Fine-tune LLM for Code Generation

Prateek Kumar Sharma
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 2023-24. This report is an original work and will be scrutinised for plagiarism and potential LLM use.*

1 Portfolio documentation

1.1 Introduction

Welcome to the documentation of our project on fine-tuning a large language model (LLM) for code generation. In this project, we aim to demonstrate the effectiveness of fine-tuning LLMs for specific tasks, particularly in the context of Python code generation.

Fine-tuning LLM for code generation involves refining pre-trained language models like GPT to specialize in programming tasks. Through exposure to code snippets and descriptions, the model learns syntax, semantics, and context, enabling it to generate accurate code.

Through a series of carefully planned phases, we will explore the process of selecting a benchmark dataset, choosing an appropriate model for fine-tuning, designing prompts for synthesized data generation, and evaluating model performance using suitable metrics.

1.2 Research Phase

In this phase, we conducted thorough research to make informed decisions regarding the choice of dataset and model for our project.

1.2.1 Benchmark Dataset Selection

We began by evaluating various benchmark datasets suitable for Python code generation tasks. After careful consideration, we selected codeparrot/apps for its comprehensive coverage of Python code examples and clear instructions for code generation.

The **"APPS" benchmark** comprises 10,000 coding problems designed for assessing the proficiency of language models in generating code based on natural language specifications. [1]

We wanted a dataset that had lots of different examples of Python code. This would help our model learn different ways of coding. We also needed the dataset to be focused on generating code. It should have pairs of inputs (like descriptions) and their matching code outputs.

It was important for the dataset to be big enough so our model could learn from a lot of examples. This would help it understand many different coding styles and rules.

We chose the codeparrot/apps Dataset from Hugging Face because it met all these needs well. It has a wide range of input-output pairs for generating Python code. With over 10,000 examples, it covers many coding situations, from different programming concepts to various types of programs. Also, because it's from Hugging Face, it's easy for us to use in our project.

We picked this dataset after looking at many others because it's **big, diverse, and perfect** for teaching our model to generate accurate code for real-world situations.

1.2.2 Model Selection

The choice of a suitable pre-trained model to serve as the foundation for fine-tuning is essential to our Project. Following a comprehensive analysis, we determined that the **Microsoft/phi-1_5 model** was the best fit for our purposes. The Microsoft/phi-1_5 model, which is well-known for its adaptability and strong performance in a range of NLP applications, offers a reliable foundation for our code generation studies.

The language model Phi-1.5 is a Transformer with **1.3 billion parameters**. It was trained using the same data sources as phi-1, augmented with a new data source that consists of various NLP synthetic texts. When assessed against benchmarks testing common sense, language understanding, and logical reasoning, Phi-1.5 demonstrates a nearly state-of-the-art performance among models with less than 10 billion parameters. [2]

We chose the Microsoft/phi-1_5 model because it's been trained on a lot of different text data and has a good reputation. By using its built-in language understanding, we aim to speed up our fine-tuning process and get better results. Also, its pre-trained weights and detailed documentation make it easy to integrate into our experiments.

1.3 Design Phase

With our dataset and model chosen, we moved on to the design phase, where we made critical decisions regarding fine-tuning approach, dataset split ratio, prompt design, and evaluation metric.

1.3.1 Fine-tuning Approach

We carefully considered the available computational resources and selected a fine-tuning approach that maximizes model performance while remaining feasible within our constraints.

A methodology known as PEFT (Prompt Engineering and Fine-Tuning) with LORA (Learning Over Random Answering) combines specific prompts to direct the behavior of the model with a fine-tuning strategy that focuses on learning patterns beyond chance. It entails creating customized cues to guide the model's answers toward intended outputs and honing the

model through methods that put learning significant patterns ahead of memorizing random noise. The goal of this comprehensive strategy is to improve language models' robustness and performance in a variety of activities and domains.

Phase 1: Prepare Model for LoRA To begin, we prepare the model for fine-tuning with LoRA. LoRA (Learned ReLU Activation) is a technique that enhances the training stability of language models. We apply LoRA to our model by modifying its architecture to include LoRA-specific configurations and parameters.

Phase 2: Define Training Parameters Next, we define the parameters for training the model. These parameters include settings such as the batch size, learning rate, and number of training epochs. These settings determine how the model learns from the training data and how its performance is optimized during training.

Phase 3: Set Up Trainer Once the model is prepared and training parameters are defined, we set up the trainer to execute the fine-tuning process. The trainer manages the training procedure, including data preprocessing, model training, and evaluation. It ensures that the model is trained effectively and efficiently for our specific task.

1.3.2 Dataset Split Ratio

Determining the split ratio between training, testing datasets and validation datasets is crucial for model evaluation. The dataset split ratio determines how the available data is divided into training and testing sets for model evaluation. It's crucial to strike a balance between having enough data for training to learn from and having enough data for testing to accurately assess the model's performance.

Common ratios include 70/30 or 80/20, where 70% or 80% of the data is used for training and the remaining 30% or 20% for testing. A larger training set can lead to better model performance but may increase the risk of overfitting, while a larger testing set provides a more reliable estimate of the model's generalization ability. The chosen ratio should reflect the size of the dataset, the complexity of the task, and computational resources available.

After deliberation, we settled on a [**60/20/20**] split ratio to ensure an adequate balance between **training, testing and validation data.**

1.3.3 Prompt Design

Designing effective prompts for synthesized data generation is essential for enhancing model performance. We developed prompts that provide clear instructions for code generation while promoting diversity in the generated data. While Desiginig the Prompt, We follow some Guidelines:

- **Included details in query to get more relevant answers -** This ensures that the prompts generated address the specific needs.

- **Ask the model to adopt a persona -** By adopting a persona, the responses to the prompts can be tailored to reflect a particular perspective or style, enhancing the authenticity and effectiveness of the communication.
- **Used delimiters to clearly indicate distinct parts of the input -** Clear delimiters help organize the prompt into manageable sections, facilitating understanding and interpretation by the model, which can lead to more coherent and focused responses.
- **Specify the steps required to complete a task -** Providing clear steps helps break down complex tasks into manageable components, guiding Model in their approach and leading to more structured and systematic responses.
- **Provided example -** Including examples helps illustrate the expectations for the response, offering a reference point for model to understand the task and criteria for evaluation, which can improve the quality and relevance.
- **Specified the desired length of the output -** Setting a desired length helps manage expectations and ensure that responses are neither too brief nor overly verbose, promoting conciseness and clarity in communication while covering the necessary depth of analysis.
- **Provided Context -** Offering context helps situate the prompt within the broader framework of the course or subject matter, enabling model to understand the relevance and significance of the task, which can enhance their motivation and engagement with the material.
- **Included Guidelines -** Providing guidelines outlines the expectations for the response in terms of format, content, and criteria for evaluation.

1.3.4 Evaluation Metric

Selecting an appropriate evaluation metric is vital for accurately assessing model performance. After careful consideration, we chose ROUGE Metrics for its ability to capture the quality and relevance of generated code.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metrics are widely used in natural language processing tasks, particularly in evaluating text summarization systems. They measure the quality of generated text summaries by comparing them to reference or ground truth summaries. Here's a simplified explanation of ROUGE metrics:

1. ROUGE-N measures how much the system-generated summary overlaps with the reference summary in terms of n-grams, which are sequences of n words.

   For example, ROUGE-1 looks at how many single words (unigrams) are shared between the system and reference summaries, while ROUGE-2 focuses on shared pairs of words (bigrams).

2. ROUGE-L assesses similarity based on the longest common subsequence (LCS) between the system and reference summaries. This method considers the sentence structure and identifies the longest sequences of words that appear in both summaries.

In summary, ROUGE metrics provide quantitative measures of how well a generated summary matches a reference summary. They are essential tools for evaluating the performance of text summarization systems and guiding improvements in their effectiveness.

1.4 Implementation Phase

The Most Interesting Phase i.e Implementation phase, we divided the Implementation phase in Following Steps:

1. **Load the dataset benchmark and split it in the designed ratio :** In this step, we loaded the benchmark dataset "codeparrot/apps" and split it into training, testing, and validation sets according to the designed ratio. The dataset contains code snippets from various applications, and we ensured reproducibility by shuffling the training split using a seed value of 42. Then, we selected a subset of 500 samples from the shuffled dataset and divided it into training, testing, and validation sets with 300, 100, and 100 samples, respectively. This allows us to train, test, and validate our model effectively.

2. **Load the Microsoft Phi-1.5 Pre-trained model [Model A]:** In this step, we loaded the selected pre-trained model [Model A], which is the Microsoft Phi-1.5 model, for causal language modeling. We used the AutoModelForCausalLM class from the transformers library to automatically select the appropriate model architecture. Additionally, we configured the model for quantization using the BitsAndBytesConfig class to improve memory efficiency. The tokenizer associated with the pre-trained model was also loaded using the AutoTokenizer class, allowing us to tokenize input sequences appropriately.

3. **Test model A on the testing dataset using the selected evaluation metric :** In this step, we tested Model A on the testing dataset using the selected evaluation metric, which is the ROUGE score. We implemented functions to calculate the mean precision and recall scores of the model in terms of Rouge1, Rouge2, and RougeL. Then, we created an evaluation dataframe containing these scores and displayed the results for analysis. The evaluation dataframe provides insights into the model's performance in terms of precision and recall for each Rouge metric.

4. **Fine-tune model A on the training dataset [Model B] :** In this step, we fine-tuned model A on the training dataset to obtain model B. We first formatted the training data into instruction format and removed unnecessary columns. Then, we prepared model A for Low-Rank Adaptation (LoRA) training and defined the LoRA configuration. Next, we initialized the SFTTrainer object with the required parameters, including the tokenizer, model, datasets, LoRA configuration, and training arguments. Finally, we started the training process for model B and saved the fine-tuned model and tokenizer to the specified path.

5. **Test model B on the testing dataset using the selected evaluation metric :** In this step, we tested model B on the testing dataset using the selected evaluation metric, which includes Rouge1, Rouge2, and RougeL scores. We loaded the fine-tuned model B and its associated tokenizer. Then, we generated predictions for a sampled subset of the test dataset using model B. Next, we calculated Rouge scores for the generated predictions and displayed the mean precision and recall scores for Rouge1, Rouge2, and RougeL.

6. **Use the designed prompt to generate a new synthesised dataset that has the nature and 3 times the size of the training dataset using the supported AWS model :** In our implementation, we employed the generate function to create text utilizing a designated API endpoint paired with a tailored prompt. This facilitated the generation of synthetic data pertinent to coding questions. Subsequently, we utilized the synthetic_data_generation function to generate synthetic datasets, drawing from a predetermined list of topics associated with coding inquiries. Once the data was generated, we employed the data_cleaning function to refine and organize it, subsequently appending it to a structured DataFrame. Lastly, to ensure accessibility and further analysis, the synthesized dataset was saved as a CSV file and made available for download onto the local machine.

7. **Fine-tune model A on the new synthesised dataset [Model C] :**In this step, we fine-tune model A on a newly synthesized dataset to obtain model C. The process involves loading the synthesized dataset, formatting it into instruction format, and then training the model using the Low-Rank Adaptation (LoRA) technique.

    - Load the synthesized dataset: First, we import the necessary libraries for file upload and specify the file path for the synthesized dataset. Then, we read the dataset into a pandas DataFrame.

    - Format instruction for generating code: Next, we define a function to format the instruction for generating code. This function takes the problem statement/question and the generated code solutions as inputs and returns a formatted instruction.

    - Process dataset:We define a function to process the dataset by converting it to instruction format and removing unnecessary columns. This function maps the format_instructionSynthetic function to each sample in the dataset, converts each sample to instruction format, and stores the result in a new column "Code". Then, it removes unnecessary columns from the dataset.

    - Prepare training, testing, and validation datasets: We process the training, testing, and validation datasets by converting them to instruction format and removing unnecessary columns. We then select the first 900 samples from the synthesized data for training, and the first 100 samples each from the test and validation datasets for evaluation.

    - Fine-tune model A using LoRA: We prepare model A for Low-Rank Adaptation

(LoRA) training by configuring LoRA parameters and obtaining a model with unfrozen LoRA layers. We then define the training arguments for the model.

- Train model C: Finally, we initialize the SFTTrainer object with the specified parameters and start the training process using the train() method of the trainer object.

This completes the fine-tuning of model A on the new synthesized dataset, resulting in the creation of model C.

8. **Test model C on the testing dataset using the selected evaluation metric :** Continuing with our evaluation process, we now focus on assessing the effectiveness of Model C on our test dataset using Rouge1, Rouge2, and RougeL scores as our metrics. Initially, we retrieve and load the fine-tuned Model C along with its associated tokenizer from the specified path. Next, we randomly select 100 samples from our test dataset to conduct evaluations. For each of these samples, Model C generates predictions, which we then use to compute Rouge scores, incorporating precision and recall, across Rouge1, Rouge2, and RougeL metrics. These scores are crucial indicators of the model's accuracy and completeness in its responses. To consolidate our findings, we organize these scores into an evaluation dataframe, providing a comprehensive overview of Model C's performance across the different Rouge metrics, thus maintaining a meticulous approach to our assessment process.

9. **Combine the training dataset and the synthesised dataset and shuffle them with suitable seeds :** In this step, we merge the training dataset with the synthesised dataset to create a combined dataset. his is accomplished by concatenating the two DataFrames using the pd.concat() function. The resulting CombinedDatasetDF DataFrame contains both the training and synthesised data, ensuring a comprehensive dataset for model training. Finally, we shuffle the combined dataset with suitable seeds to introduce randomness and enhance model robustness during training.

10. **Fine-tune model A on the new Combined dataset [Model D] :** In this Step, we proceed to fine-tune Model A on the new combined dataset, denoted as Model D. The process involves several steps:

    - Firstly, we format the dataset for training by converting it to instruction format and removing unnecessary columns. This ensures that the data is in a suitable format for model training.
    - Next, we prepare Model A for Low-Rank Adaptation (LoRA) training by configuring LoRA parameters such as rank, alpha value, dropout, and target modules. This step is crucial for enhancing the model's performance during training.
    - Then, we define the training arguments for the model, specifying parameters related to batch size, optimizer, learning rate, epochs, and evaluation strategy.
    - Afterward, we enable gradient checkpointing and disable attention output cache to optimize memory usage and ensure efficient training.

- Subsequently, we initialize the SFTTrainer object with the tokenizer, model, datasets, LoRA configuration, and training arguments. This object orchestrates the training process, utilizing the specified configurations.

Finally, we commence the training process using the train() method of the trainer object, allowing Model A to undergo fine-tuning on the combined dataset to obtain Model D. This iterative process aims to enhance the model's performance and adapt it to the specific characteristics of the combined dataset, ultimately improving its effectiveness in generating code solutions.

11. **Test model D on the testing dataset using the selected evaluation metric :** In this step, we proceed with evaluating the performance of Model D on the testing dataset using the selected evaluation metric, encompassing Rouge1, Rouge2, and RougeL scores. The evaluation process involves several steps: First, functions are defined to calculate mean precision and recall scores for each Rouge metric. Then, an evaluation dataframe is constructed to consolidate these scores for analysis. The main evaluation function randomly selects 100 records from the test dataset, generates predictions using Model D, and computes Rouge scores for each instance. These scores, including both precision and recall, are appended to the dataframe. Finally, the evaluation dataframe is displayed, providing a concise summary of Model D's performance across the Rouge metrics.

12. **Plotted the visualisation to show all model's performance :** In the final step, a visualization is created to illustrate the performance of all four models: Model A, Model B, Model C, and Model D, based on their ROUGE recall scores. The plot displays ROUGE-1, ROUGE-2, and ROUGE-L recall scores for each model. Each model is represented by a different marker shape and color for clarity. The x-axis represents the models, while the y-axis represents the percentage (%) of recall scores. The plot includes a title highlighting the analysis of ROUGE recall scores across the models. Additionally, labels for the x-axis and y-axis are provided, along with a legend to distinguish between the different ROUGE metrics. Grid lines are displayed for better visualization, and the fontsize and rotation of the axis labels are adjusted for readability. Finally, the plot is shown to visualize the performance variation of the four models [Figure 1].

1.5 Conclusion

We have carefully documented our process of optimizing a large language model (LLM) for Python code generation throughout this documentation. We started with a thorough research process and then carefully chosen the codeparrot/apps dataset based on its relevance and diversity to our work. The Microsoft/phi-1_5 model was selected due to its well-known agility and robust performance in a wide range of natural language processing applications.

As we entered the design phase, we decided on important things including the evaluation metric, dataset split ratio, prompt design, and fine-tuning strategy. These choices set the stage for our implementation phase, in which we carried out a number of actions to refine our models (Models B, C, and D) and evaluate their effectiveness using ROUGE metrics.

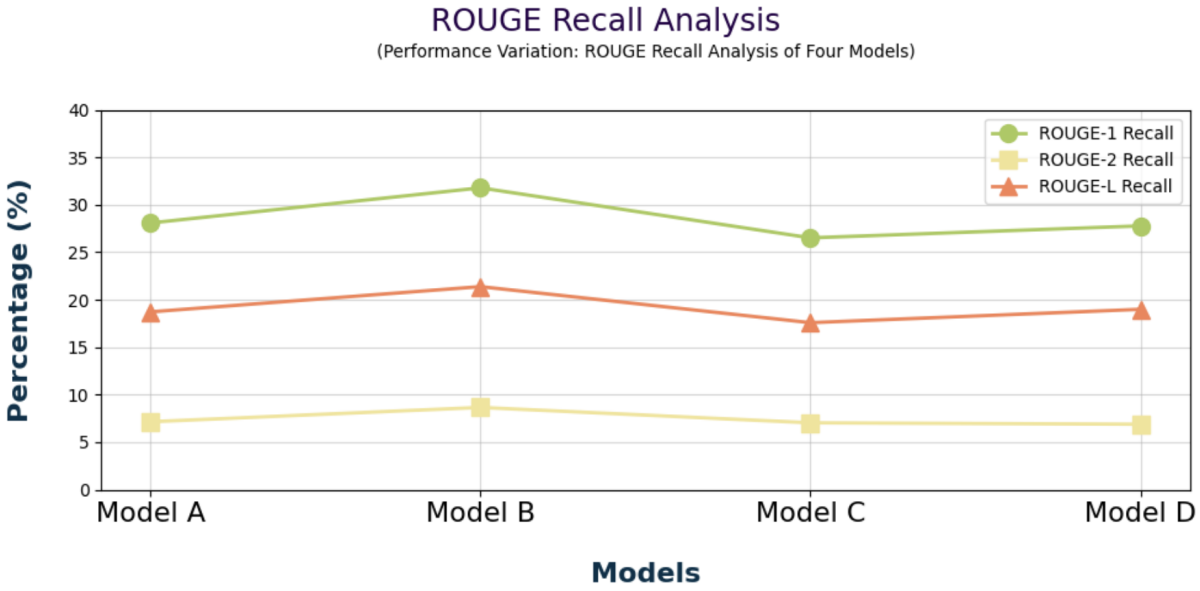

Figure 1: Performance Variation: ROUGE Recall Analysis of Four Models

The implementation step proved to be especially illuminating as it showcased our proficiency in loading datasets, optimizing models, producing synthetic data, and assessing model performance. Because every stage was carefully carried out, models that were gradually improved and customized for our particular mission were produced.

Our assessment procedure ultimately produced a thorough visualization that showed how well each of the four models performed across ROUGE recall ratings. Their ability to produce precise code solutions may be clearly compared thanks to this visual representation.

In conclusion, this documentation shows the effectiveness of improving LLMs for code generation work, focusing our systematic approach, careful execution, and thorough assessment procedure. It opens the way for more developments in this field and provides evidence of the promise of natural language programming.

2 Reflection

2.1 Introduction :

Language models represent a groundbreaking advancement in our digital world, extending far beyond deep learning, AI, and computer science into nearly every aspect of computer usage. This widespread application has drawn unprecedented attention to this field. It's now a vital part of our daily lives, with its increasing accessibility and user-friendly nature sparking interest in computer science and, by extension, programming. What used to be a daunting task, involving memorizing countless functions, syntax rules, and exceptions, can now be accom-

plished with just a few lines of context.

The main goal of this project was to develop and refine models capable of generating high-quality solutions to coding challenges. By harnessing pre-existing language models and cutting-edge techniques, our aim was to produce models that not only accurately understand coding problems but also produce solutions that are both grammatically correct and logically sound.

2.2 Most Interesting Learning:

The most interesting thing I Learned while working on this Project is the Prompt Engineering. Prompt engineering involves designing and crafting prompts or input queries that effectively guide language models to produce desired outputs. Prompt engineering is an important factor in determining how well language models function and behave in a variety of natural language processing tasks by optimizing prompts. As I learned During the lecture that English is Going to be the New Coding Langauge. After Completeing this Course and Gaining Indepth Knowledge about LLM's, I feel that Prompt Engineering is Very Important skill In Recent times. I Enjoyed Learning this Skill.

2.3 Pride and Challenges:

I Enjoyed Almost Every Phase of the Portfolio From Research Phase where I Study so many Datasets and Models to Analysing the Results and Efficiency of Models but I am most proud of the Complete implementation phase of the portfolio, particularly the successful fine-tuning of the pre-trained model and the integration of all components within the Colab notebook environment.

I still Remember the Happiness When My Last Model Get Fined Tuned Successully. However, this achievement did not come without its challenges. One of the main obstacles I faced was ensuring the efficiency and cleanliness of the code while maintaining its functionality. To overcome this challenge, I dedicated time to thoroughly documenting each step of the implementation process, ensuring clarity and ease of understanding for anyone reviewing the code.

2.4 Adjustments and Future Changes:

During the implementation phase, adjustments to the design and implementation were necessary to optimize performance and address unforeseen issues. For example, fine-tuning the model on the synthesised dataset required careful consideration of the dataset's size and nature to ensure optimal training outcomes. If I were to repeat the portfolio task, I would pay closer attention to the selection of prompts for generating synthesised data, as well as explore additional techniques for data augmentation to further enhance model performance.

I also I changed My Selected Pre-Trained Model During Implementation, I selected 7 Billion Paramenters Model, So that it will give better results but During Implementataion, By taking Computation Limit and Memory Limit into Consideration we have available on google colab, I

have to change My Model to 1.3 Billion Parameters Model. If I have to doing this type of task again, then I would Pay More Attention to So many things While Designing the Complete Process, becasue Now, I have gone through the Whole Process of implementation. So Now I Know things From the Practical Lens.

In Future, I would like to train My Model on More Data To Increase Efficiency and Accuracy of My Model. Also, I would like Experiment More and invest Little more time to Designed my Prompt For Synthesis Data Generation.

2.5 Ethical Considerations:

Ethical considerations play a crucial role in the use of language models for code generation tasks. The use of language models for code generation activities requires careful consideration of ethical issues. The possibility of bias in the generated code is one major worry since it has the potential to maintain already-existing differences and inequality in the software development community. Bias mitigation techniques are necessary to solve this problem and guarantee just and equitable results. These strategies include careful selection and preparation of training data as well as routine auditing of model outputs.

Transparency and accountability are also important principles when using code generation models. It is important that users and stakeholders maintain an in-depth understanding of the workings of these models, including any potential biases and limitations. Users can understand and correct any mistakes or biases in the resulting code when the model's architecture, training data, and decision-making process are transparently documented. This promotes trust and accountability. Developers may ethically leverage the power of code generation models, maximizing their advantages while reducing possible harm to persons and society, by giving priority to certain ethical issues.

2.6 Exciting Topics from the Course:

I feel that to Me Each Class, Session, Slides and Topics Excited me Because this is a Hot Topic in Indusrty and Market Right Now. From the lectures and course materials, one topic that excited me the most was the Fine-Tuning of Pre-Trained Models For Specific Task.

Seeing the transformational potential of modifying a pre-trained language model to perform well in a selected domain is one of the most thrilling parts of fine-tuning a model for a particular purpose. By means of fine-tuning, the model transforms from a generalist to a specialist, able to understand and generate outputs that are contextually relevant and customized to the particular needs of the task at hand. This procedure not only demonstrates the adaptability and flexibility of contemporary AI models, but it also highlights the possibility of utilizing current resources to address challenging real-world issues with never-before-seen accuracy and efficiency.

2.7 Multiple Choice Questions:

1. Question: In the context of large language models (LLMs), what role does model size primarily play?

   a) Determining the model's inference speed

   b) Influencing the model's capacity to learn complex patterns

   c) Regulating the model's memory usage

   d) Controlling the model's training timeCorrect

   Answer: b) Influencing the model's capacity to learn complex patterns.

2. Question: Which of the following best describes the purpose of prompt engineering in natural language processing models?

   a) Enhancing the model's training data

   b) Tuning hyperparameters for better performance

   c) Crafting specialized inputs to guide model behavior

   d) Improving model inference speedCorrect

   Answer: c) Crafting specialized inputs to guide model behavior

3. Question: When finetuning a pre-trained language model, what aspect is crucial to consider in order to prevent overfitting?

   a) Increasing the size of the training dataset

   b) Decreasing the learning rate

   c) Adding more layers to the model

   d) Using a smaller batch size during training

   Answer: d) Using a smaller batch size during training

3 References

1. https://huggingface.co/datasets/codeparrot/apps
2. https://huggingface.co/microsoft/phi-1_5?text=My+name+is+Lewis+and+I+like+to
3. https://arxiv.org/abs/2309.05463

Informed Consent of Participation

You are invited to participate in the field study **LLM Education** initiated and conducted by Applied Machine Learning group. The research is supervised by **Sebastian J Vollmer**. Please note:

- Your participation is entirely voluntary and can be withdrawn at any time.
- The field study will last approximately 6 weeks.
- We will record the documentation submitted by the student at the end of the portfolio exam.
- All records and data will be subject to standard data use policies.

If you have any questions or complaints about the whole informed consent process please contact Sebastian J Vollmer (E-Mail: sebastian.vollmer@dfki.de).

Purpose and Goal of this Research

Ability to create multiple choice questions of LLMs. Can we prompt an LLM to create good multiple choice questions? Your participation will help us achieve this goal. The results of this research may be presented at scientific or professional meetings or published in scientific proceedings and journals.

Participation and Compensation

Your participation in this study is completely voluntary and is unpaid.

Procedure

After confirming the informed consent the procedure is as follows:

1. Student submits the exam with deliverables
2. Student also creates questions of their choice based on instructions of good multiple choice questions.
3. We compare LLM's output to MCQs created by students.

The complete procedure of this field study will last approximately 6 weeks.

Risks and Benefits

There are no risks associated with this field study. We hope that the information obtained from your participation may help to bring forward the research in this field. The confirmation of participation in this study can be obtained directly from the researchers.

Data Protection and Confidentiality

We are planning to publish our results from this and other sessions in scientific articles or other media. These publications will neither include your name nor cannot be associated with your identity. Any demographic information will be published anonymized and in aggregated form. Contact details (such as e-mails) can be used to track potential infection chains or to send you further details about the research. Your contact details will not be passed on to other third parties. Any data or information obtained in this field study will be treated confidentially, will be saved encrypted, and cannot be viewed by anyone outside this research project unless we have you sign a separate permission form allowing us to use them. All data you provide in this field study will be subject of the General Data Protection Regulation (GDPR) of the European Union (EU) and treated in compliance with the GDPR. Faculty and administrators from the campus will not have access to raw data or transcripts. This precaution will prevent your individual comments from having any negative repercussions. During the study, we log experimental data, and take notes during the field study. Raw data and material will be retained securely and compliance with the GDPR, for no longer than necessary or if you contact the researchers to destroy or delete them immediately. As with any publication or online-related activity, the risk of a breach of confidentiality or anonymity is always possible. According to the GDPR, the researchers will inform the participant if a breach of confidential data was detected.

Identification of Investigators

If you have any questions or concerns about the research, please feel free to contact:
Sebastian J Vollmer
Principal Investigator Trippstadter Str. 122
67663 Kaiserslautern, Germany sebastian.vollmer@dfki.de

☒ I understand the explanation provided to me. I understand and will follow the hygiene rules of the institution. I understand that this declaration of consent is revocable at any time. I have been given a copy of this form. I have had all my questions answered to my satisfaction, and I voluntarily agree to participate in this field study.

☒ I agree that the researchers will and take notes during the field study. I understand that all data will be treated confidentially and in compliance with the GDPR. I understand that the material will be anonymized and cannot be associated with my name. I understand that full anonymity cannot be guaranteed and a breach of confidentiality is always possible. From the consent of publication, I cannot derive any rights (such as any explicit acknowledgment,

financial benefit, or co-authorship). I understand that the material can be published worldwide and may be the subject of a press release linked to social media or other promotional activities. Before publication, I can revoke my consent at any time. Once the material has been committed to publication it will not be possible to revoke the consent.

Signature

RPTU Kaiserslautern, 29-04-24

Place, Date