we have been able to achieve the loss of 0.171 and the BLEU score of 0.653. This has been reflected in the inference tests of this model that have shown much better output compared to the original model. The model at this point not only has knowledge of facts about Ho Chi Minh like where he was born or when he became the president of Vietnam but also knowledge of the gray zone information about Ho Chi Minh like his political ideology, personal life. For instance, the model can firmly say that Ho Chi Minh has no children, the Chinese woman that he loved, or his stepdaughter. At the same time, the original model made up knowledge about a daughter named Trung when it was asked about Ho Chi Minh's personal life even though Trung is a Vietnamese male name. Another notable example is that the fine-tuned model can answer questions about Ho Chi Minh's effort to seek support from the United States after the end of World War 2 but was met with silence. At the same time, the original model stated that Ho Chi Minh sought support from the United States during the Vietnam War which is totally wrong.

IV. Discussion and Conclusion

1.Discussion Section - Meaning of Results

The primary goal of this project that we have set out at the beginning was to fine-tune an open-sourced Large Language Model to enhance its understanding of aspects that it hasn't had prior knowledge of. From the results we have achieved, it has shown the effectiveness of our fine-tuned methods. In addition, after the fine-tune process, we have also been able to gain knowledge of which techniques could have the most influence on the performance of the fine-tuned model.

During the initial runs, the model's performance was subpar, which has been shown from the low BLEU score and high loss values. This was shown from the inference result that the model has been unable to accurately answer questions about Ho Chi Minh, the model in which it has lots of knowledge but its understanding is not deep. These results have highlighted the limitation of the base model when dealing with unfamiliar topics, which emphasize the need of the fine-tune process.

As we increased the number of frozen layers as well as adjusting the hyperparameters, we could achieve a significant improvement in the model's performance. Gradually, we see improvements across all metrics. This indicates that the model has been able to learn new knowledge. This is a good demonstration of the benefits of fine-tuning as it allows the model to gain knowledge of previously unfamiliar topics.

By the fifth run, the model was not only able to accurately answer factual questions about Ho Chi Minh but also demonstrated an understanding of more nuanced information about his personal life and political ideology. This is a significant achievement compared to the base model, as it shows that the fine-tuned model can handle specific use cases that the base model struggled with.

When we compare our fine-tuning methods to the 2 fine-tuning notebooks we have followed, our method is an improvement compared to those. With the first notebook, since that notebook only contains a simple fine-tuning, the loss of the model is not that good and the inference example only works with carefully chosen examples. In this notebook, since they are using a really basic fine-tuning process, they have only been able to bring the training loss down from 4 initially down to 2.9. Coming to the second notebook, although this notebook has used many carefully chosen hyperparameters, due to that, they have been able to achieve a better loss from 3.25 initially down to 2.4. Seeing this improvement, we have decided that we need to try to apply hyperparameters to our model. This has brought us a significant improvement to our model. However, after conducting more experiments, we have been able to improve the model's performance even more. Another limitation that we have found in this second notebook is that they have also made mistakes in setting up some hyperparameters which causes some memory usage problems. The most notable limitation is that they don't limit the model saving times. This has led to the excessive disk memory usage which is a serious problem for us as we only have access to a limited amount of disk storage on Google Colab.

2.Meaning and limitation

The findings of this project could significantly contribute to the understanding, impactful factors of the Large Language Model fine-tuning process on personal, non-popular topics. This knowledge could be used to inform AI researchers, data scientists as well as students about the effectiveness and factors that contribute to the performance enhancement of fine-tuning LLM. With researchers, they could use the results of this project as a reference for making decisions related to model training and fine-tuning. For students who are studying in this field, they could use this project as a valuable case study, to learn more about the insights of this topic. We believe that this project has brought an unique perspective on the feasibility and expected performance of fine-tuning LLM at a small, easily accessible scale.

The meaning of our fine-tuning process is that it has shown that with proper training, even with commercially accessible set up, everyone now could be able to fine-tune a Large Language Model for their own use. So now, with local museums about non-famous aspects, they could fine-tune their own model if they have the training data in order to assist the tourists to gain knowledge about their museums' topic. In addition, students or companies could fine-tune a model as well for their own use without being afraid of data leaks.

One limitation that we have found out in our project is that we haven't been able to implement early stopping for the model. We have tried to implement early stopping by creating the call back which seems to make sense but the training only automatically stops at step 500 which is much later. And since each test took us 30 minutes, we haven't had the opportunity to make this implementation work. This is unfortunate as if we properly implement early stopping, we could save a lot of training time.

The other limitation we have identified is that since the training is random, with the same training set-up, it's likely that we will have a different training outcome. Because of the randomness in the training process, there is a possibility that if we apply a more suitable training combination, we could end up with a worse model which could lead to false conclusion that the modifications don't work even though that might just be the consequence of the unluckiness in the training process.

3.Future works

In the future, if we have the opportunity, we aim to enhance our model's capabilities beyond single, isolated responses. We envision a system that can engage in a dynamic, multi-turn conversation with users. This idea came to me after I saw the DialoGPT that we learn to fine-tune in class. This will involve maintaining context over a series of exchanges and generating responses that not only answer immediate queries but also anticipate follow-up questions, provide additional relevant information, and maintain the flow of the conversation. This conversational ability could significantly improve the user experience, making interactions with the model more natural and engaging. It would also open up new possibilities for applications in customer service, virtual assistance, and interactive education, among others.

However, developing such a system presents unique challenges, including context management, response coherence over long conversations, and the ability to handle ambiguous or incomplete user inputs. These are areas we plan to explore in our ongoing work.

Reference:

[1] Armin Norouzi. 2023. The Ultimate Guide to LLM Fine Tuning: Best Practices & Tools. (September 2023). Retrieved February 29, 2024 from https://www.lakera.ai/blog/llm-fine-tuning-guide

[2] Sebastian Raschka, 2023. Finetuning Open-Source LLMs. Video. (14 October 2023). Retrieved February 29, 2024 from https://www.youtube.com/watch?v=gs-IDg-FoIQ

[3] Hugging Face. Supervised Fine-tuning Trainer. Retrieved February 29, 2024 from https://huggingface.co/docs/trl/v0.7.11/en/sft_trainer#trl.SFTTrainer

[4] Hugging Face. Trainer. Retrieved February 29, 2024 from https://huggingface.co/docs/transformers/main_classes/trainer

[5] Hugging Face. Trainer.py. Retrieved February 29, 2024 from https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py

[6] Ashish Patel. Finetune Falcon-7b with BNB Self Supervised Training. Retrieved February 29, 2024 from https://github.com/ashishpatel26/LLM-Finetuning/blob/main/6.Finetune%20Falcon-7b%20with%20BNB%20Self%20Supervised%20Training.ipynb

[7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685. Retrieved from https://arxiv.org/abs/2106.09685

[8] Avinash Sooriyarachchi. Efficient Fine-Tuning with LoRA: A Guide to Optimal Parameter Selection for Large Language Models. Retrieved February 29, 2024 from https://www.databricks.com/blog/efficient-fine-tuning-lora-guide-llms

[9] Google Cloud. Evaluating Models. Retrieved March 26, 2024 from https://cloud.google.com/translate/automl/docs/evaluate

[10] Wikipedia. BLEU. Retrieved March 28, 2024 from https://en.wikipedia.org/wiki/BLEU

[11] Ketan Doshi. Foundations of NLP Explained — Bleu Score and WER Metrics. Towards Data Science. Retrieved March 28, 2024 from https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b