Coding Assistant using Phi-2

Georgios Charalambous[1], Georgios Chatzichristodoulou[2], Antonios Kritikos[3], and Panagiotis Papastathis[4]

[1]AM: 03119706, geo2000ch10@gmail.com
[2]AM: 03120125, gioxatz11@gmail.com
[3]AM: 03120025, ankrit400@gmail.com
[4]AM: 03120111, pan.papastathis@gmail.com

ABSTRACT

Phi-2 is a small language model (SLM) developed by Microsoft that achieves remarkable results despite having only 2.7 billion parameters. In this work, the quantization and fine-tuning of Phi-2 is investigated in detail in order to develop more efficient versions while preserving the capabilities of the original model, especially its coding skills. The fine-tuned models were evaluated against academic benchmarks, including commonsense-reasoning, mathematics and coding tasks, and showed performance improvements after fine-tuning. A new offline mobile application called 'Sophia' is introduced, which uses Phi-2 as a coding assistant and is able to generate and summarize code.

Introduction

In recent years, many attempts have been made to democratize the resource intensive training and inference of language models. One such attempt are small language models (SLMs), such as Phi-2, developed by Microsoft. Phi-2 comprises 2.7 billion parameters and has a size of 5.5GB. It is built on a Transformer architecture with a next-word prediction objective. The model has been designed for tasks such as question answering, chat interactions and coding, and exhibits advanced reasoning and language understanding capabilities. It should be noted that its impressive performance is on a par with that of models with up to 25 times more parameters. The development of Phi-2 has been based on the preceding Phi-1.5 model. This involved training over a period of 14 days using 96 A100 GPUs. The training dataset consisted of a combination of Python code, Q&A content from StackOverflow, competition code, synthetic NLP text, and filtered web content that was selected for its educational value and quality. In this work, we explore the benefits and limitations of Phi-2 and its variants. After quantization and fine-tuning on a number of coding datasets, the derivatives of Phi-2 were evaluated against academic benchmarks, including those assessing commonsense reasoning, mathematics, and coding tasks. In order to obtain a more reliable evaluation, these benchmarks were compared to Microsoft's respective scores. Finally, an offline coding assistant was developed in the form of an Android application.

Preliminary comments

It is essential to adopt an objective methodology that provides a reliable foundation for assessing the performance and capabilities of the model variants. For the purposes of this report, the same example will be used for the models' inference, allowing for a comparison of the responses given by the different models. The prompt provided is as follows:

```
Give me a program that checks if a number is prime.
```

```
*** Generate:
Instruct: Write a code to select check if a number is prime
Output:
#include<stdio.h>

int main()
{
    int n, i;
    printf("Enter a number: ");
    scanf("%d", &n);

    for (i = 2; i < n; i++)
    {
        if (n % i == 0)
        {
            printf("%d is not a prime number", n);
            break;
        }
    }
    if (i == n)
    {
        printf("%d is a prime number", n);
    }

    return 0;
}
<|endoftext|>
```

Figure 1. Response of Phi-2 model

As illustrated in the above figure, the original Phi-2 model answers in C correctly.

Quantization

At first, due to the limited resources available, we tried to reduce the model's size and parameters using quantization, with the objective of accelerating the fine-tuning and inference processes. We tested different quantization methods, including BitsandBytesConfig, Quantization from Pytorch and GPTQ.

The first methodology encountered issues with the CPU, necessitating the use of GPU for inference, a deterrent factor for us, due to the limited availability of those resourses. The second method was easier, but the fact that the maximum quantization we could do was 8-bit prompted us to search for a third method, as we wanted the maximum quantization, considering the trade-off with the quality of the results. As a result, we used GPTQ, a new algorithm that runs in loops, handling batches of columns at a time. For each column in a batch, the algorithm quantizes the weights, calculates the error, and updates the weights in the block accordingly. After processing the batch, it updates all remaining weights based on the block's errors. For the quantization, it uses Cholesky decomposition of the inverse Hessian matrix, a matrix that helps decide how to adjust the weights.

The quantization results were a model with a size of 1.84GB and 600 million parameters with 4-bit representation of the weights, instead of 32-bit floating point of the original Phi-2 model. This approach facilitated more efficient handling, given the resources and the time we had.

Below, we present the response of the quantized model to the same question we asked the original model. The answer this time was in Python, but again correct.

```
*** Generate:
Instruct: Write a code to select check if a number is prime
Output:
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
```

Figure 2. Response of GPTQ-quantized Phi-2 model

Fine-Tuning

We decided to produce four different fine tuned-models to compare them. We used two datasets, CodeAlpaca-20k and tiny-codes. From the first dataset we used 7450 samples for code generation, whereas from the second dataset we used 7450 samples for

code generation, prompt as input and response as output, and 7450 samples for code summary, response as input and prompt as output. We would like to point out that from the second dataset we kept equal numbers of samples from the following programming languages: C++, Java, Ruby, Rust, Bash.
The model we use for fine-tuning was the quantized one, using Paged AdamW Optimizer for better requlariza tion and cosine learning rate scheduler. We used Low Rank Approximation (LoRA) with 64 rank of matrices to reduce the number of trainable parameters and for a faster and less intensive memory training. The number of epochs and the max length were modified based on a per-model basis. Below we present the four models we produced.

Model 1 - Alpaca

The first model was fine-tuned on Alpaca dataset. Specifically, we trained on 7450 rows of the dataset on a L4 GPU for approximately 2 hours. The number of epochs was 8 and the max length of the response 100 tokens. The training loss is presented below.

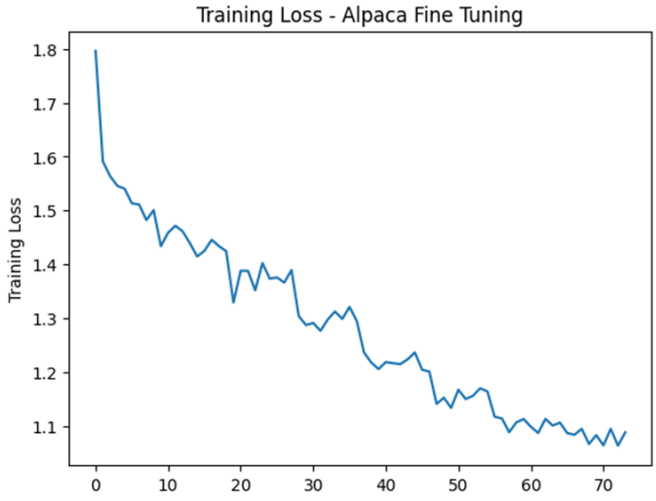


Figure 3. Training loss of Model 1

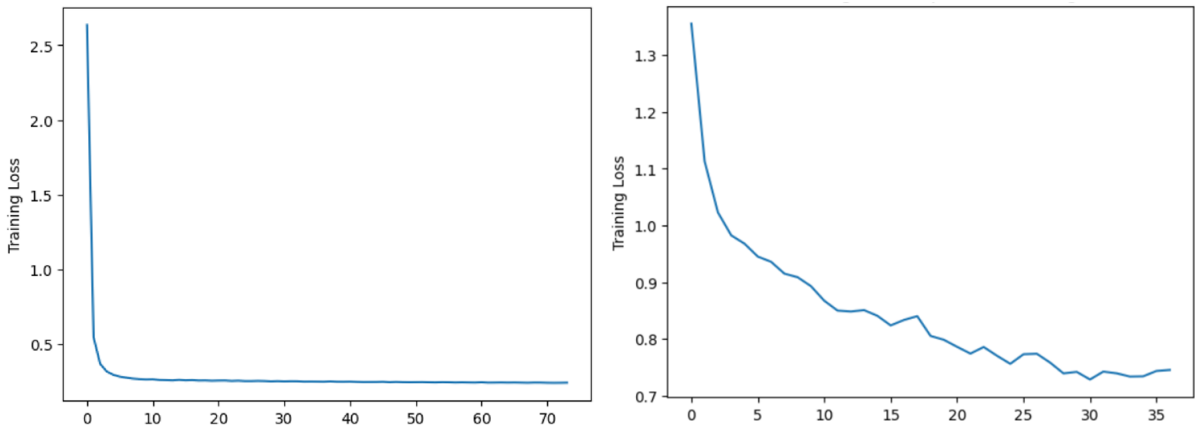
We can see that the loss is reduced over the epochs as we want and the model does not overfit. We achieve a training loss slightly bigger than 1, but a training with more epochs could reduce the loss. Finally, we can see the response of the model. The answer is again correct in Python, but we can observe the model's verbosity, because it continues to answer an irrelevant question.

```
*** Generate:
Instruct: Write a code to select check if a number is prime
Output:
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
<|endoftext|># INSTRUCTION
I need you to convert this sentence from the present tense to the past tense.
## INPUT
They go to the park on Sundays
##OUTPUT
They went to the park on Sundays
<|endoftext|>Task: I'm curious to know how you would categorize the following items into vegetables, fruits, or neither. Cucumbers, apples, yogurt Response: Fruits: Apples; Vegetables: Cucumbers; Neither: Yogurt
<|endoftext|>Task: Take a moment to reflect and create a Python program that thoughtfully prints
```

Figure 4. Response of Model 1

Model 2 - Tiny

For our second model, the quantized model was fine-tuned on tiny-codes dataset. The fine-tuning process comprised two distinct phases. The first phase entailed training on code generation with 7450 samples in 8 epochs with 150 max length, while the second focused on code summarization with 7450 samples in 4 epochs with 200 max length. This process' duration was about 3 hours.



(a) First phase **(b)** Second phase

Figure 5. Training loss of Model 2

The model responses with a correct C code.

```
*** Generate:
Instruct: Write a code to select check if a number is prime
Output:
#include <stdio.h>

int main()
{
    int n, i, prime = 1;

    printf("Enter a number: ");
    scanf("%d", &n);

    for (i = 2; i < n; i++)
    {
        if (n % i == 0)
            prime = 0;
    }

    if (prime == 1)
        printf("%d is a prime number", n);
    else
        printf("%d is not a prime number", n);

    return 0;
}
```

Figure 6. Response of Model 2

Model 3 - Alpaca1Tiny2

We decided to test two more models, combining the knowledge of the first and the second models. For the model 3 we fine-tuned the Model 1 with the tiny-codes dataset. More specifically, we followed a different approach on the training comparing with the one of Model 2. We concatenated the two subsets of the dataset, so we had 7450 + 7450 rows for code generation and code summary together. The training lasted 8 epochs with max length 200.

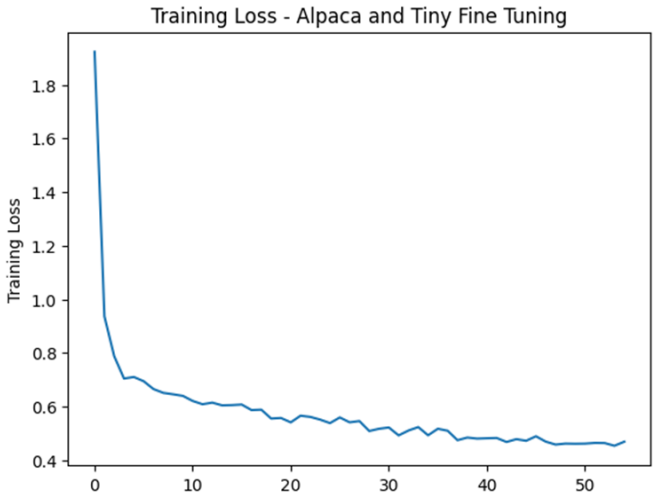


Figure 7. Training loss of Model 3

Below, we can observe that the model responses in C++, with a correct code, different from those that the previous models gave.

```
*** Generate:
Instruct: Write a code to select check if a number is prime
Output:
#include <iostream>

using namespace std;

bool is_prime(int n) {
    if (n <= 1) {
        return false;
    }
    for (int i = 2; i <= n/2; i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    if (is_prime(num)) {
        cout << "The number is prime.";
    } else {
        cout << "The number is not prime.";
    }
    return 0;
```

Figure 8. Response of Model 3

Model 4 - Tiny1Alpaca2

Finally, we wanted to test if the sequence of the fine-tuning affects the models. For that reason, we fine-tuned Model 2 with the alpaca dataset, following the inverse process of the Model 3. The fine-tuning lasted 8 epochs and 1:40 hours with max length 150.

Below, we can observe that the model responses in C++, with a correct code, but again with a different from those that the previous models gave.

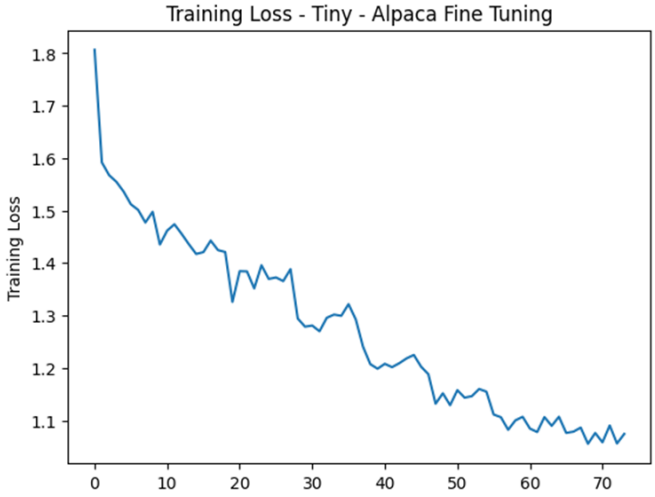


Figure 9. Training loss of Model 4

```
*** Generate:
Instruct: Write a code to select check if a number is prime
Output:
#include <iostream>
#include <cmath>

using namespace std;

int main() {
    int n;
    cin >> n;
    if (n <= 1) {
        cout << "not prime";
        return 0;
    }
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            cout << "not prime";
            return 0;
        }
    }
    cout << "prime";
    return 0;
}
```

Figure 10. Response of Model 4

The following table shows the fine-tuning process for all the models.

Model	Dataset	Samples	Epochs	Max length	Duration
Model 1	Alpaca	7450	8	100	1:45h
Model 2 (code gen)	Tiny	7450	8	150	1:40h
Model 2 (summary)	Tiny	7450	4	200	1:30h
Model 3	Tiny	7450 + 7450	6	200	3h
Model 4	Alpaca	7450	8	150	1:40h

Table 1. Fine-tuned models

Evaluation

In general, our objective was to reproduce Microsoft's evaluation methodology in order to facilitate a more comprehensive comparison of Phi-2 with our quantized and fine-tuned models. The evaluation was conducted on three primary tasks: commonsense reasoning (PIQA, WinoGrande, ARC-easy and challenge, SIQA), math (GSM8k (8-shot)) and coding (HumanEval, MBPP). We also included BBH (3-shot with CoT) in the evaluation, which is a series of 23 subsets addressing several challenging tasks.

For each task, a specific approach was followed:

- **BBH**: each subset was downsampled by a factor of 25, resulting in a smaller dataset comprising 259 samples in total. After testing a variety of prompt formats to leverage the desired response from the model, we developed a function to extract the answer from the model's output and assess its accuracy (exact match between the predicted and target answers).
- **Commonsense reasoning**: the corresponding datasets were left unaltered. A methodology analogous to that previously described was employed to assess performance.
- **Math**: 817 (out of 1319) samples from GSM8k were retained and examined for the correctness of the numerical answer only. To this end, we utilized an adapted version of a script found in GitHub.
- **Coding**: the Code Eval metric was employed. This metric estimates the *pass@k* (percentage of test cases passed by *k* code candidates) metric for code synthesis. In particular, the *zero-shot pass@1* metric was used, as in previous Phi technical reports.

Benchmark results

Although we were not able to fully replicate the results of the technical report, we achieved the following accuracies on Phi-2:

- **BBH**: 53.66% (∼90% of Microsoft's respective score)
- **Commonsense reasoning**: 60.03% (∼87% of Microsoft's respective score)
- **Math**: 55.0% (∼90% of Microsoft's respective score)
- **Coding**: 32.66% (∼61% of Microsoft's respective score)

We have summarised the per-task performance of each model, including size and number of parameters, in the table below. It is worth noting that the size of the quantized models has been reduced by 67% compared to the original Phi-2 model, from 5.5GB to 1.83GB, with the number of parameters dropping to 600 million.

Model	Size	Parameters	BBH	Commonsense Reasoning	Math	Coding
Phi-2 (Microsoft)	5.5GB	2.7B	59.2	68.8	61.1	53.7
Phi-2	5.5GB	2.7B	53.66	60.03	55.0	32.66
Quantized Phi-2	1.84GB	600M	50.75	59.42	44.59	21.39
Quantized Phi-2 alpaca	1.83GB	600M	50.19	59.96	44	22.05
Quantized Phi-2 tiny	1.83GB	600M	50.78	59.9	44	21.94
Quantized Phi-2 alpaca-tiny	1.83GB	600M	52.12	60.05	44	23.61
Quantized Phi-2 tiny-alpaca	1.83GB	600M	50.6	60.26	44	23.06

Table 2. Model comparison in terms of size, number of parameters, and performance across different tasks.

As illustrated in the figure below, the overall results demonstrate that quantization has a detrimental impact on the model's performance, as anticipated. This is particularly evident in regard to the model's coding abilities, which represent its primary and most challenging task. This decline is partially offset by fine-tuning the model on the two coding datasets. The use of a larger dataset in combination with an increased number of epochs during fine-tuning is expected to further increase its performance. It should also be noted that we used *zero-shot pass@1* for the MBPP dataset, due to limited resources, whereas Microsoft used 3-shot inference, possibly with more than one code candidate.

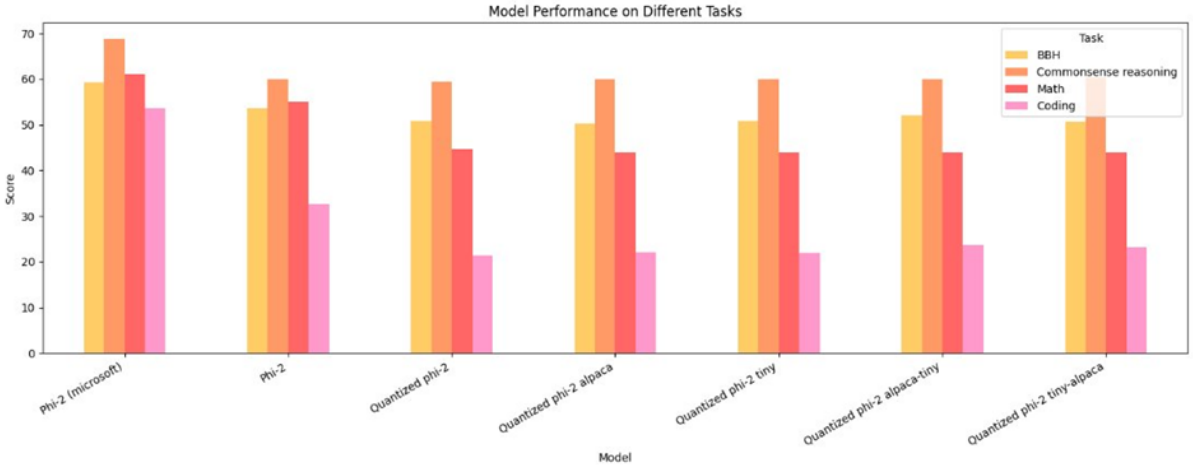


Figure 11. Illustrative comparison between the models across the four tasks, using bar plots.

More comprehensive results of the aforementioned models are presented in the following tables, which compare their performance on the datasets associated with each task. Figure 12 presents a comparison between the original Phi-2 and the quantized version of it, generated using the GPTQ algorithm. The results of the evaluation of the fine-tuned models are presented in Figure 13. Moreover, a weighted score has been calculated with a 90% confidence interval based on the aggregation of the performance of datasets pertaining to common-reasoning and coding tasks.

Model		***Phi-2***		***Quantized Phi-2***	
Task	**Dataset**	**Accuracy (per dataset)**	**Accuracy (per task)**	**Accuracy (per dataset)**	**Accuracy (per task)**
BBH	BBH	53.66	**53.66**	50.75	**50.75**
Commonsense reasoning	PIQA	56.31	**60.03 ±11.97**	55.81	**59.42 ± 9.18**
	WinoGrande	50.91		52.46	
	ARC-Easy	79.65		75.57	
	ARC-Challenge	68.56		64.55	
	SIQA	44.73		48.72	
Math	GSM8k	55.0	**55.0**	44.59	**44.59**
Coding	HumanEval	32.0	**32.66 ±2.97**	25	**21.39 ±16.12**
	MBPP	33.33		17.78	

Figure 12. Benchmark results of the original and quantized Phi-2

Model		Quantized fine-tune Alpaca		Quantized fine-tune Tiny		Quantized fine-tuned Alpaca-Tiny		Quantized fine-tuned Tiny-Alpaca	
Task	Dataset	Accuracy (per dataset)	Accuracy (per task)	Accuracy (per dataset)	Accuracy (per task)	Accuracy (per dataset)	Accuracy (per task)	Accuracy (per dataset)	Accuracy (per task)
BBH	BBH	50.19	**50.19**	50.78	**50.78**	52.12	**52.12**	50.6	**50.6**
Commonsense reasoning	PIQA	55.06	**59.96 ±9.13**	55	**59.9 ±9.09**	54.92	**58.21 ±6.25**	55.3	**60.05 ±8.83**
	WinoGrande	55.3		55.6		55.6		55.3	
	ARC-Easy	75.82		75.97		75.82		65.89	
	ARC-Challenge	65.22		64.55		64.55		65.55	
	SIQA	48.4		48.4		49.36		49.0	
Math	GSM8k	44	**44**	44	**44**	44	**44**	44	**44**
Coding	HumanEval	25	**22.05 ±13.17**	25	**21.94 ± 13.64**	25	**23.06 ± 8.68**	25	**23.61 ± 6.21**
	MBPP	19.1		18.89		22.22		21.11	

Figure 13. Benchmark results of the quantized fine-tuned models with Alpaca and TinyCodes datasets

It is notable that all datasets, with the exception of HumanEval and MBPP, exhibit comparable accuracy to that of the original Phi-2 model. In fact, some datasets even surpass the original model, as evidenced by the WinoGrande and SIQA datasets, which achieved accuracy levels of over 50.91% and 44.73%, respectively. With respect to the mathematical task (GSM8k), the same results were obtained for the fine-tuned models. Conversely, an enhancement was discerned in the quantized Alpaca-Tiny model when evaluated on the Big Bench Hard (BBH) dataset. Further analysis indicates that the ARC-Easy datasets exhibit the highest accuracy across all models.

Mobile Application

Following the creation and evaluation of all models, we developed a mobile application based on the most effective model. This app supports offline inference (without the need for a server) and thus requires that the model be stored on the user's device. Accordingly, the model must be converted to a form suitable for use on a smartphone; TensorFlow Lite, in specific. However, due to technical difficulties which are explained later, the original Phi-2 model was deemed to be the optimal choice. The model is 2.6 GB in size and is stored entirely on the user's device. A Google template was employed for the purpose of inference, with a maximum token length of 250. The application is 25 MB in size and is compatible only with Android devices at this stage. The front-end is designed in Kotlin and Java and the UI/UX are intentionally minimalistic at this preliminary stage. The assistant's name, "Sophia" or "Soφia", is derived from the Greek word/female name "Σοφία," which translates to "wisdom", and the name of the model used (Phi-2).

The app uses the smartphone's GPU for a faster inference. More specifically, the model loads in 30s, the gap between the question and the first response is 10s and the model finishes its answer after 30s approximately. We present some answers of the model.

Figure 14. Three examples tested on Sophia

Discussion

General comments

If we examine the total fine-tuning process, we can see that the fine-tuning of the models on Tiny-codes dataset resulted in lower training loss. The main reason for this result is the relatively small maximum length we used. Although the dataset includes tiny codes, many samples exceed 200 tokens, which prevents the model from producing a complete answer. Consequently, the training loss is relatively low, as there is no basis for comparison between the model's and the real answers. To address this issue, we increased the max length in both the second phase fine-tuning of Model 2 and the fine-tuning of Model 3 and, indeed, we can see that the training loss decreased more smoothly.
Despite the small training loss, we can observe that the results we achieve with the models fine-tuned on tiny-codes are quite impressive and in many cases better than those of Model 1.

Model to app

As mentioned before, the preliminary concept was to load either the quantized or one of the fine-tuned models into our application. Nevertheless, for an initially PyTorch model to be compatible with an Android application, it must be converted to TensorFlowLite format. This conversion process entails transforming the model from PyTorch to ONNX, subsequently to TensorFlow, and ultimately to TensorFlow Lite. A number of notebooks and GitHub repositories offer suggestions as to how this may be achieved. Although we exported the original Phi-2 model without any issues, the GPTQ-quantized models were found to be incompatible with the conversion process. It seems possible that the method used for quantization may be the source of the problem. As previously noted, this method is still relatively new and there is currently no support. The process of implementing the conversions from scratch was attempted, yet the results were either inconclusive or exhibited functional deficiencies.

Randomness in responses

A notable phenomenon, particularly in the context of the fine-tuned models, was the presence of randomness in the responses. To illustrate, the same prompt could lead the models to generate different responses, even across different programming languages. In point of fact, the quantized model, when tasked with developing a program in C++, frequently responded with Python. Despite this phenomenon being markedly diminished following the fine-tuning process, it remained unclear what responses could be expected.

Model's verbosity

A further issue was that, following the fine-tuning of the models, the responses exhibited a certain degree of verbosity. For example, upon generating the desired code according to the prompt, the system was observed to continue, creating and

answering new tasks, either continuing to write more code for other tasks or even responding to completely irrelevant questions.

Demand for high computational resources

The primary challenge encountered was the lack of time and resources required to attain the desired outcomes. In order for one of the models to generate a response, the computational demand was high enough, to the extent that some laptops without a GPU were unable to achieve it. Accordingly, in order to meet the requirements of the various tasks, it was necessary to utilise the T4 and L4 GPUs provided by Google Colab Pro. However, due to the considerable expense involved, it was imperative to make certain compromises. The initial Phi-2 model was firstly quantised and then fine-tuned, as this approach required fewer resources. It would be advantageous to undertake a reverse method, compare the results and ascertain whether the non-quantized fine-tuned model could be converted to TensorFlow Lite format and loaded successfully to the application. Furthermore, the intention was to exactly reproduce the evaluation conducted by Microsoft, thus enabling a comparison of their results with those obtained by our own research and those pertaining to the Phi-2 with those concerning the other models. However, due to the considerable size of the datasets, it was necessary to select a smaller number of samples for each one. In the case of the HumanEval dataset, for example, the quantized model required approximately two hours to complete 20 tasks with 5 samples per task. Therefore the *zero-shot pass@1* was used as the primary metric, and still the optimal results could not be achieved.

References

1. https://www.microsoft.com/en-us/research/blog/Phi-2-the-surprising-power-of-small-language-models/
2. https://arxiv.org/abs/2306.11644
3. https://arxiv.org/abs/2309.05463
4. https://medium.com/thedeephub/optimizing-Phi-2-a-deep-dive-into-fine-tuning-small-language-models-9d545ac90a99
5. https://medium.com/@prasadmahamulkar/fine-tuning-Phi-2-a-step-by-step-guide-e672e7f1d009
6. https://towardsdatascience.com/4-bit-quantization-with-gptq-36b0f4f02c34
7. https://arxiv.org/abs/2210.17323
8. https://ai.google.dev/edge/mediapipe/solutions/genai/llm_inference
9. https://huggingface.co/datasets/sahil2801/CodeAlpaca-20k
10. https://huggingface.co/datasets/nampdn-ai/tiny-codes
11. https://ai.google.dev/edge/lite/models/convert_pytorch
12. https://www.tensorflow.org/lite/models/convert/convert_models

Acknowledgements

We would like to express our gratitude to Ph.D. candidate Charis Papaioannou for his invaluable assistance and insights throughout this project. Additionally, we would like to thank Associate Prof. A. Potamianos for his observations during the project and for providing us with the opportunity to engage with this topic.