



UNIVERSITY OF BUCHAREST

**FACULTY OF
MATHEMATICS AND
INFORMATICS**



MASTER OF NATURAL LANGUAGE PROCESSING

Dissertation

LLMs for solving mathematics problems

Graduate

Grigore Dragoş-Gabriel

Supervisor

Conf. Dr. Uban Ana-Sabina

Bucharest

July 2025

Abstract

In the last decade, the progress was defined by the fast evolution of technology. The days when the technology wasn't an essential part of our life are long gone. Nowadays, life is characterized by the appearance of AI, not like a tool, but as day by day assistant.

This study presents how well smaller large language models (LLMs) on mathematical problems. The main two problems are the arithmetical calculations and the logical reasoning. The main goal of this project was to build a question-answering system by training a model on mathematical questions. The questions difficulty range is from fifth to twelfth grade school level.

This research compares the capabilities of a fine-tuned small model pretrained only on English language to compete with models more parameters. There is a comparison of different techniques that were used to observe what features made the biggest impact on the final result.

Abstract

În ultimul deceniu, progresul a fost definit de evoluția rapidă a tehnologiei. Zilele în care tehnologia nu era o parte esențială a vieții noastre au trecut de mult. Acum, viața este caracterizată de apariția inteligenței artificiale, nu ca și un instrument, ci ca un asistent zilnic.

Acest studiu prezintă cât de bine funcționează modelele lingvistice mari (LLM) de o dimensiune mai mică în rezolvarea problemelor matematice. Principalele două probleme sunt calculele aritmetice și raționamentul logic. Scopul principal al acestui proiect a fost de a construi un sistem de question-answering prin antrenarea unui model pe întrebări de matematică. Dificultatea întrebărilor variază între clasa a 5-a și a 12-a.

Această cercetare compară capacitățile unui model mic, ajustat, antrenat în prealabil doar pe limba engleză, pentru a concura cu modele cu mai mulți parametri. Există o comparație a diferitelor tehnici utilizate pentru a observa ce caracteristici au avut cel mai mare impact asupra rezultatului final.

Contents

1. Introduction	6
1.1. Context and Motivation	6
1.2. Problem Statement	6
1.3. Objectives of the Study	6
1.4. Chapters' overview	7
2. Related Work	9
2.1. Introduction to Mathematical Reasoning in Language Models	9
2.2. GSM8K as a Reasoning Benchmark	9
2.3. Comparative Performance on GSM8K	10
2.4. Evaluations on RoMath	12
3. Dataset	9
3.1. The GSM8K Dataset	9
3.2. Structure	9
3.3. Usage in This Project	10
3.4. Challenges and Considerations	10
3.5. Comparison with Other Datasets	11
3.6. The RoMath Dataset	12
4. Method and Approach	15
4.1. Introduction	15
4.2. Model Selection and Configuration	15
4.2.1. Choice of Model Architecture	15
4.2.2. Quantization and Memory Optimization	16
4.2.3. Tokenizer Setup	16
4.3. Fine-Tuning Methodology	16
4.3.1. LoRA: Parameter-Efficient Fine-Tuning	16
4.3.2. Training Configuration	17
4.4. Additional Considerations	18
5. Evaluation	20
5.1. Inference Procedure	20
5.2. Answer Extraction and Normalization	20
5.3. Evaluation Metrics	21
5.4. Results	21
5.5. Model Comparison	22
5.6. Comparison with Other Approaches	26

6. Conclusion.....	27
6.1. Summary of Findings.....	27
6.2. Limitations	27
6.3. Final Remarks.....	28
7. Bibliography.....	29

1. Introduction

1.1. Context and Motivation

The everyday use of technology defines contemporary life. The old days when people would use technology only in essential cases are long gone. Rapid evolution in artificial intelligence (AI) has led to significant transformations in how we access, process, and interact with information. These changes do not alter only the technological field but also the educational, social and cultural ones. Integrating AI into everyday life changed the way people acquire new information or make decisions. What was once the domain of science fiction - machines that understand and generate language, systems that learn from data, assistants that respond intelligently to human needs – has now become a normal part of modern society.

This shift in technology has really changed the way we interact with information. Instead of just reading or searching for answers on our own, we can now have real conversations with smart systems that can do things like summarize books, translate languages, or explain difficult concepts right away. These AI tools make learning and exploring ideas faster and in everyone's understanding. Whether you're a student, a professional, or just someone curious, AI is starting to feel less like a tool and more like a helpful assistant in everyday life.

1.2. Problem Statement

The considerable progress in the domain has led to significant changes in working with AI. Among the most impactful developments is the appearance of Large Language Models (LLMs) - such as GPT, T5, and Mistral - which have proved remarkable knowledge in understanding, generating, and manipulating human language. Although, there are many fields like: logical problems or law where AI struggles to give accurate answers.

1.3. Objectives of the Study

This study presents the implementation and evaluation of a question-answering (QA) application using a fine-tuned large language model (LLM). The model was designed to understand math questions in English and generate correct, step-by-step solutions that can be read by everyone who knows the language. The model is trained on the GSM8K dataset, made by OpenAI, which is a small collection of fifth to eighth grade problems. They are only algebra questions so it requires a good understanding of the words and arithmetic operations.

The educational value and linguistic complexity of the dataset make it an ideal resource for training and testing the mathematical capabilities of fine-tuned LLMs¹.

Another dataset that was used is in Romanian named RoMath from Cosma Adrian. This one is similar but has a more complex structure, with a range from fifth to twelfth grade school. The questions domain complexity is from basic algebra to geometry and even polynomials².

This work also addresses the challenges of adapting a general-purpose AI model to do a specific task. These challenges include choosing appropriate hyperparameters, handling input/output token limitations, and designing some relevant evaluation criteria. Both quantitative (e.g., accuracy, precision) and qualitative (e.g., logical coherence, clarity of explanation) metrics are used to verify the quality of the approach.

Beyond the technical scope, the relevance of this project lies in its dual impact: on one hand, it showcases the potential of modern AI technologies to support educational tools that are interactive and accessible; on the other hand, it provides insight of what LLMs can and can't do when applied to formal reasoning and structured problem-solving. Furthermore, the application developed here can be a foundation for future extensions, such as support for other subjects or different languages.

1.4. Chapters' overview

This paper is divided into six chapters. The first chapter explains why this topic was chosen and what the project aims to achieve.

Chapter 2 looks at other research in the area of math problem-solving with large language models. It discusses several important methods like chain-of-thought prompting, DUP (deeply understanding the problem) prompting, and the use of verifier models.

Chapter 3 focuses on the two datasets. It explains the characteristics of these datasets, why it is useful for testing math reasoning and compares it with other similar datasets.

¹ <https://huggingface.co/datasets/openai/gsm8k>

² <https://huggingface.co/datasets/cosmadrian/romath>

Chapter 4 describes the methods used in the project. It covers how the model was chosen, how the data was formatted and prepared, how prompts were designed, and how the model was fine-tuned using LoRA.

Chapter 5 talks about how the system was evaluated. It includes both the accuracy of the final answer and BERTScore evaluations that measure the quality of the answers, and a comparison between the best approach and other tests.

Finally, Chapter 6 is a review of the project that was done, pointing out some limitations of the system, and offering ideas for what could be improved in future work.

2. Related Work

2.1. Introduction to Mathematical Reasoning in Language Models

Trying to solve math problems, even the easier and logical ones, with large language models (LLMs) comes with a big set of challenges. Unlike tasks that involve looking up facts or understanding regular text, math questions often require a wider knowledge of the terms, a clear idea of logical steps, getting calculations right, and understanding the problem's context. Even models that are trained on some math data can still mess up calculations or misunderstand what the question is really asking. To resolve this problem, a lot of researchers started to search ways to make LLMs reasoning better. They've tried to find the best hyperparameters, train using bigger datasets, or change the questions and answers format. In this chapter, are presented some strategies that improved the capability of models to perform on logical tasks.

2.2. GSM8K as a Reasoning Benchmark

Although the GSM8K dataset was originally introduced by Cobbe et al. [1] to test how well language models handle mathematical reasoning, its real value today comes from how it's been used to track the progress of different reasoning strategies in LLMs. Since it contains thousands of grade-school-level word problems, GSM8K has become a go-to benchmark for measuring how well a model can handle step-by-step arithmetic problems.

What makes GSM8K stand out isn't just the types of math problems it contains, but also the variety of techniques that researchers have developed to improve model performance on it. For example, Cobbe et al. [1] showed that using a separate verifier model - which chooses the correct answer from a set of possible outputs - performed much better than just fine-tuning the model directly. This was one of the first signs that checking a model's answer after generation could make it more reliable.

Later on, Liu et al. [2] took a different approach by generating over 12 million synthetic math problems based on the GSM8K style and using them to fine-tune a smaller 1.3B model. What was surprising is that the smaller model ended up doing better than GPT-3.5 that created the data, showing how useful synthetic data can be in improving math reasoning.

Zhong et al. [3] introduced a paper their approach in which they got a remarkable accuracy of over 97%. They used a technique named DUP (Deeply Understanding the Problem) prompting. This technique structures the model's answers into clear, intermediate steps: understanding the question, extracting relevant information, and solving it. When paired with GPT-4, this set of techniques achieved an impressive 97.1% accuracy on GSM8K, near human-level performance, even beating some students with the age group corresponding to the math complexity. The results emphasize not only how useful the reasoning was structured, but also the role of a good semantic understanding, beyond the accuracy of the calculations.

These results position GSM8K as a benchmark dataset that changes the idea of what LLMs can do in a mathematical reasoning task.

2.3. Comparative Performance on GSM8K

As GSM8K dataset became one of the first tests a language model must take to see how well it performs on mathematical problems, there also have been used many techniques to enhance its performances. So researchers utilized prompt engineering techniques such as chain-of-thought (CoT) prompting, and self-consistency decoding. The results in the tables below show how good different models perform based on their answer accuracy.

As shown in the first table, Anthropic Claude 3 is the leader with a great accuracy of 95% in a zero-shot setting, followed closely by Google Gemini Ultra at 94.4%. OpenAI's GPT-4 is ranked third with 92% accuracy using a combination of supervised fine-tuning (SFT) and 5-shot chain-of-thought prompting.

Models like Anthropic Claude 2, Inflection 2, and Mistral Large also perform well taking into account the complexity of the dataset, reaching over 80% accuracy when using fewer shots. Few-shot learning (5-shot and 8-shot strategies) and self-consistency decoding appear to be key features in achieving higher performance. There were many other models in the leaderboard, but considering that they had just a little bit over 80% accuracy on the test set, I considered the results irrelevant.

Model	Accuracy	Methodology
Anthropic Claude 3	95%	Zero shot
Google Gemini Ultra	94.4%	Majority Vote, 32 Generations
OpenAI GPT-4	92%	SFT & 5-shot CoT
Anthropic Claude 2	88%	Zero shot
Google Gemini Pro	86.5%	Majority Vote, 32 Generations

Table 2.3.1: Accuracy as of March 5, 2025 (the last time the page Klu.ai was updated)[1]³

The second table breaks down results by model and methodology, providing deeper insight into how specific prompting and reasoning techniques impact performance. For example, GPT-3.5-Turbo achieves 78.9% accuracy with zero-shot CoT, which increases slightly with Least-to-Most prompting (77.5%) and zero-shot PS+ (79.3%). However, the DUP (Deep Understanding Prompting) method boosts performance significantly to 82.3%.

GPT-4, on the other hand, achieves much higher accuracy across all methods, with DUP pushing it to 97.1%, surpassing even previous state-of-the-art techniques like zero-shot CoT and zero-shot PS+.[3]

Model	Method	Accuracy
GPT-3.5-Turbo	Zero-shot CoT	78.9
	Least-to-Most	77.5
	Zero-shot PS+	79.3
	DUP	82.3
GPT-4	Zero-shot CoT	94.6
	Least-to-Most	92.1
	Zero-shot PS+	94.3
	DUP	97.1
GPT-3.5-Turbo	Manual-CoT	81.6
	Auto-CoT	80.2

Table 2.3.2: Accuracy of GPT using different reasoning techniques [3]⁴

³ <https://klu.ai/glossary/GSM8K-eval>

⁴ <https://arxiv.org/abs/2404.14963>

This comparison helps in two important ways. First, it puts the results of the model used in this project into context by showing how it stacks up against other popular models. Second, it draws attention to some of the most successful ideas that have helped improve performance in recent studies—like using verifier models, creating large amounts of synthetic training data, and breaking down problems into smaller reasoning steps. Even though there are many tables and results out there, I chose the ones shown here because they are the most recent and relevant, giving a clear picture of the latest progress in this area.

2.4. Evaluations on RoMath

The RoMath benchmark is a really important contribution for testing how well language models can solve math problems in Romanian. It includes different types of questions: some taken from the Romanian Baccalaureate exams, others from math competitions, and a large number of simpler, automatically generated problems. What makes this benchmark different is that it's created specifically for the Romanian language and school system. The authors show that models perform much worse when they rely on translated data instead of native Romanian examples, which proves how important it is to have local, culturally relevant datasets. They tested a variety of models, and even though some performed well, especially the ones fine-tuned on math and many of them struggled to give clear, step-by-step explanations in Romanian [14].

The results from the RoMath benchmark show some interesting patterns across different models. From what the authors of the dataset tested, some models give better results in zero-shot settings than in few-shot or fine-tuned configurations. For instance, RoMistral-7b-Instruct and RoLLaMA3-8b-Instruct, both Romanian models, achieved the highest F1 score in zero-shot (0.66, respectively 0.67), but their performance dropped noticeably in the 5-shot and fine-tuned settings (under 0.40). English models, like Qwen2-7B-Instruct, improved with fine-tuning, learning more about the dataset and language, even if it doesn't understand it. Other models that give math-specialized ones, deepseek-math-7b-instruct stands out, with a F1 score of 0.72 in the fine-tuned scenario [14].

3. Dataset

3.1. The GSM8K Dataset

GSM8K is the best dataset on this task. It is made up of 8,500 of 5th to 8th school level algebra problems, specifically curated to evaluate how well large language models handle step-by-step reasoning in natural language. Cobbe et al. [1] introduced it to verify and enhance the performance of LLMs in handling a task where reasoning is important. GSM8K differs from previous math datasets in that it emphasizes natural language explanations rather than just calculations. The dataset is perfect for training models that must clearly demonstrate their mathematical reasoning because each problem was written by a human and has a step-by-step solution [1][4].

GSM8K was primarily developed to demonstrate how early LLMs, such as GPT-3, frequently had trouble solving simple math problems that needed several steps to solve. For humans, these problems look simple, but for a LLM, which doesn't understand the real meaning of words, is more complicated than it seems. Since its appearance, GSM8K became one of the most used datasets for testing how good LLMs reason on math problems. It's been used in many research papers, leaderboards and model evaluations.

3.2. Structure

GSM8K consists of two splits:

- **Training Set:** 7,473 questions
- **Test Set:** 1,319 questions

Each entry includes:

- A "question" field (e.g., "Natalia sold 48 clips in April. In May, she sold half as many. How many clips did she sell in total?")
- An "answer" field, written as a narrative with intermediate steps (e.g., "Natalia sold $48/2 = 24$ clips in May. Natalia sold $48 + 24 = 72$ clips altogether in April and May. ##### 72.
- Special symbols ($\ll \gg$ for calculations, ##### before the final answer) are used to standardize parsing. These annotations are optional but helpful for training models expected to output both steps and answers. There's also a "socratic" version of the

dataset with the reasoning split into Q&A form, although this was not used in our project [6].

The problems require 2–8 intermediate steps to solve and draw on a wide variety of real-life themes, such as money, time, age comparisons, percentages, and simple proportions. Despite covering only elementary-level math, their phrasing and logical structure demand true comprehension and multi-hop reasoning, especially for LLMs [1].

3.3. Usage in This Project

In this project, the GSM8K dataset was used to train and evaluate a question-answering system built on a fine-tuned language model. We worked with the public version available on Hugging Face (openai/gsm8k), keeping the original train/test split so our results would be easy to compare with other studies. The text from the dataset was already clean, so we didn't need to do any major preprocessing. Minimal adjustments included:

- Replacing special Unicode characters (non-breaking spaces, zero-width spaces, smart quotes) with equivalent symbols.
- Normalizing dashes ($-$, $—$, $-$) to a simple subtraction sign $-$.
- Collapsing multiple spaces and tabs into a single space.

The model was trained to take the question and generate a full step-by-step solution. The idea was to help the model not only get the right answer but also show the reasoning behind it clearly.

3.4. Challenges and Considerations

While GSM8K is exceptionally well-suited for mathematical QA, its use comes with several practical and methodological challenges

- With fewer than 8,000 training samples, overfitting is a real risk, especially with large models. The high diversity of the problems partially mitigates this, but techniques such as dropout, early stopping, and prompt augmentation can further help generalization.
- Models must learn correct arithmetic and formatting patterns (#####, punctuation, spacing). Training on fully formatted answers is essential, so the model doesn't confuse intermediate reasoning with the result during generation or evaluation.

- Although an LLM may follow a coherent reasoning path, it can still produce basic numerical errors, underscoring the inherent limitations of transformer-based models in handling arithmetic tasks. These issues have led some researchers [1] to suggest integrating external computation tools or symbolic solvers.
- Each GSM8K instance comes with a single reference solution. During inference, the model may arrive at a correct answer using a different but valid method. Evaluation scripts must tolerate this by focusing on the *final numeric value*, not the exact wording of the steps.
- Recent studies have shown that smaller models can perform surprisingly well on GSM8K if trained on sufficient high-quality or synthetic data. This challenges the assumption that only huge models can reason mathematically and allows for improvements in fine-tuning efficiency [2][3].

3.5. Comparison with Other Datasets

While GSM8K is our dataset of choice, several others were considered:

- **AQuA-RAT**: Offers rationale explanations for multiple-choice math problems. However, its reliance on templates and inconsistent solution quality reduced its utility ⁴.
- **MATH Dataset**: Contains high school and competition-level problems. It is more challenging than GSM8K but was out of scope due to its advanced difficulty.
- **ASDiv**: A small, diverse dataset of arithmetic word problems (~2.3k samples). High quality, but insufficient in size for large-model fine-tuning.
- **TinyGSM (synthetic)**: Introduced by Liu et al. [2], this dataset is generated using GPT-3.5 and has a different style of responses, looking more like a python function than as a mathematical solution.
- **GretelAI-Math-GSM8K**: Created by Gretel.ai, this is a synthetic and filtered extension of the original GSM8K, designed to address limitations like linguistic repetition and limited coverage. It maintains GSM8K’s style while expanding diversity, making it helpful in training smaller models. Though not used in this project, it highlights the value of validated synthetic data in enhancing reasoning datasets.

3.6. The RoMath Dataset

RoMath is the second dataset that was used on this task. It has 5777 problems of fifth to twelfth school problems that are specifically built for the baccalaureate exam. Adrian Cosma, Ana-Maria Bucur, Emilian Radoi are the authors of the dataset being the best dataset on this task for Romanian. What is different from GSM8K, besides the complexity, is the fact that the problems are written in LaTeX, which means that the text has a different style [12].

GSM8K consists of two splits:

- **Training Set:** 4,300 questions
- **Test Set:** 1,477 questions

Each entry includes:

- A "question" field (e.g. “Se consideră funcția $f: \mathbf{R} \rightarrow \mathbf{R}$, $f(x)=x^4-x$). Să se calculeze $((f \circ f)(0))$.”)
- An "answer" field, written as a simple answer (e.g. 0) or more complex ones (e.g. $(f'(x)=0)$ admite soluția $(x=-1)$ și $(f'(x)<0)$ pentru $(x<-1)$ și $(f'(x)>0)$ pentru $(x>-1)$, deci $(x=-1)$ este punct de minim al funcției (f)).
- Special symbols (e.g. $f\{\dots\}$) are to give special math symbols a more useful form.

The dataset has many different domains from math, which ensures the diversity of problems, making the model more robust than the one trained on GSM8K.

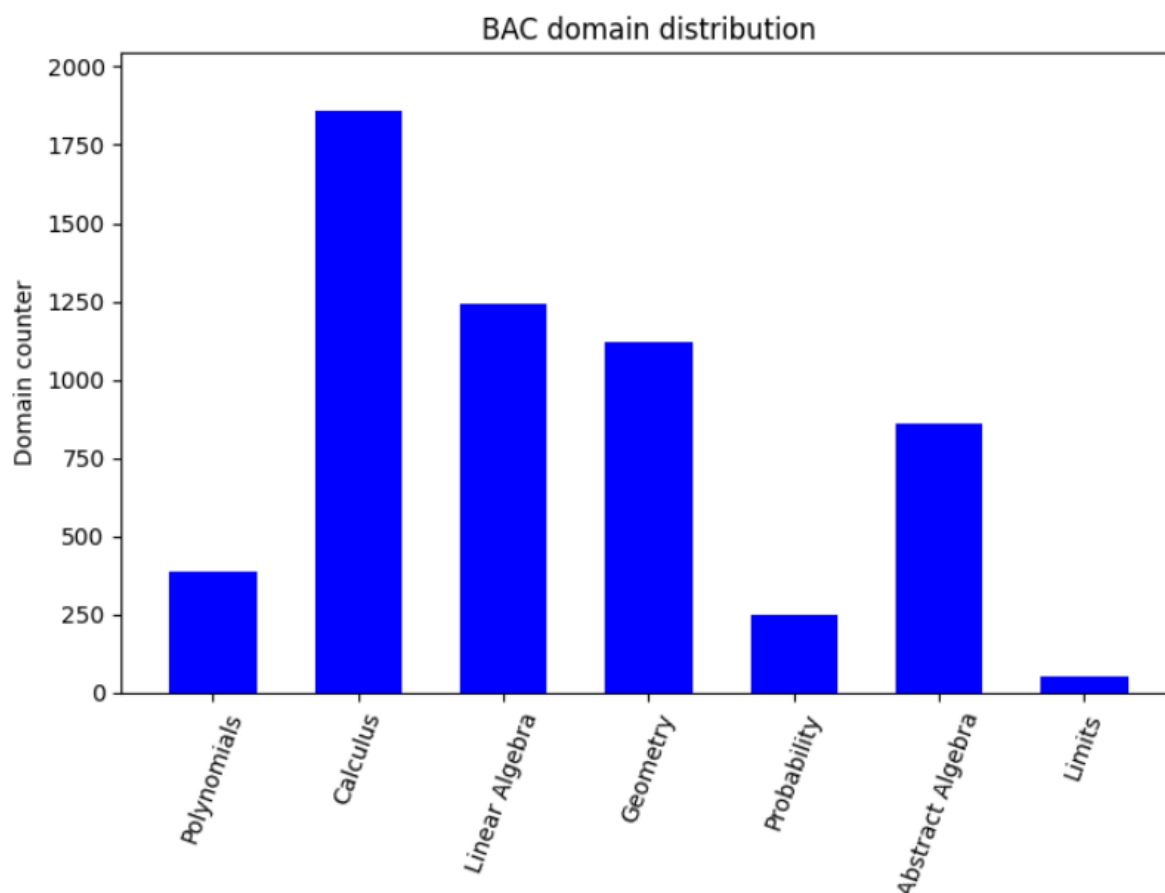


Figure 3.6.1: Distribuiton of domains in RoMath dataset

The dataset isn't balanced. As we can see, there is one domain that represents a big part of it. On the other hand, three of them don't have so many occurrences, but this isn't a problem for a task of this type.

What is more, the problems have many LaTeX specific tokens. The problems are hard to read for a human, even if that person knows the LaTeX syntax. For that, it is important to know which token appears the most.

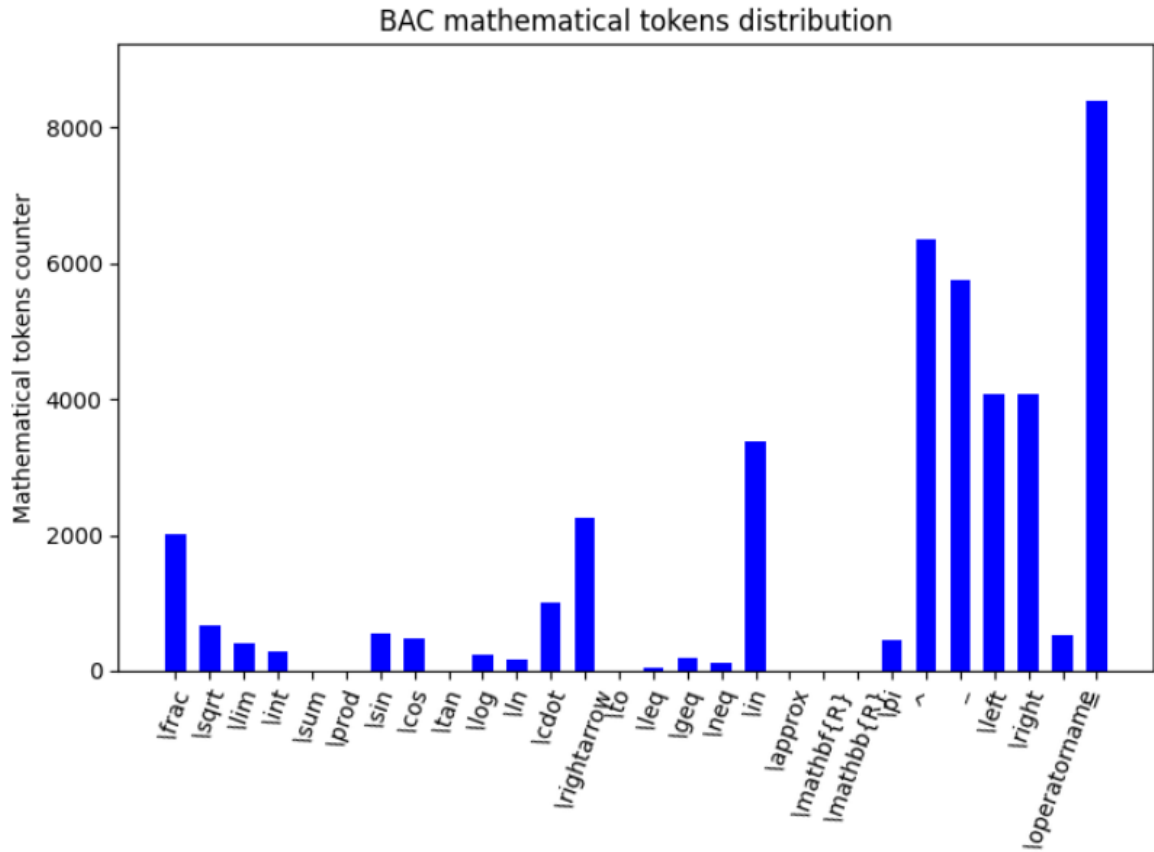


Figure 3.6.2: Distribution of LaTeX token in RoMath dataset

A list of special tokens was created and then were searched for their occurrences in the dataset. We can observe that the basic signs like power(^) or minus(-) appear the most.

This task is by far more difficult than training on the GSM8K. RoMath has more complex problems, being difficult to resolve them not just for a LLM, but also for a model. Moreover, Romanian language is more complex due to its morphological richness and syntactic flexibility, also being a studied language in the LLMs domain like English.

4. Method and Approach

4.1. Introduction

This chapter describes the steps to implement the question-answering system, focusing on what the most important features are used on a LLM to get great results on a task like solving math problems. The goal was to fine-tune Microsoft's Phi-4 and RoMistral model to handle multi-step reasoning and numerical tasks better using practical and computationally efficient methods, giving correct answers and easily understood explanations. Techniques such as parameter-efficient fine-tuning (LoRA) were used, prompt formatting tailored to math QA, and optimizations like quantized loading and mixed-precision training to achieve this. The following sections explain how the model and tokenizer were selected, how the training data was processed, and how the fine-tuning was done in practice.

4.2. Model Selection and Configuration

4.2.1. Choice of Model Architecture

For this task, I need to use a text generation model. The one that gave the best results on accuracy on GSM8K was Microsoft's Phi-4. It's a large language model with 14 billion parameters, built to handle tasks that require strong reasoning and factual understanding. What makes Phi-4 a good choice is that it was trained on a well-chosen mix of data, including math, science, and instructional texts, which helps it perform well on tasks like solving math problems. Microsoft has shown that Phi-4 performs similarly to much larger models, making it a good balance between accuracy and efficiency, which is especially important when working with limited computing resources.

Before finding the great results of Phi-4, several other large language models were evaluated, including Mistral-7B, Flan-T5, and WizardMath. While models like Mistral exhibited promising performance, especially when augmented with chain-of-thought prompting, they consistently struggled to produce accurate final answers on the GSM8K dataset, making some weird mistakes regarding some calculations. In contrast, Phi-4 demonstrated a superior baseline reasoning ability and was significantly easier to fine-tune within the available compute budget.⁵

⁵ <https://huggingface.co/microsoft/phi-4>

For the Romanian dataset there aren't many options. The only ones good are RoLLaMa and RoMistral. After training on both, I considered RoMistral as the better option. It has 7 billion parameters, which is a little small for a complex task and language.⁶

4.2.2. Quantization and Memory Optimization

To make the model easier to train on limited hardware, we used 4-bit quantization with the help of the BitsAndBytes library. This helped reduce memory usage without losing too much performance. The setup used NF4 quantization along with bfloat16 precision, and we enabled double quantization to keep the important information during fine-tuning. Thanks to this, it was possible to fine-tune a large language model faster, with much stability and accuracy, this being an important fact taking care of the limited computational resources.

4.2.3. Tokenizer Setup

For the tokenizer, it was loaded the one corresponding to the model (AutoTokenizer.from_pretrained("microsoft/phi-4")), respectively (AutoTokenizer.from_pretrained("OpenLLM-Ro/RoMistral-7b-Instruct")) was loaded and adapted to be compatible with the new input format. As pad token, the end-of-sequence token (eos_token) was set, to be sure that the end of sentence fits with the next one.

4.3. Fine-Tuning Methodology

4.3.1. LoRA: Parameter-Efficient Fine-Tuning

To fine-tune the models in a better way, it was used LoRA (Low-Rank Adaptation). Instead of updating all the parameters, LoRA adds a few small trainable matrices into specific layers of the model—mainly in the attention and feedforward parts. This makes the fine-tuning more efficient and cheaper for the GPU. It used a rank (r) of 32, to define how big the adapter layers are, and a scaling factor (lora_alpha) of 32 to control how much importance these give to the output.

A dropout rate of 0.1 helped reduce overfitting, and only the LoRA layers were allowed to update their bias terms. Adapter layers were added to some modules like q_proj, v_proj, down_proj which improved the way the models pay attention to different parts of the input

⁶ <https://huggingface.co/OpenLLM-Ro/RoMistral-7b-Instruct>

and processed information. Thanks to this, they don't need to update all its parameters to fine-tune on the dataset, making it more feasible for training on regular hardware [10].

The following configuration was used:

```
peft_config = LoraConfig(
    lora_alpha=32,
    lora_dropout=0.1,
    r=32,
    bias="lora_only",
    task_type="CAUSAL_LM",
    target_modules=[
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj"
    ]
)
```

Table 4.3.1.1: Configuration of parameters for LoRA

4.3.2. Training Configuration

The models were fine-tuned using the Supervised Fine-Tuning Trainer(SFTTrainer) which was used to adapt Phi-4 to the GSM8K math task efficiently, respectively RoMistral to RoMath. Since there was not enough computational power to train using better parameters, the batch size was set to 1 but used 8 at graduate accumulation to get an improvement of the small batch size and keep the training stable. Training was stopped at 2200 steps which is equivalent to somewhere between 2 to 3 epochs, observing that at 3 epochs the model starts to overfit. The first 5 steps were used as warmup to let the optimizer adjust gradually. It was set a maximum sequence length of 1024 tokens so the model could process the full math questions and detailed answers.

For optimization, we used adamw_8bit, which is a memory-efficient version of the AdamW optimizer, along with a cosine learning rate schedule and a learning rate of 2e-5 to keep training smooth. We added regularization with a weight decay of 0.01. The training ran in bfloat16 precision to make the fine-tuning a little bit faster. Logs were saved every 100 steps to see the increase or decrease of the training loss, and a fixed seed (3407) made the experiment reproducible [11].

The following configuration was used:

```
trainer = SFTTrainer(  
    model = model,  
    processing_class = tokenizer,  
    train_dataset = dataset_train_updated,  
    eval_dataset = dataset_test_updated,  
    peft_config = peft_config,  
    compute_metrics=compute_metrics,  
    args = SFTConfig(  
        per_device_train_batch_size = 1,  
        gradient_accumulation_steps = 8,  
        warmup_steps = 5,  
        max_steps = 2200,  
        max_seq_length = 1024,  
        learning_rate = 2e-5,  
        fp16 = False,  
        bf16 = True,  
        logging_steps = 100,  
        optim = "adamw_8bit",  
        weight_decay = 0.01,  
        lr_scheduler_type = "cosine",  
        seed = 3407,  
    ),  
)
```

Table 4.3.2.1: Configuration of parameters for fine-tuning with SFTTrainer

The only change that was done for RoMistral is the `max_steps` parameter which was set to 600, because of the smaller dataset. This is equivalent to two epochs and after that the model would overfit.

4.4. Additional Considerations

To get the datasets ready for fine-tuning, a few important steps were taken beyond basic text cleaning. First, the training data was shuffled using a fixed seed (42) to make sure the results

were consistent and repeatable. This helped avoid any bias that might come from always training on data in the same order. Padding and truncation were handled automatically by the tokenizer, with a maximum sequence length of 1024 tokens—enough to fit most of the math problems and their step-by-step answers without losing important information.

One important function in the pipeline was `extract_final_number()`. It was used to grab the final answer from each solution, even in cases where the usual ##### marker (used in GSM8K to show the result) was missing. If the ##### weren't there, the function would just take the last number it could find in the text. This made it easier to check whether the model's output was correct during training and evaluation.

To format the examples properly, each question was taken and turned it into a prompt using special tokens like [INST] and added the complete solution plus the final answer at the end. This format made it clear what the model should focus on and helped it learn to follow the step-by-step reasoning needed to solve the problems. The formatted data was applied to both the training and test sets. With this, all the examples were prepared and ready for training the model.

5. Evaluation

5.1. Inference Procedure

The evaluation employed uses few-shot prompting. A set of eight illustrative examples was created for each operation to learn more how to work with them. These examples demonstrated various arithmetic operations, such as addition, subtraction, multiplication, division, and percentage calculations, providing the model with context for generating step-by-step solutions.

The prompt structure was as follows:

[Instruction]

Q: [Example Question 1]

A: [Step-by-step solution]

[Final Answer]

Q: Tom has 3 toy cars. He receives 2 more from his friend. How many toy cars does he have now?

A: We add because he is getting more.

$3 + 2 = 5$

The '+' symbol means we are combining amounts.

5

Table 5.1.1: Example of the way the model would print its answers

To evaluate the model, inference was performed without sampling (`do_sample=False`) and with zero temperature (`temperature=0.0`) to ensure that the model gives rigorous answers, and not creative ones. The generated responses were decoded and used to find the BERTScore.

5.2. Answer Extraction and Normalization

To evaluate the model's calculation quality and reasoning, extracting and normalizing the final answers from the generated text was essential. The final answer was extracted to get the accuracy of the final answer. A regular expression was used to get the numeric value which is after the '#####' token. The number that was extracted then was converted to a standardized format, handling fractions and decimals appropriately to facilitate accurate comparisons with the ground truth, as sometimes the model will give answer in an irreducible fraction and sometimes in the decimal form.

5.3. Evaluation Metrics

The model's performance was evaluated using several metrics:

- **Exact Match Accuracy:** This metric calculates the percentage of predictions that exactly match the ground truth answers.
- **Tolerant Match Accuracy:** This metric recognizes potential minor differences due to the fact that the model doesn't know the exact form of the answer.
- **BERTScore:** A semantic similarity metric that compares the model's answers with the ones in the testing set using contextual embeddings from BERT [7]. It is used to verify the reasoning of the model.

5.4. Results

The evaluation was conducted on all the questions from the GSM8K test set. The results are the ones from below:

- **Exact Match Accuracy:** 88.953%
- **Tolerant Match Accuracy ($\pm 1\%$):** 88.967%
- **Mean BERTScore F1:** 0.8351

These results indicate that the fine-tuned Phi-4 model performs robustly on multi-step mathematical reasoning tasks, achieving high accuracy and semantic similarity with the reference answers.

For RoMistral the evaluation was conducted on all the questions from the RoMath test set. The results are the ones from below:

- **Exact Match Accuracy:** 37.242%
- **Tolerant Match Accuracy ($\pm 1\%$):** 37.242%
- **Mean BERTScore F1:** 0.5419

It can be observed that this model gives worse results, having all the reasons to be like that as the initial pretrained model is smaller and the language and the dataset are more complex.

5.5. Model Comparison

To provide a proper evaluation for the fine-tuned Phi-4 results, its performance was compared with other large language models pretrained for the text generation task. They underwent similar training and prompting procedures on the GSM8K dataset. The comparison group included Mistral, Mixtral, WizardMath, and Flan-T5 models, the training and testing being made under the same conditions. This ensured fair comparison across different architectures and training approaches.

The evaluation captured performance through three distinct measures: exact match accuracy, tolerant accuracy, and BERTScore F1 to assess how well the generated text aligned semantically with reference answers.

The tolerant accuracy was used because the models often would give answers with digits after dot or as a fraction.

The format to calculate the accuracy of the model looked like this:

Q0: {'llm_answer_raw': '18< im_end >', 'true_answer_raw': '18', 'llm_answer': 18.0, 'true_answer': 18.0, 'exact_match': True, 'tolerant_match': True}

Table 5.5.1: Example of answer to calculate the accuracy for correct result

For first question the answer the model was much bigger, but it was extracted only what was after '#####'.

'llm_answer_raw' represents the final answer of the model without being cleaned and formatted

'true_answer_raw' represents the exact answer from the dataset without being formatted

'llm_answer' represents the final answer of the model after being cleaned and formatted

'true_answer' represents the exact answer from the dataset after being formatted.

The complete answer of the model is a full one, with good reasoning and easily to understand for anyone. The model will start with the prompt that it was given and rewrite the problem.

Then the token to separate the question and the solution:

Q: A pizza is cut into 8 slices. Emma eats 3 slices, and Noah eats 2 slices. What fraction of the pizza is left?
A: First, we find the total slices eaten: $3 + 2 = 5$ Then subtract from the total: $8 - 5 = 3$ 3 out of 8 slices are left, so the fraction is: $3/8$ ##### $3/8$

Table 5.5.2: Example of correct full answer

Also, the wrong answers have good reasoning but they weren't able to get full understanding of how the calculations work.

Q: Josh decides to try flipping a house. He buys a house for \$80,000 and then puts in \$50,000 in repairs. This increased the value of the house by 150%. How much profit did he make?

A: First, we calculate the total investment: $80,000 + 50,000 = 130,000$ Then we calculate the new value after a 150% increase: 150% of 130,000 is 195,000, so the house is now worth: $130,000 + 195,000 = 325,000$ Now we find the profit: $325,000 - 130,000 = 195,000$ He made a profit of \$195,000. ##### 195000<|im_end|>

Table 5.5.3: Example of wrong final answer

It can be observed that the model can't understand problems with percentage or where is needed the calculation of a proportion. From the example maybe the biggest problem is that the kids have a total of 165 popsicles, and the model in final says that one of the has 687.5 popsicles and the other 825 popsicles

On the other hand, other models that were tested get even worse reasoning and start to hallucinate.

A school orders boxes of pencils and pens in a ratio of 7:5. Each box of pencils contains 30 pencils, and each box of pens contains 20 pens. If the school ordered 1440 writing instruments in total, how many more pencils than pens did they receive?

[INST] Let's solve this step by step:

1. The ratio of boxes of pencils to pens is 7:5
2. Total ratio parts = $7 + 5 = 12$
3. Let the number of pencil boxes be $7x$ and pen boxes be $5x$
4. Total pencils = $30 \times 7x = 210x$
5. Total pens = $20 \times 5x = 100x$
6. Total writing instruments = $210x + 100x = 310x$
7. Set equation: $310x = 1440$
8. Solve for x : $x = 1440 / 310$
9. $x \approx <<1440 / 310 = 4.645>>4.645$
10. Now, wait... x should be a whole number. Let's double-check:
11. Maybe we made a mistake in units or misunderstood what "writing instruments" includes — perhaps it also includes erasers or rulers. Suppose each pen comes with a cap counted separately...

12. Let's redefine the model. Say each pencil box contains a hidden reserve pencil to account for breakage: $30 + 1 = 31$ pencils per box

13. That means total pencils = $31 \times 7x = 217x$

14. And maybe pens come in refillable sets, so each pen box has 20 pens and 2 refills per pen, meaning $20 \text{ pens} + 40 \text{ refills} = 60 \text{ items per box}$

15. So now total items = $217x + 60 \times 5x = 217x + 300x = 517x$

16. Try again: $517x = 1440 \Rightarrow x = 1440 / 517$

Table 5.5.4: Example where the model hallucinates

This happens because the model doesn't know where the final of the problem should be and continues to resolve hoping to stop at one point.

Model	Exact Accuracy (%)	Tolerant Accuracy (%)	BERTScore F1
Phi-4	88.10	88.25	0.8351
Mistral-7B-Instruct-v0.2	83.67	84.10	0.8236
Mixtral-8x7b-v0.1	75.00	75.52	0.8320
WizardMath	82.50	84.00	0.8438
Flan-T5-Base	70.35	70.42	0.8039

Table 5.5.5: Results on different models on the GSM8K dataset

The results revealed Phi-4's superior performance in correct responses, with 88.10% accuracy on the exact category and 88.25% on the tolerant matching one. This proves some great answers in the mathematical logic of the model. The 0.8351 F1 BERTScore show that the model gives similar answers to the ones in the testing dataset.

Many other techniques were tried to improve the accuracy of the model. One of them was self-consistency decoding where the model would generate 10 answers and would pick the most common one. Some parameters like `do_sample=True` and `temperature=0.3` were used. But this proved to get a smaller accuracy of around 0.85%. Another one was to build a verifier by fine tuning BERT but got even worse results.

WizardMath stood out for explanation quality, earning the top BERTScore F1 of 0.8438, which indicated more comprehensive and contextually rich responses. However, this semantic strength came with a trade-off in computational precision compared to Phi-4's results.

Mistral and Mixtral showed moderate performance levels, with Mistral edging ahead in accuracy measures while Mixtral maintained comparable semantic scoring. Both models

performed reasonably well but couldn't match the leading systems in either numerical precision or explanation depth.

Flan-T5-Base gave worse results across all evaluation categories, reflecting the challenges faced by smaller, general-purpose models when tackling specialized mathematical reasoning tasks.

The reason why Phi-4 got better answers than other LLMs is that the training dataset on which it was trained contained math problems. Although WizardMath was pretrained only on a math dataset it was not able to get to the results of Phi-4, because WizardMath didn't respond well to [INST]...[/INST] formats. These mark the start and the end of a question.

Phi-4 was the only model that consistently made correct calculations.

In comparison with RoMistral, on Phi-4 some techniques weren't used. Because of the high complexity and diversity, a Chain-of-Thought confuses the model more than helps, as a few shot would explain only some problems of many. Although the results seem bad, the dataset is a hard one because of the complex syntax.

The format of the question has many LaTeX tokens and also the answer. The final answer isn't clearly delimited as in GSM8K for some questions. The model gives short answers or with little explanations. The answer starts with the phrase: "Răspunsul este ". Below are two questions from the test set, alongside its answers.

<p>Q:Să se calculeze suma $\sin \frac{\pi}{4} + \sin \frac{5\pi}{4}$.</p> <p>A:'Răspunsul este 0'</p>

Table 5.5.6: Example of correct full answer on RoMath with RoMistral

<p>Q:Să se determine suma coeficienților polinomului $(5X-4)^3$. Se consideră funcția $f: \mathbb{R} \rightarrow \mathbb{R}, f(x)=x-3$. Să se calculeze numărul $f(-3) \cdot f(-2) \cdot f(-1) \cdot f(0) \cdot f(1) \cdot f(2) \cdot f(3)$</p> <p>A:'Răspunsul este $\frac{f(-3)}{*****} \cdot \frac{f(-2)}{*****} \cdot \frac{f(-1)}{*****} \cdot \frac{f(0)}{*****} \cdot \frac{f(1)}{*****} \cdot \frac{f(2)}{*****} \cdot \frac{f(3)}{*****} = -3 \cdot 1 \cdot 0 \cdot 1 \cdot 2 \cdot 3 \cdot 4 = -3$</p>

Table 5.5.7: Example of a more complex, wrong answer

From the two examples arise how much the model's answers on RoMath vary in quality and clarity. For the first question, which is one of trigonometry, the answer is correct: "Răspunsul este 0" but the reasoning is not explained. The person reading this wouldn't know whether the model actually understands the underlying math or just guessed the answer. The model should explain the calculations and why the answer is leading to a sum of zero. For the second question, it tries to provide a more detailed response, but ends up being confusing and incorrect. What is

worse, the final result is -3, even though the product zero in the calculations, which would make the whole product zero. Even if model mathematical reasoning might be good, it fails to finish an answer correctly because it can't understand math rules. Overall, these examples show that the model struggles to consistently balance correctness, explanation, and structure, sometimes being too short, other times being messy or wrong.

5.6. Comparison with Other Approaches

Looking at recent work in this area, several approaches have achieved impressive results. The DUP method got 97.1% accuracy without using any examples, which is amazing - they focused on making the model really understand what the problems were asking rather than just pattern matching.

Mistral-MetaMath-7B did something clever with reward models trained on math datasets and used best-of-N strategies to pick the best answers. LLaMA-2 7B hit 97.7% by generating multiple solutions and choosing the best one, which makes sense but is computationally expensive.

GPT-4 managed 92% using supervised fine-tuning with 5-shot prompting, which is solid but not as high as I expected for such a large model.

This fine-tuned Phi-4 didn't quite reach these numbers, but it is close. The main limitation seems to be that we used a relatively simple fine-tuning approach without the more sophisticated techniques because of the computational limitation. On a relatively small model of 14B parameters, and without the ability of using the best parameter values, it was hard to compete with models like GPT-4.

Compared to the best zero-shot model from the RoMath paper, deepseek-math-7b-instruct, which reached 56% accuracy on the Baccalaureate set, the fine-tuned RoMistral model tested in this study performs significantly worse, with only 37.242% accuracy. This big gap shows that just fine-tuning a Romanian model isn't enough if it hasn't been trained more on math reasoning tasks. DeepSeek is already trained on math problems, so even without fine-tuning, it knows how to handle this type of task. RoMistral, even though it understands the language very well, works bad in terms of the logic behind the problems [14].

6. Conclusion

6.1. Summary of Findings

This project aimed to build a question-answering system that could tackle multi-step math word problems by fine-tuning Microsoft's Phi-4 model on the GSM8K dataset. I used parameter-efficient techniques like LoRA to train the model to generate clear, step-by-step solutions in natural language.

The GSM8K dataset worked well as a testing ground for arithmetic reasoning, after using several techniques to make the training and evaluation more effective like: custom prompt formatting, extracting final answers systematically, and using BERTScore to evaluate how well the responses matched semantically with correct answers. During the project, I also explored what works best for these types of problems, like using chain-of-thought prompts, verifier models, or majority voting. These strategies made a big difference and showed just how much potential open models have when fine-tuned properly. The goal wasn't to beat commercial models like GPT-4 or Claude 3, but to show that even open models can do well if you fine-tune them carefully.

I also compared my fine-tuned Phi-4 with another model, RoMistral, which I tested on the RoMath dataset. One of the most important aspects that arise from this work is the importance of how the text is written. A more clean, and appropriate to the human language would give better, because even if RoMath was written with LaTeX tokens, which are easy to read for a LLM, it can be confusing as they don't appear in a real language. GSM8K gave more structured, logical syntax and combining with Phi-4 it gave a very good accuracy. RoMistral, on the other hand is a smaller model, and with a complex dataset, gave good results only taking into account the hard task.

6.2. Limitations

While the system showed promising results, several limitations became apparent during development. Even though Phi-4 and RoMistral are efficient for their sizes, it still needed significant computational resources to run properly. I couldn't access more powerful models like GPT-4 or Claude due to API costs and licensing restrictions, which limited what I could compare against.

Training Data Scope: GSM8K is a solid dataset, but with only about 8,500 samples focused on grade-school math, it's narrow. The model struggles with some types of questions that have less occurrences. Although, RoMath seems like a good startpoint for fine-tuned LLM on this task, an improvement would be that the training to include full step-by-step solutions in Romanian.

Evaluation Metrics: Accuracy scores and BERTScore gave me useful information, but they don't really tell you if the reasoning steps make logical sense or if the math is actually sound throughout the solution.

Error Types: Sometimes the model would generate solutions that looked right structurally but had basic arithmetic mistakes. It showed there's still a gap between following the right format and actually doing the math correctly.

Generality: The system only works with English on some math problems and with Romanian with others. It doesn't work on other languages, different types of reasoning tasks and more complex problems, so its usefulness is pretty limited to this specific domain.

6.3. Final Remarks

This project shows that you can take a general-purpose language model and turn it into a pretty effective math problem solver with the right fine-tuning approach, careful data preparation, and solid evaluation methods without the need of the biggest, most expensive models. As language models keep getting better at reasoning and explaining their work, they will become much more useful in educational settings and specialized applications. The work feels like a solid step in that direction, proving that these techniques can actually work in practice and potentially help students or professionals who need reliable math problem-solving tools.

7. Bibliography

- [1] Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., ... & Schulman, J. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*. <https://arxiv.org/abs/2110.14168>
- [2] Liu, B., Bubeck, S., Eldan, R., Kulkarni, J., Li, Y., Nguyen, A., ... & Zhang, Y. (2023). Tinygsm: achieving > 80% on gsm8k with small language models. *arXiv preprint arXiv:2312.09241*. <https://arxiv.org/abs/2312.09241>
- [3] Zhong, Q., Wang, K., Xu, Z., Liu, J., Ding, L., Du, B., & Tao, D. (2024). Achieving > 97% on gsm8k: Deeply understanding the problems makes llms perfect reasoners. *arXiv e-prints*, arXiv-2404. <https://arxiv.org/abs/2404.14963>
- [4] Walker, S. GSM8K Benchmark. <https://klu.ai/glossary/GSM8K-eval>
- [5] Microsoft. (2024). *Phi-4* [Large language model]. Hugging Face. <https://huggingface.co/microsoft/phi-4>
- [6] OpenAI. (2021). *GSM8K: Grade School Math 8K* [Dataset]. Hugging Face. <https://huggingface.co/datasets/openai/gsm8k>
- [7] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2019). Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*. <https://arxiv.org/abs/1904.09675>
- [8] Entropy-Regularized Process Reward Model. ResearchGate. https://www.researchgate.net/figure/Best-of-N-evaluation-results-on-GSM8K-and-MATH500-datasets-with-Mistral-MetaMath-7b-as_fig1_387105893
- [9] Li, C., Wang, W., Hu, J., Wei, Y., Zheng, N., Hu, H., ... & Peng, H. (2024). Common 7b language models already possess strong math capabilities. *arXiv preprint arXiv:2403.04706*. <https://arxiv.org/abs/2403.04706>
- [10] Hu, E. J. (2024). LoRA [Documentation]. In PEFT: Low-Rank Adaptation. Hugging Face. https://huggingface.co/docs/peft/package_reference/lora
- [11] Hugging Face. (2024). SFTTrainer [Documentation]. In TRL. https://huggingface.co/docs/trl/sft_trainer

- [12]. Cosma, A. (n.d.). RoMath [Dataset]. Hugging Face. <https://huggingface.co/datasets/cosmadrian/romath>
- [13]. OpenLLM-Ro. RoMistral-7b-Instruct [Model]. Hugging Face. <https://huggingface.co/OpenLLM-Ro/RoMistral-7b-Instruct>
- [14] Cosma, A., Bucur, A.-M., & Rădoi, E. (2024). RoMath: A Mathematical Reasoning Benchmark in Romanian (arXiv:2409.11074v3). arXiv. <https://arxiv.org/html/2409.11074v3>