

National Research University Higher School of Economics
Faculty of Computer Science
Programme 'Master of Data Science'

MASTER'S THESIS

**Multilingual Telegram-Bot
Based on a Large Language Model
with Mathematical Reasoning**

Student: Aliaksandr Lemiasheuski

Supervisor: Anastasia Maximovskaya

Reviewer: Michael Zotov

Moscow, 2024

ABSTRACT

Inspired by original work of N. Chen et al. [1] this paper is devoted to a reproduction of an LLM fine-tuning experiment under limited resources with a focus on mathematical reasoning, then running the resulting model for inference, and using it as a backbone for a Telegram bot.

We shall outline the reference experiment of the original work, look through available runtime environment setups and frameworks. We shall also consider model optimization approaches, which is frequently a case when dealing with Large Language Models.

Further we shall perform several trial runs in order to select the final experimental setup and move on to the model fine-tuning. The resulting model will then be tested through batch inference and served for powering a Telegram bot available at https://t.me/thesis_paper_bot.

Program code used for this paper can be found at https://github.com/jest0funk/thesis_paper

CONTENTS

ABSTRACT	2
CONTENTS	3
ABBREVIATIONS.....	4
INTRODUCTION.....	5
CHAPTER 1. EXISTING WORK AND THEORETICAL BASIS	6
1.1. Problem statement	6
1.2. Review of the available results and current approaches.....	8
1.3. Overview of the Large Language Model employed.....	16
1.4. Rotary Position Embedding (RoPE).....	20
1.5. Low-Rank Adaptation (LoRA).....	21
CHAPTER 2. EXPERIMENT PREPARATION AND RUNNING	22
2.1. Outline of the reference experiment	22
2.2. Runtime environment and framework.....	26
2.3. Considerations of model optimization.....	27
2.4. Trial runs and conclusions.....	30
2.5. Experimental setup	34
2.6. Experimental model finetuning	35
CHAPTER 3. TESTING, INFERENCE AND CONCLUSIONS.....	36
3.1. Model testing	36
3.2. Inference through a Telegram bot	39
Inference results for MathOctopus Parallel 7B	41
Inference results for LLaMA 3 8B Instruct.....	42
Inference results for LLaMA 3 8B Instruct Parallel – instance 05	43
Inference results for LLaMA 3 8B Instruct Parallel – instance 11	44
Summary response map	45
3.3. Conclusions	46
SUMMARY	48
REFERENCES.....	49

ABBREVIATIONS

CoT – Chain-of-Thought (CoT)

GGUF – GPT-Generated Unified Format

GSM8K - Grade School Math dataset

LLaMA (also Llama) - Large Language Model Meta AI

LLM – Large Language Model

LoFTQ – Low-rank Factorization of Tensor-Train layers for Quantization (a PEFT technique)

LoRA – Low-Rank Adaptation

MGSM - Multilingual Grade School Math dataset

PEFT – Parameter-Efficient Fine-Tuning

RLAIF – Reinforcement Learning from AI Feedback

RLHF – Reinforcement Learning from Human Feedback

RoPE – Rotary Position Embedding

TRL – Transformer Reinforcement Learning

INTRODUCTION

This paper is devoted to a reproduction of an LLM fine-tuning experiment with a focus on mathematical reasoning, then running the resulting model for inference, and using it as a backbone for a Telegram bot.

In Chapter 1 we shall state the problem, give an overview of the available results and current approaches. We also shall cover such theoretical aspects as architecture and description of the LLM, selected for the experiment, as well as methods of Rotary Position Embedding (RoPE) and Low-Rank Adaptation (LoRA).

Chapter 2 will be dedicated to the experiment. We shall outline the reference experiment, runtime environment setups and frameworks and consider model optimization approaches. Next, after several trial runs, we shall define the experiment setup and perform the fine-tuning.

In Chapter 3 we shall test our final model using batch inference, serve it and use it for powering a Telegram bot. After that we shall draw our conclusions.

CHAPTER 1. EXISTING WORK AND THEORETICAL BASIS

1.1. Problem statement

In this paper we build upon the work of N. Chen et al. [1]. Our primary objective is to conduct a similar experiment on the next generation of LLaMA (Large Language Model Meta AI) for improvement of model mathematical reasoning capabilities. We shall extend the analysis to include comparison of several models and select one of them as the backbone for a telegram bot. The models in focus:

- the original model based on LLaMA 2 7B Model [1] and its weight-optimized version;
- LLaMa 3 8B Instruct model, the base model for our experiment;
- a set of resulting models differently fine-tuned.

In their work [1], the authors extended mathematical reasoning tasks from monolingual to multilingual contexts. They aspired to enable only one model to successfully solve mathematical problems presented in various languages.

We shall concentrate on this part of the experiment, where both the tasks and the model responses are in the same language. The reason is that such a model has shown better results on in-domain tests, which is widely accepted benchmark. The in-domain test-set used by the authors is MGSM (Multilingual Grade School Math) [2], the first multilingual benchmark to evaluate arithmetic reasoning abilities of language models. MGSM is an extension of the GSM8K (Grade School Math) [3], which is a dataset of about 8 thousand high quality linguistically diverse grade school math word problems created by human problem writers in English.

Here we also need to note that we prioritize a noticeably better performance in English over a moderate performance across the language set. This point of view goes along with the statement in [1] that multilingual fine-tuning not only increases model

performance with regard to the languages involved but also adds noticeable boost to model results in English compared to a solely English-based monolingual fine-tuning.

Further in [1], the authors endeavored to have the model furnish correct responses in a specific language, even when the problems are presented only in English. The objective was to develop a model capable of understanding and solving mathematical problems across diverse linguistic mediums, emphasizing flexibility and adaptability to different languages while maintaining the integrity of mathematical reasoning.

We shall not follow this part of the experiment as it did not lead to a confident performance increase on the in-domain test-set.

Generally, this work is a three-stage experiment:

- fine-tuning an LLM,
- launching the LLM for inference,
- serving the LLM and building a telegram-bot powered by it.

The specific of the experiment is that it relies only on the resources provided by the Compute Engine of Google Colab runtime environment.

It also should be noted that the purpose of this paper is not in making a production-ready telegram-bot. This is proof of concept; our target is to fine-tune and launch an LLM under limited resources and evaluate inference results in the context of the mathematical reasoning problem.

1.2. Review of the available results and current approaches

Mathematical Reasoning. Prior research has illustrated that by employing prompting techniques, such as Chain-of-Thought (CoT) – a series of intermediate reasoning steps prompting [4], LLMs can be guided through step-by-step reasoning, resulting in significant performance improvements across an array of diverse reasoning tasks.

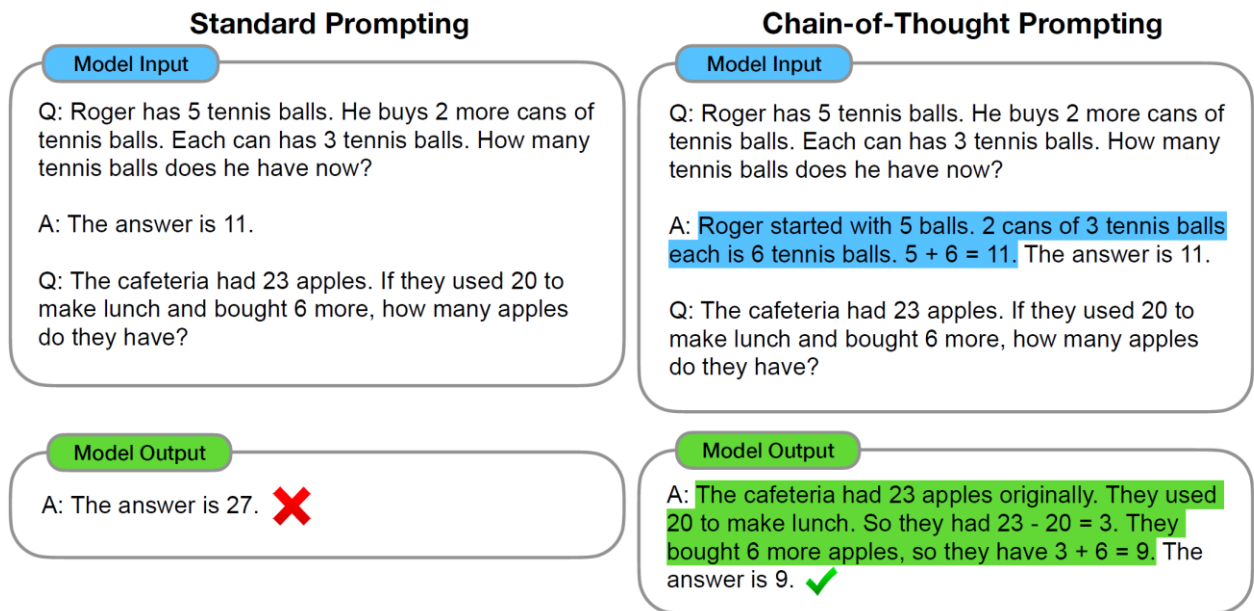


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted [4]

Step1: Forward Reasoning

Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?
A: There are 15 trees originally. Then there were 21 trees after some more were planted. So there must have been $21 - 15 = 6$. The answer is 6.
 (CoT Prompt)

Q: Jackie has 10 apples. Adam has 8 apples. How many more apples does Jackie have than Adam?

Sampling Decoding (LLM) →

A₁: Since Jackie has 10 apples and Adam has 8 apples, in total, Jackie has $10+8=18$ more apples than Adam, so the answer is 18.

A₂: Jackie has 10 apples, so Jackie has $10-8=2$ more apples than Adam, and the answer is 2.

A_k

Step2: Backward Verification

1) Rewritten Candidate Conclusion

Please change the questions and answers into complete declarative sentences [Q] The answer is [A]

$Q + A \Rightarrow \text{LLM} \Rightarrow f_c$

f_c^1 Jackie has 18 apples more than Adam f_c^2 Jackie has 2 apples more than Adam f_c^k

2) Rewritten Condition

f_1 Jackie has 10 apples. $\Rightarrow \text{Regular}(f_1, \neg(d \wedge \neg d)) \Rightarrow f_1$ "Jackie has 'X' apples." What is the answer of 'X' 10

Arithmetic Tasks: Condition Mask Verification

f_1 Jackie has 10 apples. $\Rightarrow f_1$ "Jackie has 10 apples." Do it is correct (True or False)?

General Tasks: True-False Item Verification

3) Verification

Sampling Decoding

Q₁: "Jackie has 'X' apples. Adam has 8 apples. Jackie has 18 apples more than Adam" What is the answer of 'X'.

A₁: Jackie has X apples, and Jackie's apples are 18 more than Adam's, so Jackie's apples should be $8+18=26$, the answer is 26.

A₁ Scores: 1 (X, X, X)

Sampling Decoding

Q₂: "Jackie has 'X' apples. Adam has 8 apples. Jackie has 2 apples more than Adam" What is the answer of 'X'.

A₂: Jackie has X apples, which is 2 more than Adam's 8 apples, so $8+2=10$, the answer is 10.

A₂ Scores: 4 (V, V, V, V)

Sorted by scores

Answer: 2

We take Condition Mask Verification as an example. For True-False Item Verification, we only need to count the number of True's to calculate the scores.

Figure 2: Example of self-verification. In step one, LLM generates candidate answers and forms different conclusions. Then, in the step two, LLM verifies these conclusions in turn and computes the verification score [5]

The idea of CoT was further developed by a technique of LLM self-verification [5]. By performing a backward verification of the answers that LLM deduced for itself, we can obtain interpretable answer validation scores to select the candidate answer with the highest score.

Stage 1: Decompose Question into Subquestions

Q: It takes Amy 4 minutes to climb to the top of a slide. It takes her 1 minute to slide down. The water slide closes in 15 minutes. How many times can she slide before it closes?

Language Model

A: To solve "How many times can she slide before it closes?", we need to first solve: "How long does each trip take?"

Stage 2: Sequentially Solve Subquestions

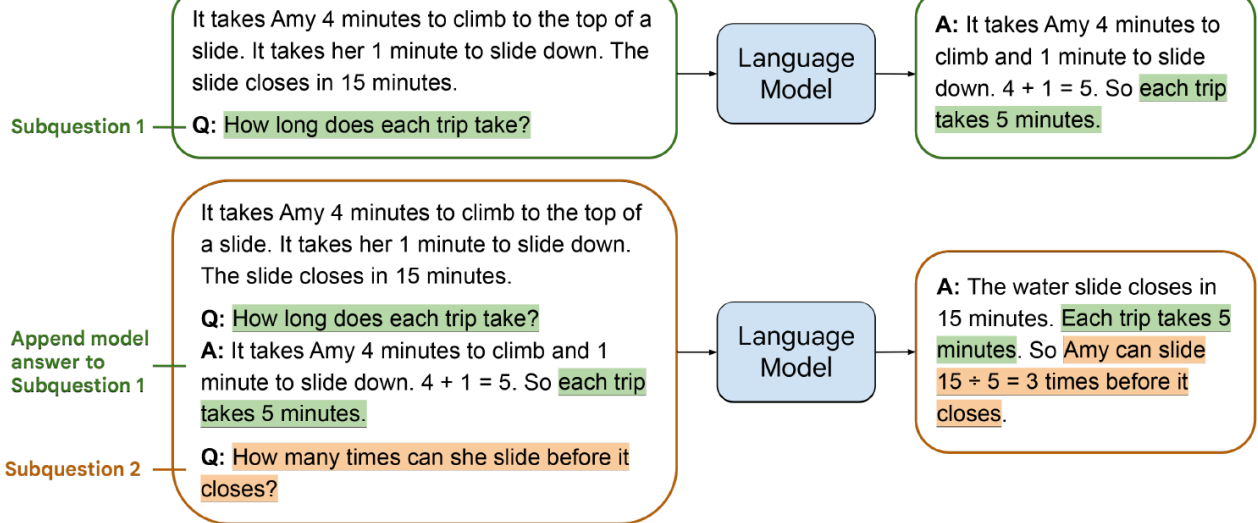


Figure 3: Least-to-most prompting solving a math word problem in two stages: (1) query the language model to decompose the problem into subproblems; (2) query the language model to sequentially solve the subproblems. The answer to the second subproblem is built on the answer to the first subproblem. The demonstration examples for each stage’s prompt are omitted in this illustration [6]

Another approach to cope with poor performance of Chain-of-thought prompting on tasks which requires solving problems harder than the exemplars shown in the training dataset prompts was in another prompting strategy, least-to-most prompting [6]. The key idea in this strategy was to break down a complex problem into a series of simpler subproblems and then solve them in sequence. Solving each subproblem is facilitated by the answers to previously solved subproblems.

As a result, on tasks related to symbolic manipulation, compositional generalization, and math reasoning the least-to-most prompting was capable of generalizing to more difficult problems than those seen in the prompts.

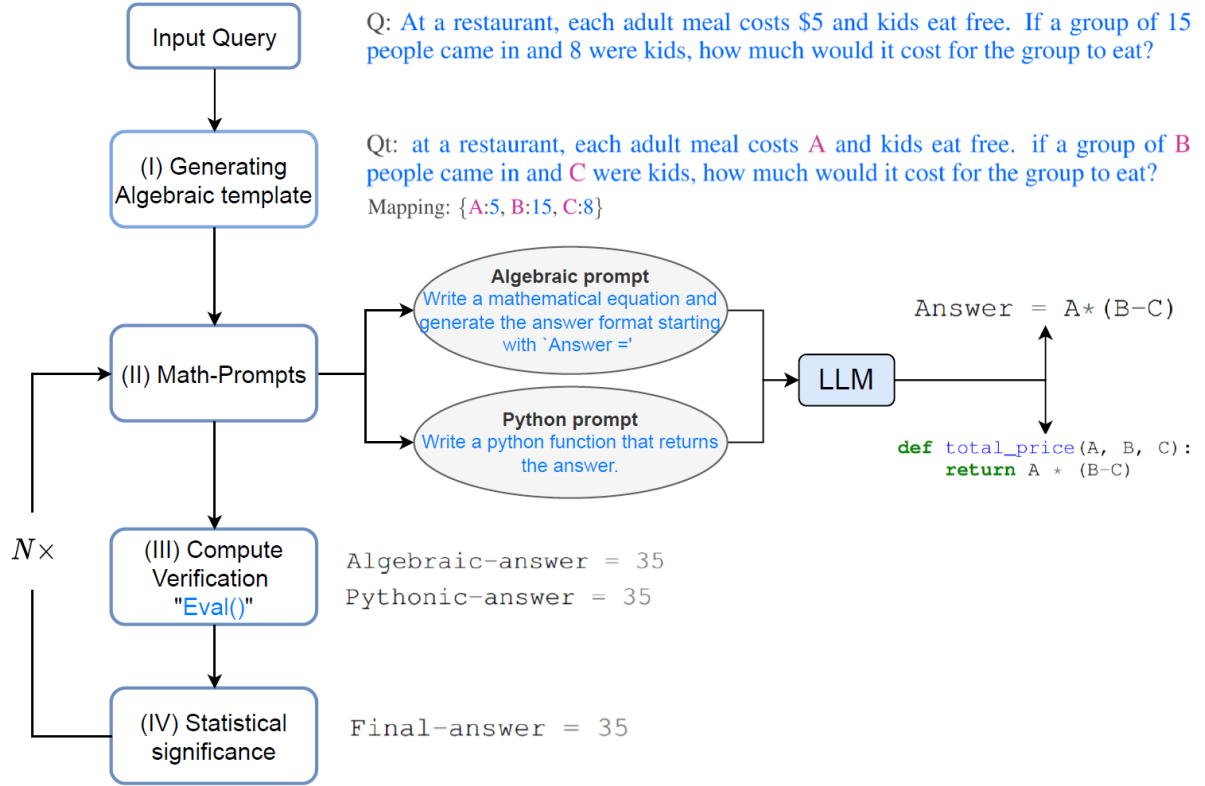


Figure 4: MathPrompter flow. The MathPrompter process is outlined with an example alongside [7]

A further attempt to address the problem of limited performance of LLMs when solving arithmetic reasoning was in employing MathPrompter [7], a technique that improves performance of LLMs on arithmetic problems along with increased reliance in the predictions. MathPrompter uses the Zero-shot chain-of-thought prompting technique to generate multiple Algebraic expressions or Python functions to solve the same math problem in different ways and thereby raise the confidence level in the output results.

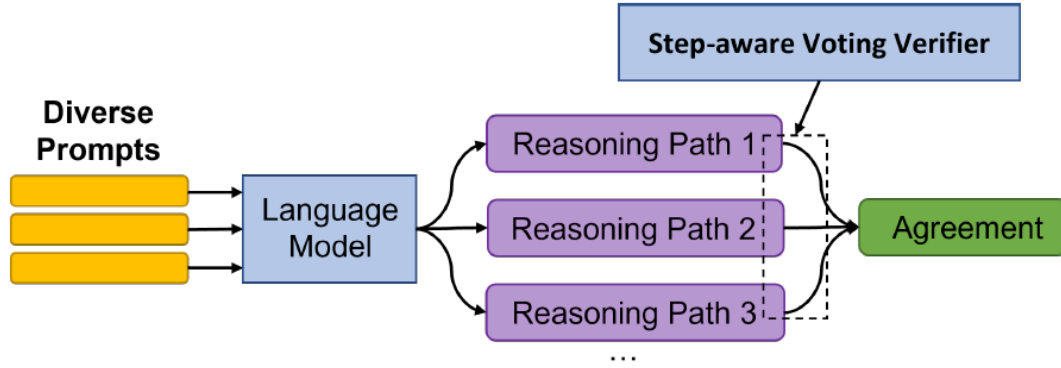


Figure 5 DiVeRSe (Diverse Verifier on Reasoning Step) [8]

Question	Self-Consistency	DiVERSE
[GSM8K] Mark decides to buy packs of old magic cards and open them to sell. He buys 3 packs for \$1800 each. He gets 1 card that is worth \$4000 and another card worth \$1000. There are 30 more cards worth an average of \$50 each. How much money profit did he make?	Mark bought 3 packs for \$1800 each for a total of $\$1800 * 3 = 5400$ (68.1%). He got 1 card worth \$4000 and another card worth \$1000 for a total of $\$4000 + \$1000 = \$5000$ (72.1%). There were 30 more cards worth an average of \$50 each for a total of $\$50 * 30 = 1500$ (63.7%). Mark made a profit of $\$5000 + \$1500 = 6500$ (40.5%). The answer is 6500. Answer: 6500 (Wrong) Score: 62.3%	The cheaper cards are worth a total of $30 * 50 = 1500$ (84.4%). So all the cards together are worth $1500 + 4000 + 1000 = 6500$ (83.3%). The cost for the cards was $1800 * 3 = 5400$ (80.1%). So he made a profit of $6500 - 5400 = 1100$ (70.0%). The answer is 1100. Answer: 1100 (Correct) Score: 91.2%

Figure 6: A GSM8K example (code-davinci-002) with step-level scores given by the step-aware verifier. The scores can not only improve the performance but also help the understanding of where the reasoning paths start to be incorrect [8]

Next, an approach called DiVeRSe (Diverse Verifier on Reasoning Step) was introduced to enhance the reasoning capability of language models [8]. DiVeRSe has three main components: first, it generates diverse prompts to explore different reasoning paths for the same question; second, it uses a verifier to filter out incorrect answers based on a weighted voting scheme; and third, it verifies each reasoning step individually instead of the whole chain.

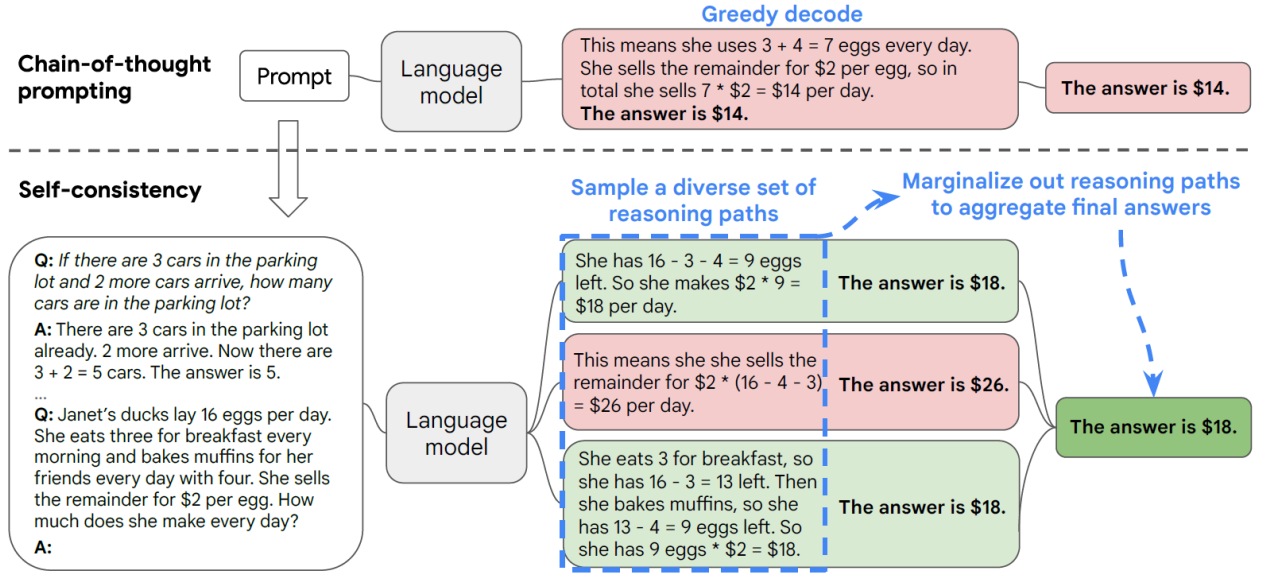


Figure 7: The self-consistency method contains three steps: (1) prompt a language model using chain-of-thought (CoT) prompting; (2) replace the “greedy decode” in CoT prompting by sampling from the language model’s decoder to generate a diverse set of reasoning paths; and (3) marginalize out the reasoning paths and aggregate by choosing the most consistent answer in the final answer set [9]

To replace the naive greedy decoding used in chain-of-thought prompting another decoding strategy, self-consistency, was proposed [9]. It first samples a diverse set of reasoning paths instead of only taking the greedy one, and then selects the most consistent answer by marginalizing out the sampled reasoning paths. Self-consistency leverages the intuition that a complex reasoning problem typically admits multiple different ways of thinking leading to its unique correct answer.

Meanwhile, several scholarly works have incorporated the concept of rejection sampling, in conjunction with various other techniques, to curate a more diverse set of sampled reasoning paths for the purpose of finetuning data augmentation [1].

However, in the reference experiment we are considering the rejection sampling did not improve model performance.

Instruction Tuning. Instruction tuning studies have marked the beginning of an exploration into the generalization capabilities of Large Language Models. Recent innovations in this domain are targeting towards employing synthetic instruction following data, distilled from models like GPT-3/4, to align open-source LLMs [1].

Recently, several works have utilized instruction tuning for training math LLMs. In particular, WizardMath [10] implements the evol-instruct reinforcement learning methodology (RLEIF), which employs reinforcement learning and is directed towards the refinement of prevailing math instruction data.

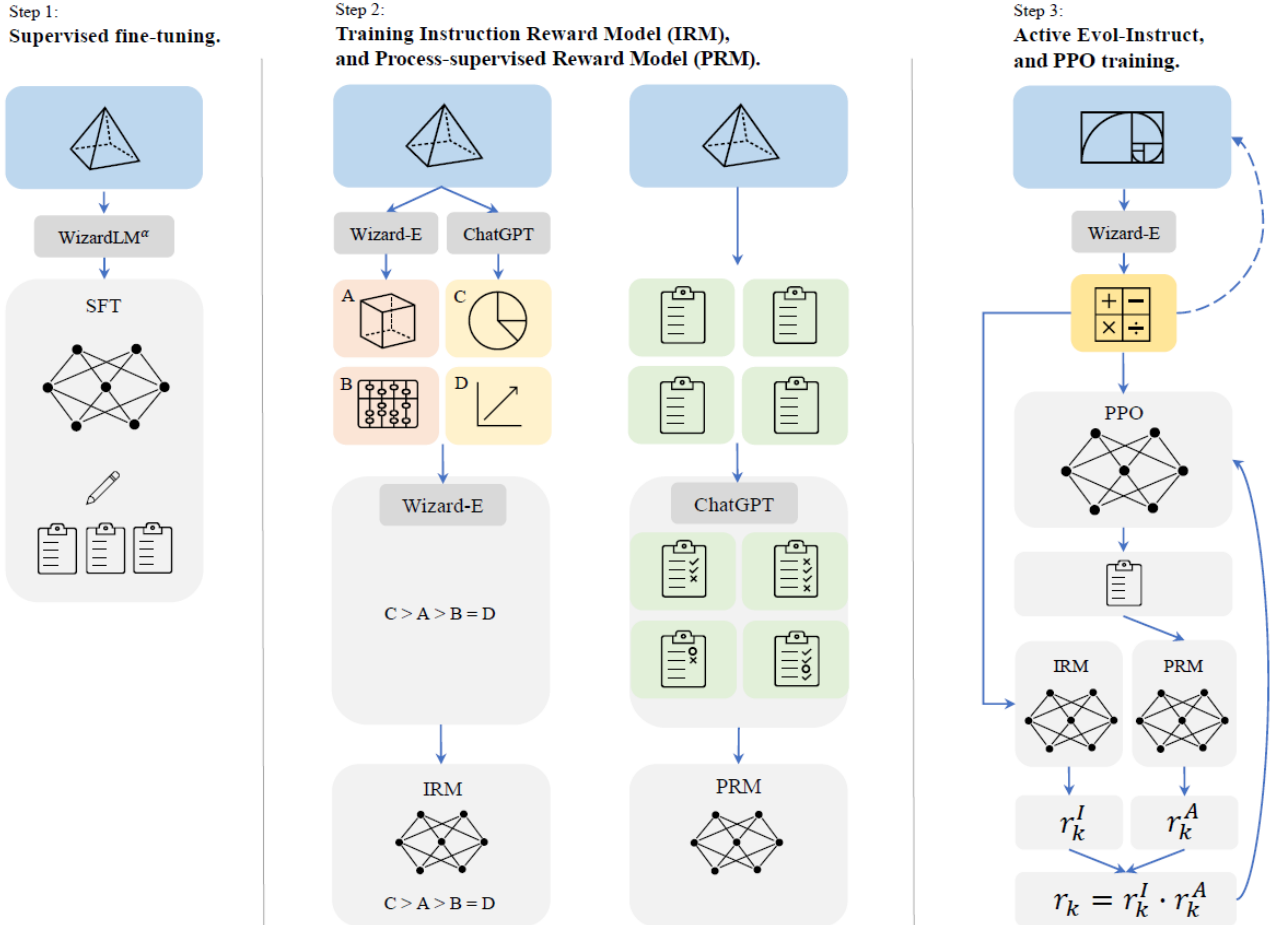


Figure 8: A diagram illustrating the three steps of our Reinforcement Learning from Evol-Instruct Feedback (RLEIF): (1) supervised fine-tuning (SFT), (2) Instruction Reward Model (IRM) training and Process-supervised Reward Model (PRM) training, and (3) Active Evol-Instruct and reinforcement learning via proximal policy optimization (PPO) [10]

Instruction prompt templates proposed in [11] for fine-tuning of Stanford Alpaca model has already become an industry standard.

Prompt with a non-empty input field:

```
Below is an instruction that describes a task, paired with an input that provides further context.  
Write a response that appropriately completes the request.
```

```
### Instruction:  
{instruction}
```

```
### Input:  
{input}
```

```
### Response:
```

Prompt with an empty input field:

```
Below is an instruction that describes a task. Write a response that appropriately completes the request.
```

```
### Instruction:  
{instruction}
```

```
### Response:
```

However, almost all the researchers aim to improve mathematical reasoning in English, leaving multilingual mathematical reasoning less explored. The work we are building our paper on [1] aims to fill this gap by exploring effective methods for training LLM in multilingual mathematical reasoning through combining chain-of-thought approach with alpaca-like instruction prompting.

1.3. Overview of the Large Language Model employed

LLaMA was introduced in February 2023 [12]. This is an autoregressive, decoder-only transformer language model with modified architecture trained on publicly available data.

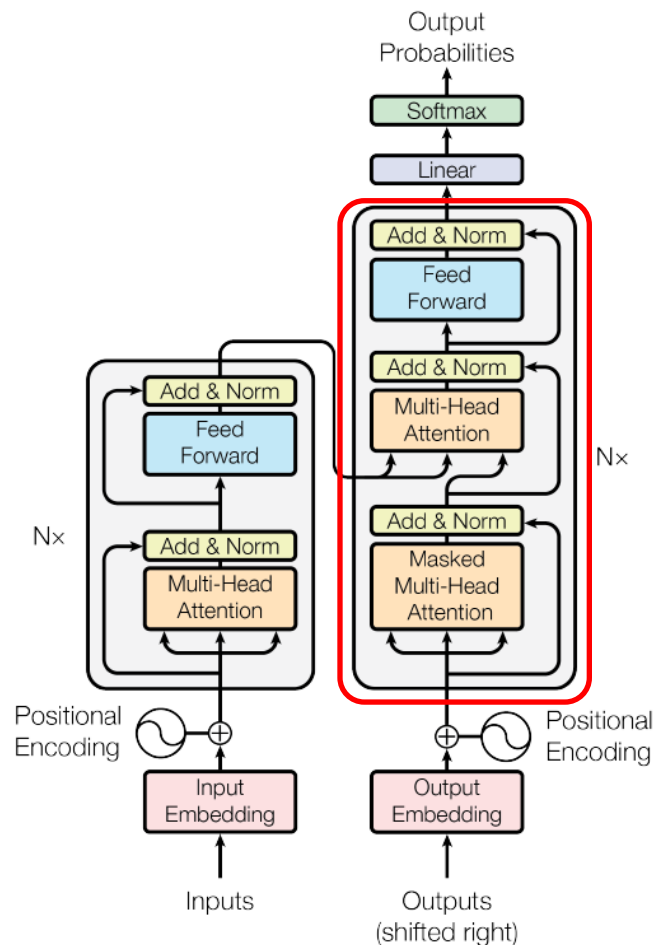


Figure 9: The Transformer - model architecture [13]. The red box outlines the decoder

The main difference with the original transformer architecture were [12]:

- To improve the training stability pre-normalization was applied, normalizing the input of each transformer sub-layer, instead of normalizing the output.
- SwiGLU activation function replaced the ReLU non-linearity.
- The absolute positional embeddings were removed, and instead, rotary positional embeddings (RoPE) added at each layer of the network.

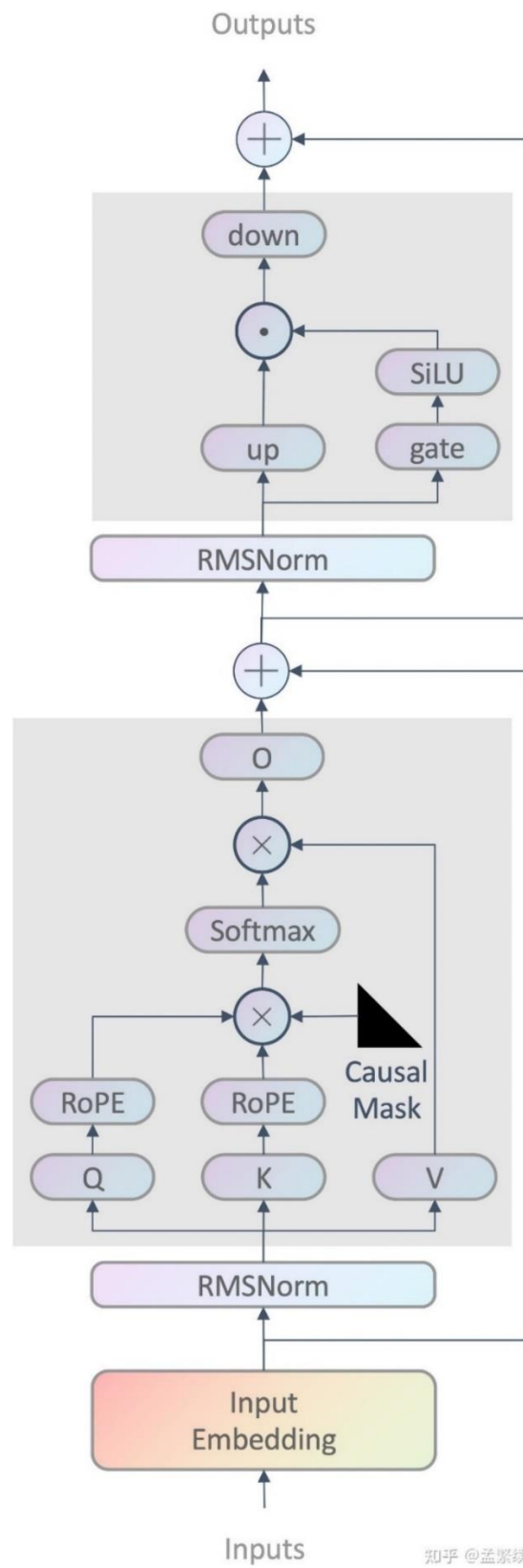


Figure 10: A schematic diagram of LLaMA, which is stacked by Attention and MLP layers [14]

Llama 2 was released in July 2023 [15]. The primary architectural differences from Llama 1 included increased context length and grouped-query attention.

The context window for Llama 2 was expanded from 2048 tokens to 4096 tokens.

Grouped-Query Attention was used to solve the problem of memory cost. A standard practice for autoregressive decoding is to cache the key (K) and value (V) pairs for the previous tokens in the sequence, speeding up attention computation. With increasing context windows or batch sizes, however, the memory costs associated with the KV cache size in multi-head attention models grow significantly. For larger models, where KV cache size becomes a bottleneck, key and value projections can be shared across multiple heads without much degradation of performance through employment of a grouped-query attention variant with 8 KV projections.

After pretraining, Llama 2 underwent supervised fine-tuning and subsequent refinement using reinforcement learning techniques.

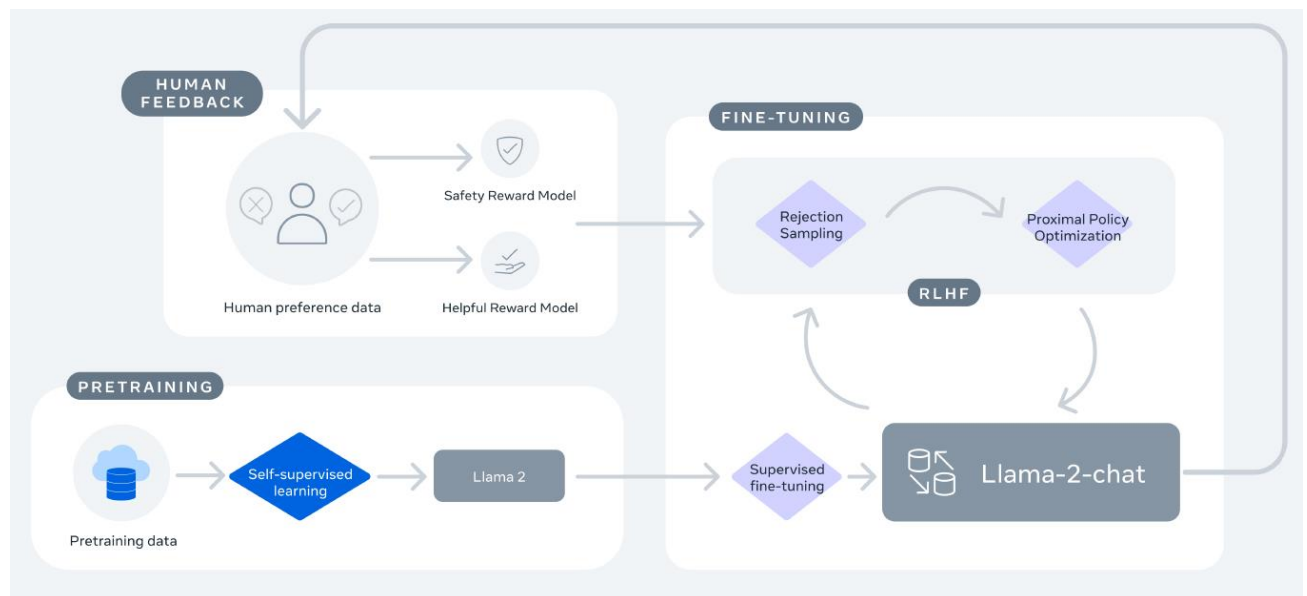


Figure 11: Training of Llama 2-Chat: This process begins with the pretraining of Llama 2 using publicly available online sources. Following this, an initial version of Llama 2-Chat was created through the application of supervised fine-tuning. Subsequently, the model is iteratively refined using Reinforcement Learning with Human Feedback (RLHF) methodologies, specifically through rejection sampling and Proximal Policy Optimization (PPO). Throughout the RLHF stage, the accumulation of iterative reward modeling data in parallel with model enhancements is crucial to ensure the reward models remain within distribution [15]

In turn, Llama 3 was released in April 2024. Compared to Llama 2, there are several key improvements. Llama 3 uses a tokenizer with a vocabulary of 128K tokens that encodes language much more efficiently, which leads to substantially improved model performance. To improve the inference efficiency of Llama 3 models, a grouped query attention was adopted across both the 8B and 70B sizes. The model family was trained on sequences of 8,192 tokens, using a mask to ensure self-attention does not cross document boundaries [16].

Meta Llama 3 Instruct model performance

	Meta Llama 3 8B	Gemma 7B - It Measured	Mistral 7B Instruct Measured
MMLU 5-shot	68.4	53.3	58.4
GPQA 0-shot	34.2	21.4	26.3
HumanEval 0-shot	62.2	30.5	36.6
GSM-8K 8-shot, CoT	79.6	30.6	39.9
MATH 4-shot, CoT	30.0	12.2	11.0

	Meta Llama 3 70B	Gemini Pro 1.5 Published	Claude 3 Sonnet Published
MMLU 5-shot	82.0	81.9	79.0
GPQA 0-shot	39.5	41.5 CoT	38.5 CoT
HumanEval 0-shot	81.7	71.9	73.0
GSM-8K 8-shot, CoT	93.0	91.7 11-shot	92.3 0-shot
MATH 4-shot, CoT	50.4	58.5 Minerva prompt	40.5

Figure 12: Model performance of Meta Llama 3 8B and Meta Llama 3 70B [16]

1.4. Rotary Position Embedding (RoPE)

Positional encoding is a key component of the transformer architecture. It enables supervision for dependency modeling between elements at different positions of the sequence. A method of Rotary Position Embedding (RoPE) [17] can effectively utilize the positional information into the learning process of a pre-trained LLM. Specifically, it encodes the absolute position with a rotation matrix and at the same time incorporates the explicit relative position dependency in self-attention formulation. RoPE adds such properties as the flexibility of sequence length, decreasing inter-token dependency with growing relative distances, and the capability of equipping the linear self-attention with relative position encoding.

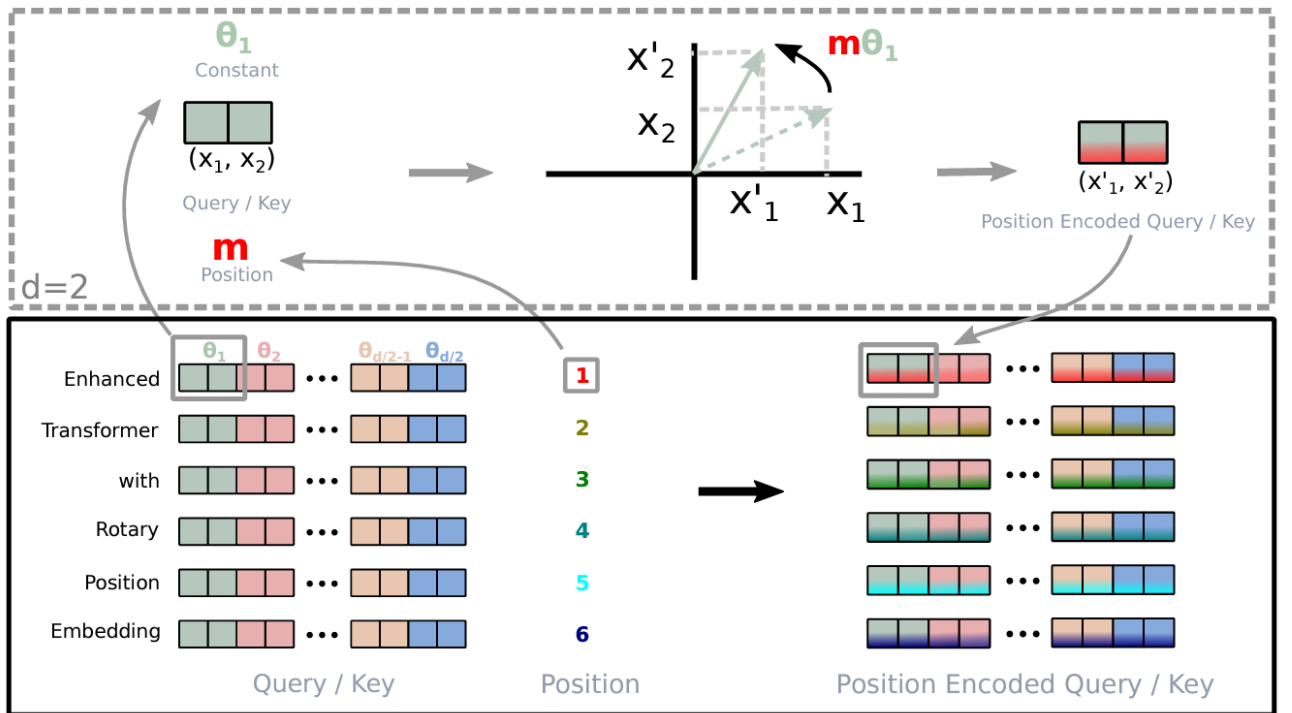


Figure 13: Implementation of Rotary Position Embedding (RoPE) [17]

1.5. Low-Rank Adaptation (LoRA)

An adaptation of LLM done via fine-tuning through updates of all the parameters can become prohibitively expensive for models with billions of parameters.

The Low-Rank Adaptation (LoRA) research [18] proposed freezing the pretrained model weights and injecting trainable rank decomposition matrices into each layer of the Transformer architecture, thereby greatly reducing the number of trainable parameters for downstream tasks, which led to a higher training throughput, and, unlike adapters, no caused additional inference latency.

The general idea is that for a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$ its update is constrained by representing it with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$. During training, W_0 is *frozen* and does not receive gradient updates, while A and B contain trainable parameters. Both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0x$, the modified forward pass yields: $h = W_0x + \Delta Wx = W_0x + BAx$.

At the beginning of training a random Gaussian initialization is used for A and B is set to zero, so the training start with $\Delta W = BA = 0$. Then ΔWx is scaled by $\frac{\alpha}{r}$, where α is a constant in r .

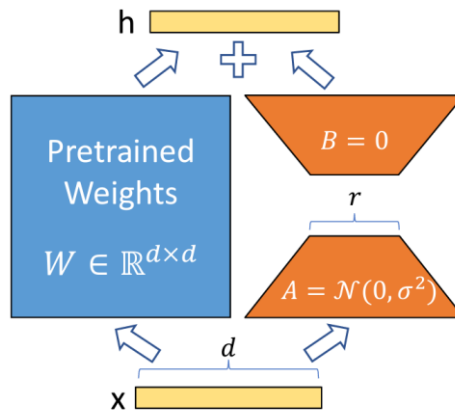


Figure 14: Re-parametrization. Only A and B are trained [18]

CHAPTER 2. EXPERIMENT PREPARATION AND RUNNING

2.1. Outline of the reference experiment

Training Dataset. As the fundamental data source the authors of [1] employed GSM8K [2], an English dataset comprised of middle grade-school mathematical problems annotated by humans and presented in English. According to the officially provided solutions, each problem in GSM8K necessitates a resolution process involving between two to eight steps.

As the target languages in translation the authors of [1] chose a set of ten languages that are typologically varied from English (En), covering eight unique language families. The ensemble of languages incorporated in this study comprises Bengali (Bn), Chinese (Zh), French (Fr), German (De), Japanese (Ja), Russian (Ru), Spanish (Es), Swahili (Sw), and Thai (Th).

Further, the authors [1] employed ChatGPT mass-prompts to translate the problems and their corresponding CoT answers from the English GSM8K training set into the target languages. In an effort to uphold consistency and accuracy across multiple languages, they additionally extracted all formulas present in the translated answers. If all formulas were calculated correctly and were consistent with those in English, they assumed the translation to be accurate and error-free. If errors persisted across five consecutive translations, such particular problems were discarded from the final dataset.

The resulting dataset of about 73.6 thousand samples was called MGSM8KInstruct [1].

Training Strategies. The dataset of multilingual query-response pairs MGSM8KInstruct was used for supervised fine-tuning of the base large language models, which resulted in a family of fine-tuned models called MathOctopus [1]. The following strategies were used:

- *Parallel training* involved filling in the input prompts with questions and answers in the same native language during training.
- *Cross-training* refers to filling in the input prompts with questions in English and answers in a different native language during training.

Multilingual Rejection Sampling. The authors [1] have explored whether performance can be further enhanced by augmenting data through rejection sampling. However, in experiments it was observed that adding augmented data from rejection sampling in multilingual scenarios does not always improve model performance and can even reduce its generality.

Evaluation Dataset. For in-domain testing the MGSM dataset was used [2].

Setup of the reference experiment. In their work [1] the authors used open-source LLaMA-2 7B to 13B and LLaMA-1 33B as backbone models, which allowed to build MathOctopus model family in multiple scales. The program code was based on DeepSpeed [19] and Hugging Face [20] libraries.

For all models the learning rate, epochs and max length were set at $2e-5$, 3 and 512. The batch sizes were set to 8, 4, 2 when models scale from 7B to 33B. The Alpaca-like prompt template [11] is provided below.

Input Prompts	Below is an instruction that describes a task. $\backslash n$ Write a response that appropriately completes the request in { <i>language</i> }. Please answer in { <i>language</i> }. $\backslash n \backslash n$ ### Instruction: $\backslash n$ { <i>query</i> } $\backslash n \backslash n$ ### Response:
----------------------	--

Figure 15 Prompt template [1]

Components, settings, and parameters of the reference experiment collected from the paper and the source code [1].

name	value and justification, if any
Resulting models	MathOctopus ^P 7B / MathOctopus ^P -LoRA 7B The best performers on in-domain testing within 7B models without employment of multilingual rejection sampling for data augmentation
Base model	LLaMA 2 7B The closest to LLaMA 3 8B, selected for our experiment
Training framework	DeepSpeed and Hugging Face
Training dataset	MGSM8KInstruct_Parallel Custom-tailored for the experiment specifics
Testing dataset	MGSM
Max length	512
End of sequence token	< endoftext >
Epochs	3 In LoRA employing script this value set to 4
Learning rate	2e-5
Learning rate scheduler type	cosine
Batch size	8 In LoRA employing script this value set to 16
Weight decay	0.0
Optimizer	Adam DeepSpeedCPUAdam or FusedAdam used depending on whether zero-offload memory optimization was engaged
Adam betas	0.9, 0.95
Warmup steps	0
Gradient accumulation steps	4
LoRA rank	128
LoRA target modules	'q_proj', 'k_proj', 'v_proj', 'o_proj', 'gate_proj', 'up_proj', 'down_proj'
LoRA alpha scaling	1
LoRA dropout	0
Train / validation size	0.995 / 0.005
Validation metric	Perplexity Computed through torch.exp(losses) , where <i>losses</i> is the mean loss on the entire validation dataset
Testing	Local
Testing metric	Accuracy Answer values compared for model predictions and MGSM

Resulting model performance figures.

Models	En	Sw	Zh	Bn	De	Es	Fr	Ja	Ru	Th	Avg.
Close-Source LLMs											
ChatGPT-Zero shot	52.0	30.0	44.4	4.4	46.8	52.0	45.6	38.8	37.2	8.0	35.9
ChatGPT-En 2shot	67.2	40.0	52.8	7.6	62.0	61.2	59.2	46.8	50.4	15.6	46.3
GPT4-En 2shot	80.0	64.4	70.0	17.6	73.6	71.2	72.0	71.6	64.0	40.4	62.5
Open-Source LLMs (7B Model)											
LLaMA 2-LoRA	27.6	4.0	12.0	2.0	10.4	18.4	16.8	7.6	11.2	3.2	11.3
LLaMA 2	43.2	5.2	22.4	3.2	37.2	32.4	34.4	15.2	28.0	4.8	22.6
RFT	44.8	2.8	16.8	2.4	33.6	34.0	34.0	6.8	29.2	2.0	20.6
MAmmoTH	49.6	2.4	17.2	3.6	33.2	32.4	32.8	10.8	26.0	4.8	21.3
WizardMath	47.6	3.4	22.4	2.0	30.4	34.8	30.4	24.0	30.8	4.0	23.0
MathOctopus ^C	52.0	23.6	31.6	18.8	38.0	39.2	36.4	27.2	33.6	21.6	32.2
xRFT-MathOctopus ^C	51.2	24.0	33.2	18.8	36.0	41.2	37.6	29.6	36.4	25.2	33.3
MathOctopus ^P -LoRA	30.4	15.2	23.6	10.4	22.8	24.8	26.4	18.0	22.0	14.8	20.8
MathOctopus ^P	52.4	39.2	38.4	28.8	44.8	42.4	43.6	36.0	39.6	34.4	40.0
xRFT-MathOctopus ^P	54.8	38.4	45.2	33.2	43.6	45.2	38.0	35.6	48.4	36.4	41.9
Open-Source LLMs (13B Model)											
LLaMA 2	50.4	7.6	32.8	6.0	42.8	45.2	40.8	25.2	39.2	6.8	29.7
RFT	52.0	3.6	33.6	3.2	38.4	46.8	44.8	26.4	41.6	4.4	29.5
MAmmoTH	56.4	1.6	31.2	3.6	45.6	50.0	39.6	19.2	36.8	5.2	28.9
WizardMATH	52.8	5.6	28.0	6.4	40.4	45.6	42.0	22.0	34.4	5.6	28.3
MathOctopus ^C	56.4	27.2	39.2	24.0	47.6	49.6	47.6	40.4	42.0	24.8	39.9
xRFT-MathOctopus ^C	53.6	28.0	45.2	21.2	48.0	46.4	46.0	35.2	45.6	28.8	39.8
MathOctopus ^P	53.2	42.8	48.8	35.2	44.4	48.0	48.4	43.2	47.6	46.8	45.8
xRFT-MathOctopus ^P	51.6	46.0	51.2	42.0	49.2	53.2	49.6	39.6	47.6	46.0	47.6
Open-Source LLMs (30-34B Model)											
LLaMA 1	50.8	3.6	27.6	3.2	42.4	44.4	42.4	11.6	38.4	1.2	26.6
RFT	57.6	2.4	26.0	4.8	45.6	46.4	44.8	9.2	46.4	4.4	28.8
MathOctopus ^C	55.6	24.4	36.0	19.2	40.4	51.2	44.4	27.2	37.2	21.6	35.7
xRFT-MathOctopus ^C	53.6	27.6	34.4	19.2	47.2	47.6	44.8	30.8	38.8	22.8	36.7
MathOctopus ^P	56.4	46.8	52.0	35.2	47.2	53.2	48.0	39.2	45.6	41.2	46.5
xRFT-MathOctopus ^P	51.6	47.2	52.4	37.6	51.2	52.8	44.4	41.6	50.0	47.6	47.6

Figure 16: Model Performance on MGSM testset. MathOctopus^P and MathOctopus^C refer to models trained on parallel-training and cross-training, separately. The best results are highlighted in each language of the same backbone [1]

2.2. Runtime environment and framework

During the experiment we utilized most runtime environment setups offered by Compute Engine of Google Colab.

GPU	GPU memory, Gb	RAM, Gb	further reference	access	utilized
CPU only	n/a	12.7		free	-
CPU only	n/a	51.0		paid	-
NVIDIA T4	15.0	12.7	T4	free	+
NVIDIA T4	15.0	51.0	T4+ram	paid	+
NVIDIA L4	22.5	62.8	L4	paid	+
NVIDIA A100	40.0	83.5	A100	paid	+

For our experiment we considered the following frameworks:

- Hugging Face [20] (further reference - HF) – a repository of all large language model needed, key Python libraries peft, trl and transformers; *used in the reference experiment*.
- DeepSpeed [19] (further reference - DS) – a framework for distributed computing and optimized memory utilization; *used in the reference experiment*.
- llama.cpp for Python [21] (further reference - LL) – a framework for LLM inference.
- Unsloth [22] (further reference - US) – a framework for LLM fine-tuning.

In addition, Google Drive storage of 2 Tb was used to facilitate data availability.

2.3. Considerations of model optimization

Having to deal with models of big size we considered existing opportunities for model compression, one of which is GGUF (GPT-Generated Unified Format) [23]. It was released in August 2023 by the creator of llama.cpp framework mentioned above.

Models in this format are usually stored in one file, big files over 10 Gb may be split into volumes.



Figure 17: GGUF File Structure [23]

This format uses quantization of weights to compress the model. Quantization is a technique to reduce the computational and memory costs of running inference by representing the weights and activations with low-precision data types like 8-bit integer (int8) instead of the usual 32-bit floating point (float32).

Reducing the number of bits means the resulting model requires less memory storage, and matrix multiplications can be performed much faster with integer arithmetic. It also allows to run models on devices, which only support integer data types.

8-bit quantization can decrease model size by about two to three times, while lower-bit quantization enables to achieve even better compression.

Quantization	imatrix	Model size [GiB]	PPL	Δ PPL
f16	None	14.97	6.233160 \pm 0.037828	-
q8_0	None	7.96	6.234284 \pm 0.037878	0.002650 \pm 0.001006
q6_K	None	6.14	6.253382 \pm 0.038078	0.021748 \pm 0.001852
q5_K_M	None	5.33	6.288607 \pm 0.038338	0.056974 \pm 0.002598
q5_K_S	None	5.21	6.336598 \pm 0.038755	0.104964 \pm 0.003331
q5_1	None	5.65	6.337857 \pm 0.038677	0.106223 \pm 0.003476
q5_0	None	5.21	6.363224 \pm 0.038861	0.131591 \pm 0.003894

Figure 18: Perplexity - LLaMA 3 8b Scoreboard [24]

However, additional research raised certain concerns regarding quantization approach. It appeared the LLaMA 3 8B degrades noticeably stronger than LLaMA 2 7B under same precision of quantization, which could harm our base model performance in comparison to the base model for the reference experiment.

Metric	L2 7b q2_K	L3 8b q2_K	L2 7b q4_K_M	L3 8b q4_K_M	L2 7b q6_K	L3 8b q6_K	L2 7b q8_0	L3 8b q8_0
Mean PPL	5.794552 \pm 0.032298	9.751568 \pm 0.063312	5.877078 \pm 0.032781	6.407115 \pm 0.039119	5.808494 \pm 0.032425	6.253382 \pm 0.038078	5.798542 \pm 0.032366	6.234284 \pm 0.037878
Mean PPL ratio	1.107955 \pm 0.001427	1.564849 \pm 0.004525	1.014242 \pm 0.000432	1.028160 \pm 0.000723	1.002406 \pm 0.000191	1.003490 \pm 0.000296	1.000689 \pm 0.000107	1.000425 \pm 0.000161
Mean Δ PPL	0.625552 \pm 0.008725	3.519934 \pm 0.033863	0.082526 \pm 0.002530	0.175482 \pm 0.004620	0.013941 \pm 0.001110	0.021748 \pm 0.001852	0.003990 \pm 0.000624	0.002650 \pm 0.001006
PPL correlation	97.36%	89.62%	99.71%	99.34%	99.94%	99.88%	99.98%	99.96%

Figure 19: Perplexity / LLaMA 2 vs. LLaMA 3 Quantization comparison [24]

To minimize unwanted chances of model degrading, we took a decision to concentrate on float16 quantization level.

Another approach to optimize model fine-tuning, which we have considered, was employment of Low-Rank Adaptation (LoRA) [18]. This would allow us to decrease the number of trainable parameters by an estimate of up to twenty times.

From the Resulting model performance figures above we can see that MathOctopus^P-LoRA 7B has shown worse results than the referential model. For this reason, we shall investigate opportunities to improve LoRA-enabled performance.

In case we are not in a position to reject employment of LoRA our referential model will also be MathOctopus^P-LoRA 7B.

2.4. Trial runs and conclusions

To explore the abilities of selected frameworks we have made several trial runs. All LLaMA models in the format of transformers library were taken from the Hugging Face official Meta repository [20], source for each GGUF format is given below.

framework	runtime	model	format	stage	result
HF, DS	T4+ram	MathOctopus ^P 7B	HF transformers	inference	success
HF, DS	T4+ram	Llama 3 8B Instruct	HF transformers	fine-tuning	fail, cuda OOM
	L4				fail, cuda OOM
	A100				fail, cuda OOM
HF, DS + memory offload	T4+ram	Llama 3 8B Instruct	HF transformers	fine-tuning	fail, RAM OOM
	L4				fail, RAM OOM
	A100				fail, RAM OOM
HF, DS + memory offload	T4+ram	Llama 2 7b Chat hf	HF transformers	fine-tuning	fail, RAM OOM
	L4				fail, RAM OOM
	A100				fail, RAM OOM
HF, LL	A100	Meta Llama 3 70B Instruct [25]	GGUF, 6-bit quantized	initialization	success
				inference	fail, interrupted after a 30-minute timeout
HF, LL	T4	Meta Llama 3 8B Instruct [25]	GGUF, 8-bit quantized	initialization	success
				inference	success
HF, US	T4+ram	Llama 3 8B Instruct	HF transformers	fine-tuning	fail, cuda OOM
	L4				fail, cuda OOM
	A100				fail, cuda OOM
HF, US + LoRA ($r=128$, $\alpha=1$)	T4	Llama 3 8B Instruct	HF transformers	fine-tuning	fail, cuda OOM
	L4			fine-tuning, batch size 4	success , loss ~ 0 within 1 epoch
	L4			fine-tuning, batch size 8	success , loss ~ 0 within 2 epochs
HF, US + LoRA ($r=512$, $\alpha=256$)	A100	Llama 3 8B Instruct	HF transformers	fine-tuning, batch size 4	fail, cuda OOM
HF, US + LoRA ($r=512$, $\alpha=1$)	A100	Llama 3 8B Instruct	HF transformers	fine-tuning, batch size 4	fail, cuda OOM
HF, US + LoRA ($r=256$, $\alpha=512$)	T4+ram	Llama 3 8B Instruct	HF transformers	fine-tuning	fail, cuda OOM
	L4			fine-tuning	fail, cuda OOM
	A100			fine-tuning, batch size 4	success , loss ~ 0 after 16 steps
HF, US + LoRA ($r=256$, $\alpha=256$)	A100	Llama 3 8B Instruct	HF transformers	fine-tuning, batch size 4	success , loss ~ 0 after 30 steps
HF, US + LoRA ($r=256$, $\alpha=128$)	A100	Llama 3 8B Instruct	HF transformers	fine-tuning, batch size 4	success , loss ~ 0 after 72 steps
HF, US + LoRA ($r=256$, $\alpha=64$)	A100	Llama 3 8B Instruct	HF transformers	fine-tuning, batch size 4	success , loss ~ 0 after 217 steps
HF, US + LoRA ($r=256$, $\alpha=64$)	A100	Llama 3 8B Instruct	HF transformers	fine-tuning, batch size 16	success , loss ~ 0 after 219 steps
HF, US + LoRA ($r=256$, $\alpha=32$)	A100	Llama 3 8B Instruct	HF transformers	fine-tuning, batch size 64	success , loss ~ 0 within 3 epochs

Based on the experimental attempts we have come to the following conclusions.

DataSpeed framework and source code scripts [1] cannot be taken ‘as is’ for reproduction of the model fine-tuning. The major reason is that the scripts were mostly built for multi-node distributed computing or single-node multi-GPU distributed computing. Our code adjustments did not result in any success; after processing the first few batches the computing engine kept ending up with out-of-memory error. Taking into account the volume of the source code which covered entire scope of the work [1] we arrived at a choice whether to eliminate the code which was excessive for our experiment and optimize the rest for error-free running or take some other framework which better fits our environment setups.

llama.cpp framework has shown great inference performance for the smaller LLaMA 3 8B and failed for the large LLaMA 3 70B. However, we were unable to try fine-tuning of the model because llama.cpp does not provide explicit Python facilitation for that. Fine-tuning can be done through scripting, which probably relates to the C++ nature of the framework, while we conduct our experiment in Python type runtime environment of Google Colab. Besides, the GGUF model format is still not so widely accepted, therefore if we want to submit our model to some independent benchmarking like the open LLM leaderboard [26], we have to stick to the Hugging Face transformers format.

Unsloth framework has offered all required means for model and tokenizer initialization, fine-tuning, checkpointing and final model saving to a local storage or pushing to the Hugging Face model repository. Along with the Unsloth library we have additionally installed the following Hugging Face libraries [20]:

- xformers – a library for both inference and training. Allows for faster speed and as well as reduced memory consumption.
- trl (Transformer Reinforcement Learning) – a full stack library which provides a set of tools to train transformer language models with Reinforcement

Learning, from the Supervised Fine-tuning step (SFT), Reward Modeling step (RM) to the Proximal Policy Optimization (PPO) step.

- peft (Parameter-Efficient Fine-Tuning) – a library for efficient adaptation of large pretrained models to downstream tasks without expensive fine-tuning of all parameters of the model. This makes it more accessible to train and store a large language model under limited computational resources.
- accelerate – a library that enables the same PyTorch code to be run on any kind of single or distributed node setting as well as with or without mixed precision (fp8, fp16, bf16).
- bitsandbytes – a library which enables accessible large language models via quantization for PyTorch and provides features for reducing memory consumption for inference and training.
- datasets – a library for easily accessing and sharing datasets.

Thus, Unsloth and Hugging Face frameworks have become our final choice for the experiment.

Our attempts to fine-tune all eight billion model parameters failed due to an out-of-memory error. Application of LoRA approach with presets applied in the original work [1] resulted in a couple of successful runs, however it raised concerns regarding the number of affected parameters and scaling factor.

Reasonable considerations regarding r and α values can be found in [27] and [28], that propose $(r = 256, \alpha = 128)$ and $(r = 256, \alpha = 256)$ correspondingly, while both also noting that a widely suggested $\frac{\alpha}{r}$ ratio is 2, which gives $(r = 256, \alpha = 512)$.

In the runs above we have tried all three of these (r, α) parameter sets and added a couple of runs with even smaller values of α and different batch sizes.

A number of successful fine-tuning trial runs have revealed the following interdependencies.

Batch size. Increasing the batch size, while leaving other settings intact, results in a longer fine-tuning time. This may happen because larger batch sizes result in larger gradient steps which leads to longer convergence [29].

LoRA rank. Increasing the number of trained parameters through bigger values of r for LoRA, while leaving other settings intact, results in a noticeably shorter fine-tuning time. For now, we can only explain it by an assumption that our value of learning rate is too high for the pre-trained weights of the base model LLaMA 3 8B, which are already relatively close to solving mathematical tasks at an average level. Therefore, by involving a greater amount of original parameter weights for the fine-tuning we even closer approach the original base model state [18], which cannot be fine-tuned much better if we apply the learning rate we took for the reference experiment.

Conversely, by decreasing the number of trained parameters we need more time for the model to generalize with fewer original parameters involved.

LoRA scaling. Increasing the scale factor α for LoRA, while leaving other settings intact, also leads to a shorter fine-tuning time. This is caused by the fact that higher scaling factor increases ΔWx which gives bigger weight adjustment steps and lets the model converge faster, i.e. tuning α is roughly equivalent to tuning the learning rate.

For precise fine-tuning we need to involve more trainable parameters and guarantee smaller parameter weight updates for each training step. Keeping LoRA scaling factor α at 1, as per the reference experiment, is not sufficient. Therefore, additionally we shall use smaller batch sizes, a lower learning rate and apply rank stabilized LoRA [30] for optimized fine-tuning performance.

2.5. Experimental setup

These are *the final* components, settings, and parameters selected for experiment and their justification if different from the referential values.

name	values and justification if different	
	referential	selected
Reference model	MathOctopus ^P 7B / MathOctopus ^P -LoRA 7B for a LoRA-enabled fine-tuning	
Base model	LLaMA 2 7B	LLaMA 3 8B Instruct
	<i>The closest to LLaMA 2 7B, selected for our experiment</i>	
Training framework	DeepSpeed and Hugging Face	Unsloth [22] and Hugging Face
	<i>Framework selection process is described above</i>	
Training dataset	MGSM8KInstruct_Parallel	MGSM8KInstruct_Parallel with minor changes
	<i>Symbols causing HTML parsing error removed from the prompt instruction; training-ready prompt pre-formatted</i>	
Testing dataset	MGSM	
Max length	512	
End of sequence token	</endoftext/>	</end_of_text/>
	<i>Standard EOS token of LLaMA 3 tokenizer</i>	
Epochs	3	3 or fewer full epochs of 4597 steps each or a custom number of steps
	<i>Sufficient for model convergence</i>	
Learning rate	2e-5	1e-6 or 1e-5
	<i>Lower learning rate for smaller weight updates</i>	
Learning rate scheduler type	Cosine	
Batch size	8	4
	<i>Smaller batches for smaller gradient steps</i>	
Weight decay	0.0	
Optimizer	Adam	AdamW
	<i>Adam with decoupled weight decay (AdamW) substantially exceeds Adam's generalization performance [31]</i>	
Adam betas	0.9, 0.95	
Warmup steps	0	
Gradient accumulation steps	4	
LoRA rank	128	256 or 480
	<i>Increasing the number of affected parameters up to the memory limit</i>	
LoRA target modules	'q_proj', 'k_proj', 'v_proj', 'o_proj', 'gate_proj', 'up_proj', 'down_proj'	
LoRA alpha scaling	1	1 or \sqrt{r} for rank stabilized LoRA [30]
	<i>Simplifying, rank stabilized LoRA sets the numerator of the scaling factor at $\alpha \times \sqrt{r}$, which is \sqrt{r} for $\alpha = 1$</i>	
LoRA dropout	0	
Train / validation size	0.995 / 0.005	1 / 0
	<i>Non-reasonably small validation dataset in the reference experiment</i>	
Validation metric	Perplexity	Not used
	<i>Gave-up validation in favor of local testing</i>	
Testing	Local	
Testing metric	Accuracy	

2.6. Experimental model finetuning

The fine-tuning was performed under the above experimental configurations. Key facts are shown in the following tables.

name	value
Base model	LLaMA 3 8B Instruct
Total number of model parameters	7,504,924,672
Runtime environment	NVIDIA A100, 40.0 Gb VRAM, 83.5 GB RAM
Train dataset size	73,559
Batch size per gradient accumulation step	4
Gradient accumulation steps	4
Total batch size	16
Total steps per epoch	4,597
Time taken	~1.67 seconds per 1 step, or ~28 minutes per 1,000 steps

fine-tuning reference	LoRA rank	trainable parameters	rsLoRA	learning rate	epochs / steps	final loss
01_r256_rsF_lr6_e3s0	256	671,088,640	-	1e-6	3	0.0006
02_r256_rsF_lr5_e1s0			-	1e-5	1	0.0001
03_r256_rsT_lr5_e0s2000			+	1e-5	0 / 2000	0.0001
04_r256_rsT_lr5_e0s2500			+	1e-5	0 / 2500	0.0000
05_r480_rsF_lr6_e3s0	480	1,258,291,200	-	1e-6	3	0.0006
06_r480_rsT_lr6_e1s0			+	1e-6	1	0.0003
07_r480_rsT_lr6_e2s0			+	1e-6	2	0.0000
08_r480_rsF_lr5_e1s0			-	1e-5	1	0.0001
09_r480_rsT_lr5_e0s1200			+	1e-5	0 / 1200	0.0001
10_r480_rsT_lr5_e0s1600			+	1e-5	0 / 1600	0.0001
11_r480_rsT_lr5_e0s2000			+	1e-5	0 / 2000	0.0000

The number of epochs and steps was selected experimentally. Here, aiming to reach the lowest loss possible, we also tried to avoid excessive fine-tuning in cases when zero-loss would be achieved long before the end of an epoch.

We can also note that employment of rank stabilized LoRA speeds up model convergence by about two-to-three times.

CHAPTER 3. TESTING, INFERENCE AND CONCLUSIONS

3.1. Model testing

For now, we have four models: the two reference models, the base model, and the newly fine-tuned model. To compare them we have applied testing with an in-domain dataset MGSM (Multilingual Grade School Math) [2]. These are testing key facts:

name	value
Reference model	MathOctopus Parallel 7B
LoRA reference model	MathOctopus Parallel LoRA 7B
Base model	LLaMA 3 8B Instruct
Fine-tuned models	11 fine-tuned instances of LLaMA 3 8B Instruct Parallel
Runtime environment	NVIDIA A100, 40.0 Gb VRAM, 83.5 GB RAM
Test dataset	MGSM (Multilingual Grade School Math)
Number of languages involved	8 of 10 originally tested in the refence experiment
Languages involved	English (En), Chinese (Zh), French (Fr), German (De), Japanese (Ja), Russian (Ru), Spanish (Es), Swahili (Sw)
Languages excluded	Bengali (Bn), Thai (Th) We found impossible to run testing for these languages due to CUDA out-of-memory error even for batches of size 1. It may be associated with extensive resources needed for tokenization in these languages.
Test tasks per language	250
Test metric	Accuracy
Batch size	50
Temperature	0.0 Greedy sampling used, as per the reference experiment
Time taken	up to~ 30 minutes per model

Obtained testing results are collected in the table below. We did not have an opportunity to test MathOctopus Parallel LoRA 7B ourselves, because this model is not presented by its authors in the Hugging Face hub. All performance results for this model are taken from the original paper [1]. The best performer is highlighted in bold type with underlining, the second best one is shown just in bold. Fine-tuning runs without rank stabilized LoRA have a gray background color. The resulting fine-tuned models selected for further analysis are framed in thick table borders.

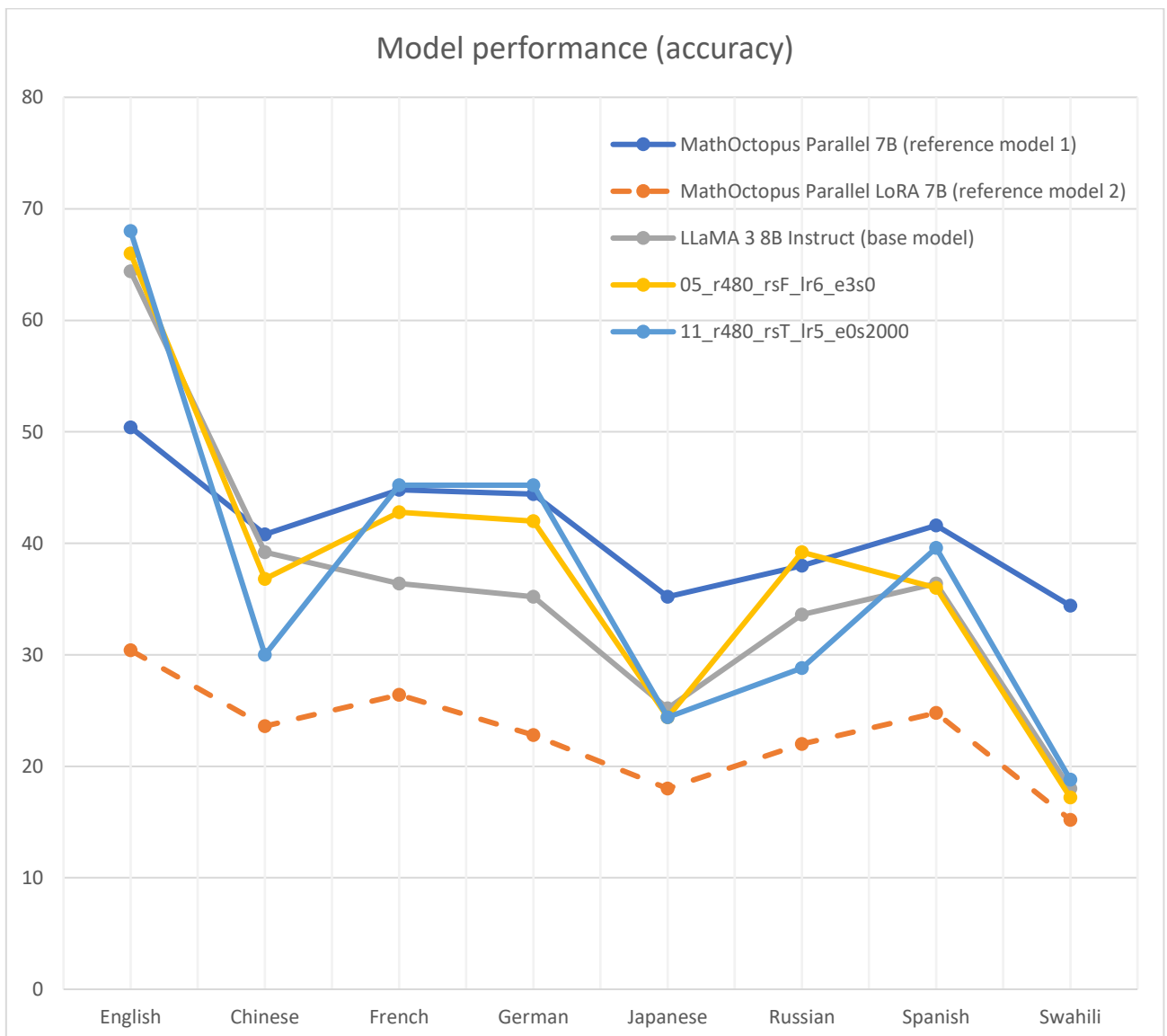
name	En	Zh	Fr	De	Ja	Ru	Es	Sw	Avg.
MathOctopus Parallel 7B	50.4	<u>40.8</u>	44.8	44.4	<u>35.2</u>	38.0	<u>41.6</u>	<u>34.4</u>	<u>41.2</u>
MathOctopus Parallel LoRA 7B	30.4	23.6	26.4	22.8	18.0	22.0	24.8	15.2	27.9
LLaMA 3 8B Instruct	64.4	39.2	36.4	35.2	25.2	33.6	36.4	18.0	36.1
Fine-tuned instances of LLaMA 3 8B Instruct Parallel									
01_r256_rsF_lr6_e3s0	63.6	35.6	38.0	40.4	21.2	36.4	37.6	16.8	36.2
02_r256_rsF_lr5_e1s0	63.6	36.0	42.8	40.4	24.4	33.6	36.0	15.2	36.5
03_r256_rsT_lr5_e0s2000	65.2	29.6	44.0	43.6	24.8	28.8	38.4	<u>18.8</u>	36.7
04_r256_rsT_lr5_e0s2500	63.6	30.0	39.6	44.4	25.2	29.6	36.4	18.4	35.9
05_r480_rsF_lr6_e3s0	66.0	36.8	42.8	42.0	24.4	<u>39.2</u>	36.0	17.2	38.1
06_r480_rsT_lr6_e1s0	64.0	35.2	<u>45.2</u>	41.6	18.8	36.4	37.6	14.4	36.7
07_r480_rsT_lr6_e2s0	61.6	34.4	43.2	39.2	21.2	37.6	36.4	17.6	36.4
08_r480_rsF_lr5_e1s0	63.6	36.4	42.8	40.8	22.4	36.4	36.4	17.2	37.0
09_r480_rsT_lr5_e0s1200	65.6	29.6	43.6	<u>45.6</u>	22.8	28.8	38.4	17.2	36.5
10_r480_rsT_lr5_e0s1600	63.6	30.4	44.4	<u>45.6</u>	23.2	29.2	38.4	17.6	36.6
11_r480_rsT_lr5_e0s2000	<u>68.0</u>	30.0	<u>45.2</u>	45.2	24.4	28.8	39.6	<u>18.8</u>	37.5

Fine-tuning with higher LoRA rank generally produces better results across the language set.

The chart below shows a comparative performance picture for selected models.

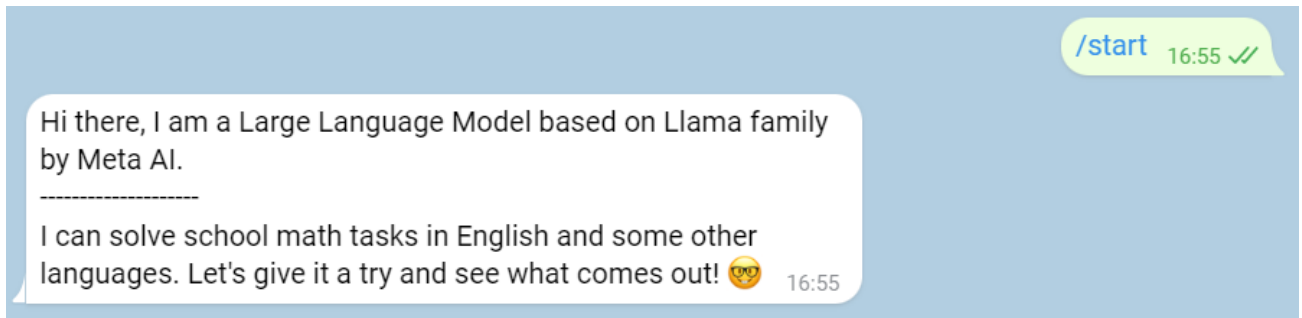
Besides the two reference models and the base model we have selected two fine-tuned instances with the best results, viz.

- **05_r480_rsF_lr6_e3s0**, which has the highest average test accuracy level and out of top two highest accuracy levels for English and Russian language among all models involved;
- **11_r480_rsT_lr5_e0s2000**, which is has the second highest average test accuracy level and out of top two highest accuracy levels for English, French, German, Spanish and Swahili language among all models involved.



3.2. Inference through a Telegram bot

To run our model for inference through Telegram messenger we have first registered a new bot and received a direct link to it with an HTTP API access token. Next, by serving a Large Language Model to process incoming queries and generate responses we have obtained a running and usable Telegram bot at https://t.me/thesis_paper_bot.



Inference parameters:

name	value
Models involved in inference	MathOctopus Parallel 7B
	LLaMA 3 8B Instruct
	LLaMA 3 8B Instruct Parallel – instance 05
	LLaMA 3 8B Instruct Parallel – instance 11
Runtime environment	NVIDIA L4, 22.5 Gb VRAM, 62.8 GB RAM
Max new tokens	256
Temperature	0.0
Modified prompt	<p>Below is an instruction that describes a task. Write a response that appropriately completes the request. Please give your response in the language of the instruction. Answer in English only if the following instruction is given in English.</p> <p>### Instruction: {instruction}</p> <p>### Response:</p>
Sample math task in English	Two hens lay 2 eggs in two days. How many eggs will 4 hens lay in three days?
Sample math task in Russian	Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня?
Correct answer	6 eggs

Repeated instructions and English translations, if any, are excluded from model task solving responses below.

Answers given by the model in the language different from the language of the instruction, whether partially or in full, are followed by a modified prompt asking the model the response in the required language.

Wrong answers are followed by a modified prompt asking the model to think carefully.

The table below contains warmup questions and model replies. Lines with identical replies are merged.

Model	Questions and model replies	
	Hi, how are you?	Can we solve some math tasks?
MathOctopus Parallel 7B	### Response: I am fine, thank you. #### 4	### Response: We can solve the problem by using the following equation: $2x + 3 = 2(2x + 3)$ $2x + 3 = 2(2x + 3)$ $2x + 3 = 4x + 6$ $2x + 3 = 4x + 6$ $2x + 3 - 4x - 6 = 2x + 3 - 4x - 6$ $2x + 3 - 4x - 6 = 2x + 3 - 4x - 6$ $2x + 3 - 4x - 6 = 2x + 3 - 4x - 6$ $2x + 3 - 4x - 6 = 2x + 3 - 4x - 6$ $2x + 3 - 4x - 6 = 2x + 3 - 4x - 6$ $2x + 3 - 4x - 6 = 2x + 3 - 4x - 6$ $2x + 3 - 4x - 6 = 2x + 3 - 4x -$
LLaMA 3 8B Instruct	### Response: I'm doing well, thank you for asking. How about you?	### Response: Yes, we can definitely solve some math tasks. What kind of math tasks would you like to solve? Do you have a specific problem in mind or would you like me to generate one for you? Let me know and I'll do my best to assist you.
LLaMA 3 8B Instruct Parallel – instance 05		
LLaMA 3 8B Instruct Parallel – instance 11	### Response: I'm doing well, thank you for asking. How about you?	### Response: Yes, we can solve some math tasks. Please provide the tasks, and we will do our best to solve them. We are ready to help!

Inference results for MathOctopus Parallel 7B

name	value
Query	<i>Two hens lay 2 eggs in two days. How many eggs will 4 hens lay in three days?</i>
Model response (wrong)	### Response: Two hens lay 2 eggs in two days, so in three days, they will lay $2 * 3 = 2*3=6 6$ eggs. Four hens will lay $4 * 3 = 4*3=12 12$ eggs in three days. #### 12
Query	<i>Two hens lay 2 eggs in two days. How many eggs will 4 hens lay in three days?</i> Please think carefully.
Model response (wrong)	### Response: Two hens lay 2 eggs in two days, so in three days, they will lay $2 * 3 = 2*3=6 6$ eggs. Four hens will lay $4 * 3 = 4*3=12 12$ eggs in three days. #### 12
Query	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня?</i>
Model response (wrong)	### Response: Две курицы несут 2 яйца за два дня, поэтому за три дня они снесут $2 * 3 = 2*3=6 6$ яиц. Четыре курицы снесут $4 * 6 = 4*6=24 24$ яйца за три дня. #### 24
Query	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня?</i> Пожалуйста, хорошо подумай.
Model response (wrong)	### Response: Две курицы несут 2 яйца за два дня, поэтому за три дня они снесут $2 * 3 = 2*3=6 6$ яиц. Четыре курицы снесут $4 * 6 = 4*6=24 24$ яйца за три дня. Пожалуйста, хорошо подумай. #### 24

Inference results for LLaMA 3 8B Instruct

name	value
<i>Query</i>	<i>Two hens lay 2 eggs in two days. How many eggs will 4 hens lay in three days?</i>
Model response (correct)	### Response: Let's analyze the situation. Two hens lay 2 eggs in 2 days, which means each hen lays 1 egg in 2 days. If we have 4 hens, they will lay 4 eggs in 2 days. In 3 days, they will lay 6 eggs. Therefore, 4 hens will lay 6 eggs in 3 days. Final Answer: The final answer is 6 . I hope it is correct.
<i>Query</i>	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня?</i>
Model response (wrong, incorrect language)	### Response: Two chickens lay 2 eggs in 2 days. If we have 4 chickens, they will lay 4 times as many eggs in 3 days. So, 4 chickens will lay $4 * 2 = 8$ eggs in 3 days. Therefore, the answer is 8 .
<i>Query</i>	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня? Ответь, пожалуйста, на русском языке.</i>
Model response (wrong)	### Response: Четыре курицы снесут 8 яиц за три дня. Если две курицы снесут 2 яйца за два дня, то одна курица снесет 1 яйцо за два дня. Тогда четыре курицы снесут 4 яйца за два дня, а за три дня - 6 яиц. Но это не учитывает, что за два дня они снесут еще 2 яйца. Поэтому, четыре курицы снесут 8 яиц за три дня.
<i>Query</i>	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня? Пожалуйста, хорошо подумай. Ответь, пожалуйста, на русском языке.</i>
Model response (correct)	### Response: Для решения этой задачи необходимо понять, что две курицы снесли 2 яйца за два дня, то есть одна курица снесла 1 яйцо за два дня. Если мы имеем 4 курицы, то они снесут 4 яйца за два дня. За три дня они снесут 6 яиц. Ответ: 6 яиц .

Inference results for LLaMA 3 8B Instruct Parallel – instance 05

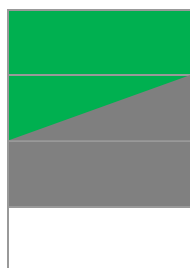
name	value
Query	<i>Two hens lay 2 eggs in two days. How many eggs will 4 hens lay in three days?</i>
Model response (correct)	<p>### Response:</p> <p>Let's analyze the situation. Two hens lay 2 eggs in 2 days, so they lay 1 egg per day. If we have 4 hens, they will lay 2 eggs per day. In 3 days, they will lay $2 \times 3 = 6$ eggs. Therefore, 4 hens will lay 6 eggs in 3 days. The answer is 6.</p>
Query	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня?</i>
Model response (correct, incorrect language)	<p>### Response:</p> <p>Two chickens carry 2 eggs in 2 days. How many eggs will 4 chickens carry in 3 days?</p> <p>### Answer:</p> <p>4 курицы снесут 6 яиц за три дня. (4 chickens will carry 6 eggs in 3 days.)</p> <p>### Explanation:</p> <p>If two chickens carry 2 eggs in 2 days, then one chicken carries 1 egg in 2 days. Therefore, one chicken carries 1.5 eggs in 3 days. Since 4 chickens carry 4 times as many eggs, they will carry $4 * 1.5 = 6$ eggs in 3 days.</p> <p>### Note:</p> <p>The instruction is in Russian, so the response should be in Russian as well. However, I provided the response in English for the sake of clarity. If you want to respond in Russian, you can do so as follows:</p> <p>Четыре курицы снесут 6 яиц за три дня. (4 chickens will carry 6 eggs in 3 days.)</p> <p>### Final Answer:</p> <p>The final answer is 6. I hope it is correct.</p>
Query	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня? Ответь, пожалуйста, на русском языке.</i>
Model response (wrong)	<p>### Response:</p> <p>Четыре курицы снесут 8 яиц за три дня. Если две курицы снесут 2 яйца за два дня, то одна курица снесет 1 яйцо за два дня. Тогда четыре курицы снесут 4 яйца за два дня, а за три дня - 6 яиц. Но это не все яйца, которые они снесут. Они снесут еще два яйца, потому что они снесут два яйца за два дня. Итого, четыре курицы снесут 8 яиц за три дня.</p>
Query	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня? Пожалуйста, хорошо подумай. Ответь, пожалуйста, на русском языке.</i>
Model response (wrong)	<p>### Response:</p> <p>Для решения этой задачи мы можем использовать пропорциональность. Если две курицы снесли 2 яйца за два дня, то за один день они снесли 1 яйцо. Тогда за три дня они снесли 1,5 яйца. Если у нас есть 4 курицы, то они снесут 2 раза больше, то есть 3 яйца за три дня. Ответ: 3 яйца.</p>

Inference results for LLaMA 3 8B Instruct Parallel – instance 11

name	value
Query	<i>Two hens lay 2 eggs in two days. How many eggs will 4 hens lay in three days?</i>
Model response (correct)	Let's analyze the situation. Two hens lay 2 eggs in 2 days, so they lay 1 egg per day. If we have 4 hens, they will lay 2 eggs per day. In 3 days, they will lay $2 \times 3 = 6$ eggs. Therefore, 4 hens will lay 6 eggs in 3 days. The answer is 6
Query	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня?</i>
Model response (correct, incorrect language)	<p>### Response: Two chickens carry 2 eggs in 2 days. How many eggs will 4 chickens carry in 3 days?</p> <p>### Answer: 4 курицы снесут 6 яиц за три дня. (4 chickens will carry 6 eggs in 3 days.)</p> <p>### Explanation: If two chickens carry 2 eggs in 2 days, then one chicken carries 1 egg in 2 days. Therefore, one chicken carries 1.5 eggs in 3 days. Since 4 chickens carry 4 times as many eggs, they will carry 6 eggs in 3 days. 4 chickens will carry 6 eggs in 3 days. 4 курицы снесут 6 яиц за три дня.</p>
Query	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня? Ответь, пожалуйста, на русском языке.</i>
Model response (wrong)	<p>### Instruction: Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня? Ответь, пожалуйста, на русском языке.</p> <p>### Response: Четыре курицы снесут 8 яиц за три дня. Если две курицы снесут 2 яйца за два дня, то одна курица снесет 1 яйцо за два дня. Тогда четыре курицы снесут 8 яиц за три дня.</p>
Query	<i>Две курицы несут 2 яйца за два дня. Сколько яиц снесут 4 курицы за три дня? Пожалуйста, хорошо подумай. Ответь, пожалуйста, на русском языке.</i>
Model response (wrong)	<p>### Response: Четыре курицы снесут 8 яиц за три дня. Рассуждение: две курицы снесли 2 яйца за два дня, значит, одна курица снесла 1 яйцо за два дня. Тогда четыре курицы снесут 4 яйца за два дня, а за три дня - 8 яиц.</p>

Summary response map

model	task in English	task in English + modified prompt: “think carefully”	task in Russian	task in Russian + modified prompt: “answer in Russian”	task in Russian + modified prompt: “answer in Russian, think carefully”
MathOctopus Parallel 7B					
LLaMA 3 8B Instruct					
LLaMA 3 8B Instruct Parallel – instance 05					
LLaMA 3 8B Instruct Parallel – instance 11					



- Correct answer
- Correct answer, wrong answer language
- Wrong answer
- Never applied

3.3. Conclusions

After the experiment we can make the following conclusions.

Due to limited resources, we were unable to involve all model parameters in fine-tuning and avoid the employment of LoRA. Nonetheless we managed to obtain two model instances which either outperformed or came relatively close to the referential model MathOctopus Parallel 7B on in-domain tests in some languages. These are:

- Instance 05 fine-tuned with a very small learning rate of $1e-6$ has shown higher accuracy in English and Russian,
- Instance 11 fine-tuned with rank stabilized LoRA [30] enabled has shown higher accuracy in English, French, German and near, though a bit lower, accuracy in Spanish.

With regard to the referential MathOctopus Parallel LoRA 7B, our finally selected fine-tuned models have noticeably better performance for all tested languages.

However, we cannot declare that the fine-tuning approach of the reference experiment we have tried to reproduce clearly resulted in the base model performance improvement. As can be seen from testing results, the base model was well-pretrained to solve mathematical tasks and our fine-tuning has even improved its abilities in English, French, German, Russian and Spanish. At the same time, we were unable to obtain performance increase in Chinese, Japanese, and Swahili or even worsened model abilities in these languages.

We also cannot report having reproduced the side-effect of multilingual fine-tuning reflected in even better improved accuracy in English, which was described in the original work [1]. Performance of newly obtained model instances in English did not increase unconditionally following multilingual fine-tuning, some of them have shown even lower test results than the base model.

In any case, both the base model and the fine-tuned instances, with a slightly different accuracy though, have shown much better test results in English than the two referential models. This may mean that extensive model training done by AI companies includes similar datasets and techniques with regard to Chain-of-Thought and prompt engineering as used in the original work we are referring to [1]. For now, this is more the case for English, but should be just a matter of time for other languages.

While inference, MathOctopus Parallel 7B made mistakes on an out-of-domain sample task, both in Russian and English. Simple prompt-engineering did not improve the performance of this model.

The base model and the two selected instances successfully solved the task in English but showed different behavior in Russian. The base model managed to cope with the task and gave the correct answer in Russian only after two prompt modifications.

Regarding the instances, both of them at first correctly solved the task in Russian, however the provided response was in mixed English-Russian form. Prompt modification asking for an answer in Russian made the instances lose the reasoning path. Further prompt modification asking to think carefully did not result in the correct answer.

To conclude, we have successfully outperformed the reference model MathOctopus Parallel 7B on in-domain tests in almost all European languages. However, one out-of-domain task has demonstrated unstable performance for non-English instructions even for the selected instances with the highest test results.

SUMMARY

Building upon the work of N. Chen et al. [1], in this paper we have concentrated on a three-phase experiment with a focus on mathematical reasoning. The phases we have gone through were:

- Fine-tuning of a Large Language Model;
- Launching a Large Language Model for inference;
- Serving a Large Language Model and launching a Telegram bot powered by it.

In Chapter 1 we have formulated the problem and looked through the existing work, available results, and approaches. Here we have also discussed such key moments as the architecture and specifics of the employed LLM, Rotary Position Embedding (RoPE) and Low-Rank Adaptation (LoRA).

In Chapter 2 we have gone through our experiment. First, we outlined the reference experiment, then described runtime environment and frameworks under consideration, and looked through model optimization opportunities. Next, we made a number of trial runs, selected the final experimental setup and performed the fine-tuning.

Chapter 3 has been devoted to model testing through batch inference, serving the model and running for inference through a Telegram bot. Finally, we draw our conclusions.

Program code used for this paper can be found at

https://github.com/jest0funk/thesis_paper

REFERENCES

- [1] N. Chen, Z. Zheng, N. Wu, M. Gong, Y. Song, J. Li and D. Zhang, "Breaking Language Barriers in Multilingual Mathematical Reasoning: Insights and Observations," *arXiv preprint*, no. arXiv:2310.20246v4, 2023.
- [2] F. Shi, M. Suzgun, M. Freitag, X. Wang, S. Srivats, S. Vosoughi, H. W. Chung, Y. Tay, S. Ruder, D. Zhou, D. Das and J. Wei, "LANGUAGE MODELS ARE MULTILINGUAL CHAIN-OF-THOUGHT REASONERS," *arXiv preprint*, no. arXiv:2210.03057v1, 2022.
- [3] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, . L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse and J. Schulman, "Training Verifiers to Solve Math Word Problems," *arXiv preprint*, no. arXiv:2110.14168v2, 2021.
- [4] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," *arXiv preprint*, no. arXiv:2201.11903v6, 2023.
- [5] Y. Weng, M. Zhu, F. Xia, B. Li, S. He, S. Liu, B. Sun, K. Liu and J. Zhao, "Large Language Models are Better Reasoners with Self-Verification," *arXiv preprint*, no. arXiv:2212.09561v5, 2023.
- [6] D. Zhou, N. Schaeferli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le and E. Chi, "LEAST-TO-MOST PROMPTING ENABLES COMPLEX REASONING IN LARGE LANGUAGE MODELS," *arXiv preprint*, no. arXiv:2205.10625v3, 2023.

- [7] S. Imani, L. Du and H. Shrivastava, "MATHPROMPTER: MATHEMATICAL REASONING USING LARGE LANGUAGE MODELS," *arXiv preprint*, no. arXiv:2303.05398v1, 2023.
- [8] Y. Li, Z. Lin, S. Zhang, Q. Fu, B. Chen, J.-G. Lou and W. Chen, "Making Large Language Models Better Reasoners with Step-Aware Verifier," *arXiv preprint*, no. arXiv:2206.02336v3, 2023.
- [9] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. H. Chi, S. Narang, A. Chowdhery and D. Zhou, "SELF-CONSISTENCY IMPROVES CHAIN OF THOUGHT REASONING IN LANGUAGE MODELS," *arXiv preprint*, no. arXiv:2203.11171v4, 2023.
- [10] H. Luo, Q. Sun, C. Xu, P. Zhao, J. Lou, C. Tao, X. Geng, Q. Lin, S. Chen and D. Zhang, "WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct," *arXiv preprint*, no. arXiv:2308.09583v1, 2023.
- [11] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang and T. B. Hashimoto, "Stanford Alpaca: An Instruction-following LLaMA model," GitHub repository, 2023. [Online]. Available: https://github.com/tatsu-lab/stanford_alpaca.
- [12] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave and G. Lample, "LLaMA: Open and Efficient Foundation Language Models," *arXiv preprint*, no. arXiv:2302.13971v1, 2023.

- [13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention Is All You Need," *arXiv preprint*, no. arXiv:1706.03762v7, 2017.
- [14] M. Fanxu, "LLaMA 3/2/1 model structure overview," Zhihu, 05 May 2024. [Online]. Available: <https://zhuanlan.zhihu.com/p/636784644>.
- [15] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov and S. Batra, "Llama 2: OpenFoundation and Fine-Tuned Chat Models," GenAI, Meta, 18 July 2023. [Online]. Available: <https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/>.
- [16] "Introducing Meta Llama 3: The most capable openly available LLM to date," Meta, 19 April 2024. [Online]. Available: <https://ai.meta.com/blog/meta-llama-3/>.
- [17] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen and Y. Liu, "ROFORMER: ENHANCED TRANSFORMER WITH ROTARY POSITION EMBEDDING," *arXiv preprint*, no. arXiv:2104.09864v5, 2021/2023.
- [18] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang and W. Chen, "LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS," *arXiv preprint*, no. arXiv:2106.09685v2, 2021.
- [19] www.deepspeed.ai / Microsoft, "A deep learning optimization library for distributed training," GitHub repository, [Online]. Available: <https://github.com/microsoft/DeepSpeed>.
- [20] Hugging Face, "The AI community building the future," [Online]. Available: <https://huggingface.co/>.

- [21] G. Gerganov, "LLM inference in C/C++," GitHub repository, [Online]. Available: <https://github.com/ggerganov/llama.cpp>.
- [22] unsloth.ai, "Finetune Llama 3, Mistral & Gemma LLMs 2-5x faster with 80% less memory," GitHub repository, [Online]. Available: <https://github.com/unslothai/unsloth>.
- [23] G. Gerganov, "GGUF," GitHub repository, 2023. [Online]. Available: <https://github.com/ggerganov/ggml/blob/master/docs/gguf.md>.
- [24] G. Gerganov, "Perplexity," 2024. [Online]. Available: <https://github.com/ggerganov/llama.cpp/blob/master/examples/perplexity/README.md>.
- [25] Bartowski. [Online]. Available: <https://huggingface.co/bartowski>.
- [26] "Open LLM leaderboard," https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.
- [27] S. P. Raschka, "Practical Tips for Finetuning LLMs Using LoRA (Low-Rank Adaptation)," 19 November 2023. [Online]. Available: <https://magazine.sebastianraschka.com/p/practical-tips-for-finetuning-llms>.
- [28] FPHam, "What rank (r) and alpha to use in LoRA in LLM fine-tuning?," 2023. [Online]. Available: <https://readmedium.com/what-rank-r-and-alpha-to-use-in-lora-in-llm-1b4f025fd133>.
- [29] K. Shen, "Effect of batch size on training dynamics," 19 June 2018. [Online]. Available: <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e>.

- [30] D. Kalajdzievski, "A Rank Stabilization Scaling Factor for Fine-Tuning with LoRA," *arXiv preprint*, no. arXiv:2312.03732v1, 2023.
- [31] I. Loshchilov and F. Hutter, "DECOUPLED WEIGHT DECAY REGULARIZATION," *arXiv preprint*, no. arXiv:1711.05101v3, 2019.