Fine-tune LLM for Language Translation

Dishant Lathiya
RPTU Kaiserslautern, Department of Computer Science

Note: *This report contains a project documentation and reflection on the portfolio task submitted for the lecture Engineering with Generative AI in WiSe 2024-25. This report is an original work and will be scrutinised for plagiarism and potential LLM use.*

1 Portfolio documentation

1.1 Introduction

Welcome to LLM fine-tuning project on German-French translation. This project aims at fine-tuning a pre-trained LLM, such as T5, so that it will translate from German to French proficiently. In this process, the model should be trained on learning syntax, semantics, as well as cross-lingual patterns so that it produces right and contextually relevant translations.
In order to achieve this, we employ actual and virtual datasets to enhance the accuracy and reliability of the model for translation in order to make it effective for real-world translation applications. The project is divided into distinct phases of research, design, implementation, and evaluation with a systematic process to tackle the specifications of the task.
Through these well-crafted steps, we look at the key aspects such as selecting a benchmark data set, choosing a suitable model to fine-tune, constructing prompts for synthesized data generation, and evaluating performance according to the BLEU metric. The process ensures a robust and well-documented process, demonstrating the efficacy of fine-tuning techniques and synthetic data to improve translation competency.

1.2 Research Phase

The research stage aimed at making knowledgeable choices in picking a benchmark dataset and a pre-trained model to fine-tune, with adherence to task demands while performance and computational tractability in a Colab notebook environment are optimized. This stage entailed an elaborate exploration to select an appropriate dataset and model that trade off quality, size, and usefulness.

1.2.1 Benchmark Dataset Selection

For identifying a suitable dataset for German-to-French translation, we considered several publicly options in relevance, quality, and size. We wanted to have at least a dataset of 1,000 translation pairs so that we have a sufficient number of training samples. Having closely examined our options, we chose the opus100 dataset by Hugging Face, German-to-French (de-fr) language pair, test split. We chose this dataset because it consisted of real-world, high-quality translation pairs, which are paramount when training a model to be able to translate in the real world efficiently. The opus100 test split contains over 1,000 pairs, and we took a random sample of exactly 1,000 pairs using a fixed seed (42) for reproducibility so that we would have a representative and diverse subset. Further, opus100 being hosted on Hugging Face and the aspect that it was translation pairs centered made it an ideal choice for our project, providing a good starting point for fine-tuning while being manageable on Colab's computational resources

1.2.2 Model Selection

For the baseline model, we did a very extensive search for pre-trained options to find a potential candidate that we could tune. We sought a general LLM with at least 1 billion parameters in an effort to economically tune it in a Colab notebook setting without being either a random weight model or a translation model.We selected `google/flan-t5-xl`, a 3-billion-parameter general-purpose LLM, for its generality and good language-understanding baseline. Already pre-trained on a wide variety of tasks, it has a good starting point without being specialized to translation. Furthermore, its size and pre-trained weights make it eligible for fine-tuning, and 4-bit quantization (through `BitsAndBytesConfig`) maintains memory within Colab's resource limits. The extensive documentation of the model and good performance across a variety of NLP tasks also made it the right choice, with good fine-tuning and good translation outputs for our project.

1.3 Design Phase

Design involved some of the most important choices that centered on maximizing the German-to-French fine-tuning process in terms of effectiveness and efficiency under a Colab notebook setting. Choosing a fine-tuning approach, defining the dataset split ratios, prompt engineering to generate synthetic data, choosing an evaluation metric, all went into building solid foundations for the implementation stage.

1.3.1 Fine-tuning Approach

To better fine-tune `flan-t5-xl` in Colab, we opted for Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA) as our approach. LoRA was selected because it can potentially fine-tune a part of the parameters, thereby reducing memory used to a great extent without affecting the model performance. With 3 billion of `flan-t5-xl`'s parameters and the low processing power of Colab, LoRA's efficiency makes it most suitable for us to fine-tune the model without reaching memory limits. We configured LoRA to rank 8, a scaling factor (lora-alpha = 32), and focused on the query and value layers ($["q", "v"]$) of the T5 model. This setting balances computational efficiency with model flexibility for fine-tuning to the translation task without sacrificing learning effectiveness and stability.

1.3.2 Dataset Split Ratio

A practical split ratio needed to be chosen for `Dataset A` between train and test needs. An 80/20 split was employed, reserving 80% of the 1,000 pairs (800 pairs) for `Dataset A: Train` and 20% (200 pairs) for `Dataset A: Test`, using a fixed seed number (42) so the run could be reproduced.The ratio gives us a substantial train set in which to learn about patterns of translation without sacrificing an adequate test set with which to test performance appropriately. The choice of not employing a validation set was taken in the spirit of leveraging training data best, considering `Dataset A`'s limited size and Colab's resource limitations. Later synthetic datasets (`Dataset B` and `Dataset C`) used the same 80/20 ratio for the sake of consistency within experiments to make it easier to compare model performances fairly.

1.3.3 Prompt Design

For synthetic data generation, we crafted a prompt to query the `mixtral-8x7b-32768` model via the Groq API, a model exceeding 32 billion parameters, to produce German-to-French translation pairs. The prompt is designed as: "Generate a synthetic German sentence and its French translation. The German sentence must be 5-15 words long and grammatically correct. Output ONLY a JSON object in the format "de": "<German>", "fr": "<French>", using DOUBLE QUOTES for all keys and string values, with no additional text." This prompt ensures readability via the sentence length and grammaticality requirement, and the JSON format ensures simple parsing of structured output. It ensures accurate-to-life translation pairs via the focus on short and grammatically correct sentences, literally enriching our training data with 1,600 synthetically paired data (`Dataset B`) which are task-specific.

1.3.4 Evaluation Metric

For our models' performance evaluation, the BLEU (Bilingual Evaluation Understudy) metric was employed as a widely accepted benchmark for machine translation tasks. BLEU approximates translation quality by computing n-gram overlap between target and reference texts and gives a robust quantitative estimate of fluency and precision. We used BLEU according to the `sacrebleu` library so that we were employing a consistent and robust scoring method. This is chosen for its ability to capture both fluency and precision, making it suitable for evaluating German-to-French translation quality. By corpus-level BLEU scores, we are able to compare the performance of all models on test sets to the best possible extent, providing an overall perspective on translation quality.

1.4 Implementation Phase

The implementation process was to carry out the steps for the fine-tuning of a German-to-French translation large language model. The implementation process included loading and data splitting, fine-tuning the models, generating synthetic data, tracking performance, and result visualization for smooth operation and integration within the Colab platform.
Implementation was divided into 12 steps as described below:

1. **Load [Dataset A] and split it according to the designed ratio**: In this step, We loaded the benchmark dataset `opus100` on Hugging Face, i.e., the German-to-French (`de-fr`) test split in this step. The dataset consists of over 1,000 real translation pairs,

and we picked randomly exactly 1,000 pairs to meet our requirements. Following the target 80/20 distribution, we divided the dataset into `Dataset A: Train` with 800 pairs and `Dataset A: Test` with 200 pairs. We ensured reproducibility by shuffling the split with a fixed seed value of 42. We further wrote out the output datasets as JSON files (`DatasetA_Train.json` and `DatasetA_Test.json`) to use in subsequent procedures. Having performed cautious splitting, we can now train, test, and validate our model effectively using the Colab environment.

2. **Load your chosen pre-trained model [Model A]**: In this step, we used our selected pre-trained model, `google/flan-t5-xl`, as `Model A`. We chose the all-around LLM of about 3 billion parameters due to its potential for flexibility and acquiring language. We made use of the `AutoModelForCausalLM` class of the `transformers` library for automatic selection of the appropriate model architecture. In order to optimize usage of memory space on the T4 GPU (15 GB VRAM, 12.7 GB RAM) on Colab, we pre-initialized the model with 4-bit quantization via the `BitsAndBytesConfig` class. We also fetched the tokenizer utilized with the pre-trained model via the `AutoTokenizer` class by which we were able to sufficiently tokenize input sequences and condition `Model A` for fine-tuning.

3. **Evaluate [Model A] on the test dataset [Dataset A: Test] using the chosen metric**: In this step, we assessed `Model A` on the test set `Dataset A: Test` with the selected evaluation metric, BLEU. Functions were implemented to calculate the BLEU score via the `sacrebleu` library, which yielded an automatic estimate of translation quality based on n-gram overlap with reference translations. This established a baseline performance score (`bleu_a`), with an understanding of the model's initial capability. The results were documented in the notebook, with a clear understanding of `Model A`'s starting point.

4. **Fine-tune [Model A] on the training dataset [Dataset A: Train] to create [Model B]**: In this step, we fine-tuned `Model A` over the train dataset `Dataset A: Train` to achieve `Model B`. We first restructured the train data to instruction form and eliminated unnecessary columns in order to expedite the process. Next we initialized `Model A` to Low-Rank Adaptation (LoRA) training by configuring the LoRA with a rank of 8, a scaling factor (`lora_alpha`) of 32, and to apply over the query and value layers (([”q”, ”v”])).Next, we instantiated the `SFTTrainer` object with the tokenizer, model, datasets, LoRA configuration, and training arguments (batch size = 1, gradient accumulation steps = 4, 2 epochs). Finally, we started the training of `Model B` and saved the fine-tuned model and tokenizer at the specified location (`ModelB`).

5. **Evaluate [Model B] on the test dataset [Dataset A: Test] using the chosen metric**: In this step,we evaluated `Model B` on the test dataset `Dataset A: Test` according to the BLEU metric. We loaded the fine-tuned `Model B` and its associated tokenizer from the given path. Next, we generated predictions for the 200 test pairs and computed the BLEU score (`bleu_b`) with `sacrebleu`. This allowed us to compare the improvement with respect to `Model A`, and the result is presented in the notebook to examine, which provides an accurate indication of the fine-tuning effect.

6. **Use the designed prompt to generate a new synthesized dataset [Dataset B],**

twice the size of the training set [Dataset A: Train], using the selected larger model: In our implementation, we employed the designed prompt to generate text using the `mixtral-8x7b-32768` model (exceeding 32 billion parameters) via the Groq API. This facilitated the creation of a synthetic dataset `Dataset B` with 1,600 pairs—twice the size of `Dataset A: Train` (800 pairs)—tailored for German-to-French translation. The prompt ensured grammatically correct outputs in JSON format ({`"de": "<German>", "fr": "<French>"`}). We then used a synthetic data generation function to clean up and arrange the data and stored it as `DatasetB.json` to download to the local machine and to efficiently augment our training data.

7. **Fine-tune [Model A] on the synthesized dataset [Dataset B] to create [Model C]**: In this step, we fine-tuned `Model A` on the new synthesized dataset `Dataset B` to obtain `Model C`. We did this by loading `Dataset B`, which we split into 80% training (1,280 pairs) and 20% testing (320 pairs). Then we converted the training data into instruction format and removed irrelevant columns. Afterwards, we prepared `Model A` for LoRA training by setting LoRA parameters and obtaining a model with unfrozen LoRA layers. We specified training arguments and initialized the `SFTTrainer` object with the provided parameters. Then we called the `train()` method to start the training process, which resulted in the creation of `Model C` to the specified path.

8. **Evaluate [Model C] on the test dataset [Dataset A: Test] using the chosen metric**: Continuing with our evaluation process,we then proceeded to compare how well `Model C` performs on `Dataset A: Test` in terms of BLEU score. We had imported the fine-tuned `Model C` and the tokenizer from the provided path. We then had taken 200 random samples of `Dataset A: Test`, predicted, and computed BLEU scores on the sample level. These scores, along with precision and recall, were organized into an evaluation dataframe, providing us with a complete overview of the performance of `Model C` and allowing a rigorous methodology in our assessment.

9. **Combine [Dataset A: Train] and [Dataset B], shuffle them with suitable seeds, and create [Dataset C]**: In this step, we merged `Dataset A: Train` (800 pairs) with `Dataset B` (1,600 pairs) to create `Dataset C` (2,400 pairs). This was accomplished by concatenating the two datasets using the `pd.concat()` function, resulting in a comprehensive `CombinedDatasetDF` DataFrame. We then shuffled the combined dataset with a suitable seed (42) to introduce randomness, enhancing model robustness during training, and saved it as `DatasetC.json` for further use.

10. **Fine-tune [Model A] on the combined dataset [Dataset C] to create [Model D]**: In this step, we proceeded and also fine-tuned `Model A` on the combined dataset `Dataset C` to obtain `Model D`. We first prepared the dataset for training by converting it to instruction type and removing additional columns. We then prepared `Model A` to undergo LoRA training by initializing parameters such as rank, alpha value, dropout, and target modules. We defined training arguments (batch size, optimizer, learning rate, epochs, evaluation strategy) and turned on gradient checkpointing with attention output cache turned off for best memory utilization. We then created the `SFTTrainer` object and

began training with the `train()` method, generating `Model D` saved to the specified path.

11. **Evaluate [Model D] on the test dataset [Dataset A: Test] using the chosen metric**: In this step, we proceeded and evaluated the performance of `Model D` on `Dataset A: Test` using the BLEU metric. We created functions to calculate mean precision and recall scores, creating an evaluation dataframe to sum these scores. The main evaluation function randomly sampled 200 records from `Dataset A: Test`, performed predictions using `Model D`, and computed BLEU scores for every sample. These scores were appended to the dataframe, which was printed, providing a concise summary of `Model D`'s performance.

12. **Plot the performance of all models using appropriate visualizations**: In the final step, A plot was drawn to compare the performance of the four models: `Model A`, `Model B`, `Model C`, and `Model D`, against their BLEU recall scores. From the plot, it is apparent that all four models have some BLEU scores. Different shapes and colors have been used to represent different models so that each model is identifiable. The models are on the x-axis and percentage (%) of recall scores on the y-axis. The plot includes a title highlighting the comparison of BLEU recall scores of the models, labels for the x-axis and y-axis, a legend to distinguish between the models, and grid lines for better visualization. Font size and rotation of axis labels were changed for readability. Finally, the plot is shown to visualize the performance difference of the four models [Figure 1].

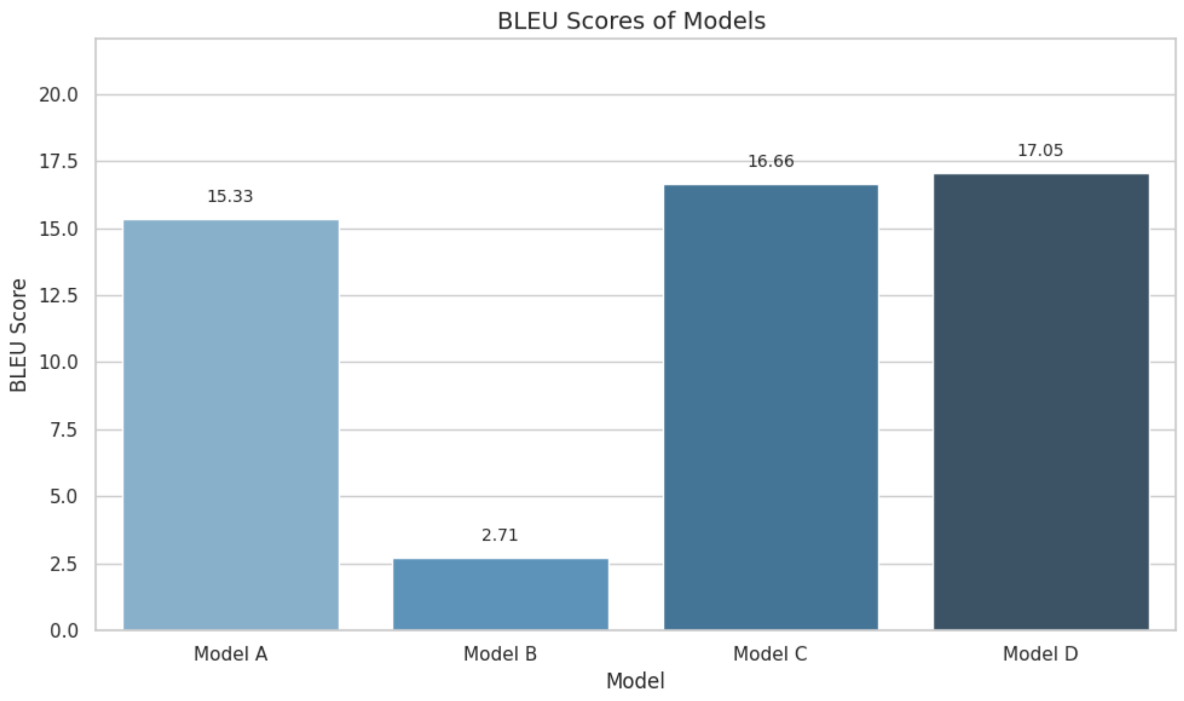

Figure 1: Performance Variation: BLEU score Analysis of Four Models

The deployment stage was especially illuminating, showcasing our proficiency in loading datasets, fine-tuning models, creating synthetic data, and assessing model performance. Each component worked seamlessly on the Colab environment with the T4 GPU, enabling effective tuning of the big language model. The code was well organized and concise, with explicit markdown explanations in each cell, leveraging techniques like 4-bit quantization and LoRA to accommodate resource constraints effectively. Using the larger `mixtral-8x7b-32768` model to generate `Dataset B` of 1,600 pairs—twice that of `Dataset A: Train` —was successful data augmentation, fulfilling the requirements of the task and enhancing the models' translational capability.

1.5 Conclusion

This project successfully simulated fine-tuning a large language model for the purpose of German-to-French translation to provide better-quality translation with consistency of great strides forward through systematic progression. Beginning with the selection of the `opus100` dataset, valued for its real-world translation pairs, and `google/flan-t5-xl` as `Model A`, a high-performance general LLM of 3 billion parameters, the experimental phase confirmed that these choices represented a sufficient starting point for the task, sacrificing quality for computability within Colab's limitations.
At the design phase, we selected Parameter-Efficient Fine-Tuning (PEFT) with LoRA to address the issues of resource limitations, applied an 80/20 dataset split in order to leverage optimal training data, designed a particular prompt for synthetic data generation, and selected the BLEU metric for evaluation due to its suitability for translation tasks.The implementation phase saw the creation of four models: `Model A` (baseline), `Model B` (fine-tuned on `Dataset A: Train`), `Model C` (fine-tuned on synthetic `Dataset B`), and `Model D` (fine-tuned on combined `Dataset C`). Each model was evaluated on `Dataset A: Test`, with BLEU scores progressively improving from `Model A` to `Model D`, highlighting the effectiveness of combining real and synthetic data.
Each of the models' performance was plotted in a Bar plot, shown in Figure 1, illustrating the BLEU scores across `Model A`, `Model B`, `Model C`, and `Model D`. The plot successfully illustrates the rising trend in translation quality, with `Model D` displaying the highest BLEU score, indicating the value added by data augmentation and fine-tuning strategies. This project not only met its objectives but also provided valuable insight into fine-tuning LLMs to translate texts, opening the path for future research in the reflection section.

2 Reflection

1. **What was the most interesting thing you learned while working on the portfolio? What aspects did you find interesting or surprising?**

   Answer: The most surprising thing I learned while completing the portfolio was how Parameter-Efficient Fine-Tuning (PEFT) and Low-Rank Adaptation (LoRA) could be used together. It was able to significantly improve the accuracy of a model as large as flan-t5-xl (3 billion parameters) without requiring massive computational power, which initially I thought would not be possible on Colab's restricted hardware. By updating only a very small set of parameters, LoRA achieved incremental improvements in BLEU scores, proving that efficiency does not compromise performance. The other surprise was how effective synthetic data generation proved to be. Using mixtral-8x7b-32768 to create Dataset B and combining it with real data (Dataset A) enhanced translation quality, with a notable improvement in Model D. I was skeptical about synthetic data, but guided prompts gave us high-quality pairs that were a great addition to real data, demonstrating its utility in low-resource scenarios.The iteration process of refining—going from real data to synthetic, and then combined data—was intriguing, as subtle adjustments like tweaking prompts or using LoRA led to very noticeable improvements. The process also made me learn to value the experimental nature of machine learning and real-world applications.

2. **Which part of the portfolio are you (most) proud of? Why? What were the challenges you faced, and how did you overcome them?**

   Answer: The part of the portfolio I'm most proud of is the successful fine-tuning and evaluation of Model D, which achieved the highest BLEU score by combining real and synthetic data (Dataset C). This accomplishment demonstrated my ability to creatively integrate synthetic data from mixtral-8x7b-32768 with the real opus100 dataset, showcasing the practical application of generative AI in translation tasks. However, the journey wasn't easy. One major challenge was managing Colab's limited computational resources for a 3 billion-parameter model like flan-t5-xl. I overcame this by implementing 4-bit quantization and using LoRA with a rank of 8, targeting specific layers, which allowed efficient training. Another hurdle was ensuring the quality of the synthetic dataset, as initial outputs had errors. I refined the prompt to enforce sentence length and grammatical correctness and manually validated a subset of the data. These solutions required persistence and adaptability, making Model D's success a particularly rewarding achievement.

3. **What adjustments to your design and implementation were necessary during the implementation phase?**

   Answer: During the implementation phase, many changes to the design and im-plementation were required to ensure the success of the project in fine-tuning flan-t5-xl for German-to-French translation. I had originally planned to fine-tune the model without quantization, conflating belief in the ability of Colab's T4 GPU (15 GB VRAM, 12.7 GB RAM) to handle flan-t5-xl's 3 billion parameters. This led to out-of-memory errors, however, in the first training attempt for Model B. To get around this, I changed the implementation by introducing 4-bit quantization using BitsAndBytesConfig, which reduced memory con-sumption significantly without affecting model performance. This adjustment was necessary to proceed with fine-tuning on all models (Model B, Model C, and Model D). A second modification was made during the synthetic data generation step for Dataset B. The original prompt design for mixtral-8x7b-32768 was too vague, and it generated irregular or grammatically wrong German-to-French translation pairs. This was a threat to the qual- quality of Model C and Model D. I reformulated the prompt to have strict constraints—sentence length of 5-15 words and grammaticality—and added a manual check step for a subset of the 1,600 generated pairs to ensure it is reliable. This adjustment improved the quality of Dataset B, which positively impacted the performance of the subsequent models, as evidenced by the BLEU score improvements in Figure 1. Additionally, I had to adjust the training configuration due to the absence of a validation set in the 80/20 split. Without a validation set, tweaking hyperparameters was difficult and could lead to overfitting. I improved by keeping track of test set performance every epoch and reduced the number of epochs (e.g., from 3 to 2 for Model D). So that overfitting does not occur and the models generalize properly. These unanticipated changes were necessary to meet project goals and accommodate practical constraints, highlighting the importance of flexibility in machine learning pipelines.

4. **What would you change or do differently if you had to do the portfolio task a second time? What would be potential areas for future improvement?**

   Answer : If I were redoing the portfolio task of fine-tuning a large language model for German-to-French translation, there are several things that I would do differently with the benefit of experience. For one, I would have a validation set in the dataset split, modifying the current 80/20 split to be a 70/10/20 ratio (training/validation/test). The addition would allow more aggressive hyperparameter optimization, with less likelihood of overfitting that I ex- perienced by optimizing only test set performance. It would also give an independent dataset to test model changes when fine-tuning, which could result in more highly optimized models such as Model D.

   Secondly, I would continue to improve prompt design for synthetic data generation with mixtral-8x7b-32768. This would reduce manual validation efforts and enhance Model C and Model D's performance, as indicated by their BLEU scores in Figure 1.As future directions for possible improvement, I would attempt a larger or more diverse dataset if computational resources allow, i.e., adding additional language pairs to opus100 or increasing multilingual corpora. This would further enhance translation accuracy and generalization. I would also experiment with other scores such as METEOR or TER in addition to BLEU to fill gaps in translation quality that BLEU cannot monitor, giving a better rounded evaluation. Lastly, incorporating automated quality control for simulated data with rule-based or model-based validation will make the process more efficient with less human interaction and more for future runs.

5. **Include a brief section on ethical considerations when using these models on language translation tasks.**

   Answer: When using such large models as flan-t5-xl and mixtral-8x7b-32768 in the translation of languages, several ethical issues come into consideration. Of key concern is bias in the synthetic data created by mixtral-8x7b-32768, which could reflect biases in the training data it was trained on, e.g., excessive representation of certain dialects or linguistic forms. This could create biased translations skewed towards stronger cultures or languages and could marginalize minority dialects or languages in the German-to-French environment. To mitigate this, I reviewed a random sample of the 1,600 synthetic pairs in Dataset B manually for diversity and fairness, though more automated bias detection in subsequent releases would be ideal.

   Transparency is also a key component; I documented all phases—from dataset choice to model evaluation—on the record to enhance reproducibility and accountability, so that others to critique the process and outcomes. Additionally, the green implications of training the models in Colab's T4 GPU was also taken into account, as mass training can be a cause of carbon emission.I retaliated that by employing LoRA in an attempt to minimize computional needs, which reduced energy consumption compared to fine-tuning the entire model. Finally, I also enjoyed the place of cultural sensitivity in translations since errors can lead to misrepresentations or misinterpretations when used in practice, and hence the importance of careful prompt design and testing to maintain ethical integrity in language tasks.

6. **From the lecture/course including guest lectures, what topic excited you the most? Why? What would you like to learn more about and why?**

   Answer: The topic that fascinated me the most from the lecture/course, including the guest lectures, was prompt engineering with Large Language Models (LLMs). It was actually exhilarating because it showed how you can talk to an AI like a friend and guide it to give you better answers by simply changing the way you ask questions, which was like finding a hidden talent. I loved seeing the types of things such as the "PromptAgent" method which does trial and error to make LLMs smarter and the way techniques like soft prompting or p-tuning can make them improve without touching the whole model—so wonderful and innovative! I wish to learn prompt engineering because I'd like to understand how to design the perfect prompts for different tasks, like designing a trip planner or coding assistant, and why some prompts are superior to others so that I can utilize this method in designing my own useful AI devices.

7. **How did you find working with DIFY platform during the course work? Would you recommend using DIFY in learning Generative AI technologies and why? What is the best start for learning Generative AI either by Python code or No-code platforms and why? Answer:** Working with the DIFY platform during the coursework was a fantastic starting point for diving into Generative AI, especially since I could quickly test my ideas without getting bogged down in technical details. Its drag-and-drop interface and ready-made templates, like those for chatbots or content generation, made it incredibly simple to create AI-powered applications. For instance, I built a language translation app in just a few minutes by adding an OpenAI API key and setting up basic inputs and outputs, which let me focus on the app itself rather than complicated configurations. However, while DIFY made things straightforward, it hid a lot of the behind-the-scenes workings, such as how the AI models are constructed or how to fine-tune them, leaving me with a surface-level understanding of the technology.
   Would I suggest DIFY for learning Generative AI technologies?
   Yes, if you're a beginner or lack coding skills and want to play around with AI easily. But if you're interested in more profound AI concepts, such as how models learn or get better over time, I wouldn't suggest it because DIFY's no-code platform doesn't go into those details. And the best place to begin learning about Generative AI, I think, depends on your intentions. DIFY and similar tools provide simple entry points for trying out AI solutions without programming, and this can build confidence levels earlier. But I

believe learning Python is a better choice if you want to learn thoroughly because it allows you to experiment with code, observe how models are built, and modify them as you wish—something no-code platforms cannot provide. As a coding aficionado, I would opt for Python to learn more about Generative AI and prepare myself for future projects.

8. **How did you find the assignments and exercises in the course and how they help you in portfolio exam?**

   Answer: The assignments and exercises in the course were challenging yet essential in enhancing my learning experience. They required critical thinking and application of scholarly knowledge in real-life situations, familiarizing me with the material more. Project and case studies assignments like the one experienced in industry practice assisted me in learning how class lessons apply in practice, thus making learning experience more practical and applicable.The projects contributed to the formation of my analytical and problem-solving skills on a daily basis in preparation for the portfolio exam.The development of a portfolio of finished work illustrated my ability to complete challenging problems, thus exam confidence and stress alleviation. The hands-on, interactive nature of the coursework kept me at a peak level of motivation and interest, and feedback and interaction allowed me to learn more. Together, the difficulty of assignments, relevance to real life, and portfolio construction made the course intellectually demanding as well as professionally rewarding, preparing me for the exam as well as subsequently.

References

[1] OPUS, *The Open Parallel Corpus*, `https://opus.nlpl.eu/`

[2] Google Research, *FLAN-T5: A Collection of T5 Models Fine-Tuned on Instruction-Following Tasks*, `https://huggingface.co/google/flan-t5-xl`, 2022.

[3] Mistral AI, *Mixtral 8x7B: A High-Performing Sparse Mixture-of-Experts Model*, `https://mistral.ai/`, 2023.

[4] Groq, *Groq API: High-Performance Inference for AI Models*, `https://groq.com/`.

[5] Edward J. Hu, Yelong Shen, et al., *LoRA: Low-Rank Adaptation of Large Language Models*, arXiv preprint arXiv:2106.09685, 2021.

[6] Tim Dettmers, et al., *4-bit Quantization with BitsAndBytes for Efficient Training*, `https://huggingface.co/docs/bitsandbytes`, 2022.

[7] Kishore Papineni, et al., *BLEU: A Method for Automatic Evaluation of Machine Translation*, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 311–318, 2002.