الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جامعة الشهيد حمه لخضر الوادي
University of Echahid Hamma Lakhdar El-Oued
كلية العلوم الدقيقة
Faculty of Exact Sciences
قسم الإعلام الآلي
Computer Science Department

Thesis

Submitted in partial fulfilment of the requirements for the Degree of

ACADEMIC MASTER

Field: Mathematics and Computer Science

Option: Computer Science

Specialty: Artificial Intelligence And Data Science

Theme

Enhancing Mathematical Reasoning in Large Language Models

Presented by:

- Daga Imane
- Lila fatma zohra

Discussed on Mai 26, 2025 by the jury:

• NAOUI Mohammed Anouar	PLB	Supervisor	Univ. El Oued
• Abbas Masoud	Professor	President	Univ. El Oued
• Ratima Farida	APA	Examiner	Univ. El Oued

Academic year: 2024/2025

KNOWLEDGMENT

"In the name of Allah, the Most Beneficent, the Most Merciful, acknowledging that our efforts are nothing without His will. All praise is due to Allah, who has granted us the strength and determination to carry out this work."

We extend our sincere gratitude to **Professor NAOUI Mohammed Anouar**, whose valuable guidance, insightful comments, and generous support have been essential to the completion of this work.

Finally, we are deeply grateful to our families, who supported us with patience, understanding, and unwavering encouragement throughout this journey. Their steadfast belief in us was a constant source of strength and motivation.

Thank you.

Dedication

To my dear parents, who never stopped believing in my abilities and have always been my pillar of strength. To my mother, who instilled in me a love for knowledge, and to my father, who taught me patience and perseverance—I dedicate this work as a token of my deep gratitude.

To my esteemed professors, who illuminated my path with knowledge and wisdom, and to my colleagues, who made this journey an unforgettable experience—thank you for your continuous support and encouragement.

To everyone who supported me through difficult times and stood by my side, believing that I could achieve this milestone—this work is for you. Without your support and faith, I would not have reached this stage.

And to coffee, my faithful companion during long nights, and to my friends who never stopped asking, *"When will you finally finish your thesis?"*—here it is at last!

Finally, to all those who strive for knowledge despite challenges, and to those who believe that education is the key to progress and advancement, I dedicate this work as a humble contribution to the pursuit of knowledge and research.

Abstract

The automation of mathematical problem solving is a growing field driven by the evolution of language models. This work explores the use of four pre-trained large language models—GPT-2, Qwen2.5-7B, DeepSeek-R1-Distill-Qwen-14B, and DeepSeek-R1-Distill-Qwen-1.5B—to generate accurate solutions for math problems. The models were fine-tuned using recent methods, including Low-Rank Adaptation (LoRA), to enhance performance while optimizing resource usage. Evaluation across different models allowed for performance comparison and highlighted the most effective approaches. The results demonstrate the potential of combining advanced language models with efficient fine-tuning to support Arabic-language education and automate problem-solving tasks.

Key words: Math Problem Solving,LLMs, LoRA, Qwen2.5-7B, DeepSeek, GPT-2

Résumé

L'automatisation de la résolution de problèmes mathématiques est un domaine en pleine expansion, stimulé par l'évolution des modèles de langage. Ce travail explore l'utilisation de quatre grands modèles de langage pré-entraînés — GPT-2, Qwen2.5-7B, DeepSeek-R1-Distill-Qwen-14B et DeepSeek-R1-Distill-Qwen-1.5B — pour générer des solutions précises à des problèmes mathématiques. Les modèles ont été ajustés à l'aide de méthodes récentes, notamment l'adaptation par matrices de faible rang (LoRA), afin d'améliorer les performances tout en optimisant l'utilisation des ressources. L'évaluation de ces différents modèles a permis de comparer leurs performances et de mettre en évidence les approches les plus efficaces. Les résultats démontrent le potentiel de la combinaison entre modèles de langage avancés et ajustements efficaces pour soutenir l'enseignement des mathématiques en langue arabe et automatiser les tâches de résolution de problèmes.

Mots-clés : Résolution de problèmes mathématiques, LLMs, LoRA, Qwen2.5-7B, DeepSeek, GPT-2

Contents

List of Figures

List of Tables

General Introduction

Mathematical reasoning remains a core challenge for large language models (LLMs) due to its reliance on symbolic precision, multi-step logic, and structured problem solving. In this study, we evaluate and enhance the capabilities of four LLMs—**GPT-2 (small)**, **Qwen2.5-7B**, **DeepSeek-R1-Distill-Qwen-14B**, and **DeepSeek-R1-Distill-Qwen-1.5B**—on tasks involving arithmetic, equation solving, and reasoning over multi-step problems.

We utilize a composite dataset consisting of real-world problems from a *Kaggle mathematical reasoning challenge* and synthetic data generated through custom Python scripts. This hybrid dataset is designed to test LLMs across both structured and naturalistic mathematical tasks.

To improve performance and efficiency, we adopt parameter-efficient fine-tuning techniques such as **LoRA**, **Adapters**, and **Prefix-Tuning**, alongside **QLoRA** quantization to enable training on limited hardware. In addition, we apply prompt engineering, rigorous data preprocessing, and **Reinforcement Learning with an automatic reward function** using the TRL library. This reward mechanism is designed to guide model outputs based on task correctness and logical consistency without requiring human-labeled feedback.

- **Chapter 1:** reviews LLM and Transformer fundamentals in the context of math problem solving.
- **Chapter 2:** explores advanced reasoning techniques, relevant datasets, and model limitations.
- **Chapter 3:** details our system design for prompt engineering, dataset generation, and fine-tuning strategies (QLoRA, GRPO).
- **Chapter 4:** describes the implementation environment—editor, libraries (Transformers, TRL, PyTorch), and compute resources—as well as the developed user interfaces.

CHAPTER I

LARGE LANGUAGE MODELS (LLMS)

I.1 Introduction

Large Language Models (LLMs) are a category of advanced AI models trained on vast amounts of data, enabling them to understand and generate natural language and other types of content. These models exhibit remarkable flexibility, allowing a single model to perform a wide range of tasks, including:

- **Answering questions** with intelligence and clarity.
- **Summarizing documents** to provide concise and relevant information.
- **Translating languages** with accuracy and fluency.
- **Completing sentences and texts** based on context.

With these capabilities, LLMs are revolutionizing content creation and enhancing user experiences in search engines and virtual assistants, making them one of the most impactful advancements in modern artificial intelligence.

I.2 Transformers

Traditional NLP models, such as RNNs and LSTMs, faced significant limitations, including difficulty capturing long-range dependencies, inefficient sequential processing, and poor handling of global context. These issues hindered their performance on tasks requiring a deep understanding of relationships between distant words or complex structures. The introduction of the Transformer architecture in *"Attention is All You Need"* by suggested by [67] marked a paradigm shift in NLP, offering a new approach to address these challenges and paving the way for advancements across various domains.A revolutionary technology introduced by Google in the **BERT** model and further refined in **GPT** (Generative Pre-trained Transformer), the Transformer model is based on the self-attention mechanism, as illustrated in Figure I.2.

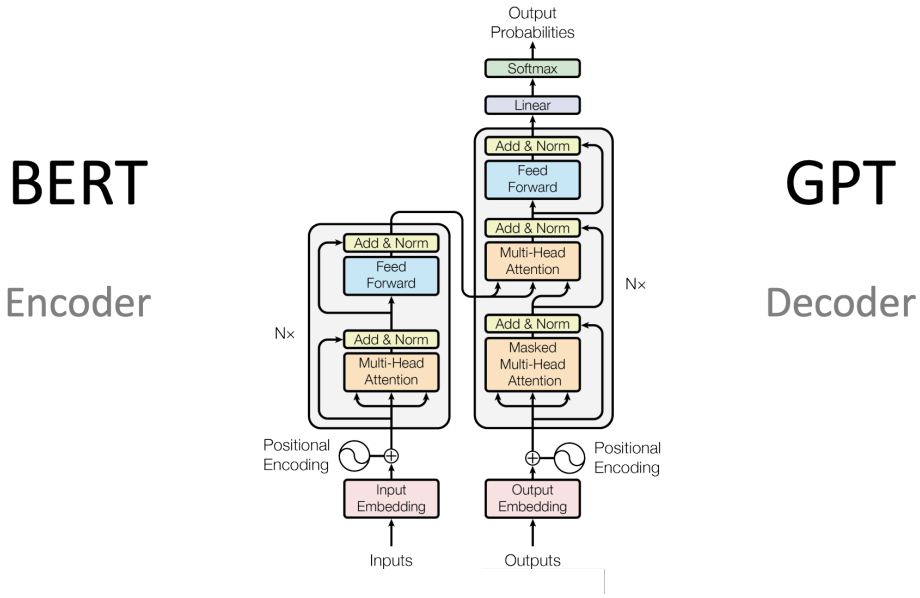


Figure I.1: The Transformer architecture, as introduced in [67].

I.3 Architecture of Large Language Models

I.3.1 Attention Mechanisms in Transformers

The Transformer architecture relies heavily on attention mechanisms, which allow the model to focus on relevant parts of the input sequence. The key components include **Self-Attention**, **Scaled Dot-Product Attention**, and **Multi-Head Attention**. These mechanisms enable the model to process and understand sequences effectively.

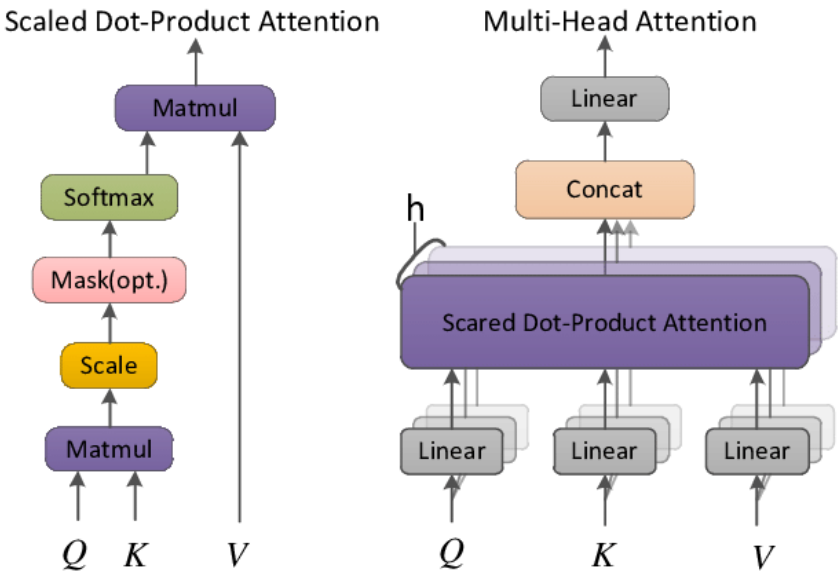


Figure I.2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention.

I.3.1.1 Self-Attention

Self-Attention is a fundamental component of the Transformer architecture, enabling the model to capture dependencies across different positions in a sequence. This mechanism allows the model to **weigh the relevance** of each token in relation to others, making it possible to attend to multiple aspects of the input simultaneously.

To achieve this, given an input sequence $X \in \mathbb{R}^{n \times d}$, where n is the sequence length and d is the embedding dimension, Self-Attention computes three essential matrices:

- **Query matrix (Q):** Defines what each token is looking for in other tokens.
- **Key matrix (K):** Represents the information contained in each token that may be relevant to others.
- **Value matrix (V):** Holds the actual token representations used to compute the final output.

These matrices are obtained by applying learned weight matrices W_Q, W_K, and W_V to the input sequence X as follows:

$$Q = XW_Q, \quad W_Q \in \mathbb{R}^{d \times d_k} \tag{I.1}$$

$$K = XW_K, \quad W_K \in \mathbb{R}^{d \times d_k} \tag{I.2}$$

$$V = XW_V, \quad W_V \in \mathbb{R}^{d \times d_v} \tag{I.3}$$

This formulation allows each token in the sequence to dynamically focus on other relevant tokens, **improving contextual understanding** [67].

I.3.1.2 Scaled Dot-Product Attention

Scaled Dot-Product Attention is a specific implementation of Self-Attention. It computes attention scores using the dot product of queries (Q), keys (K), and values (V). The attention weights are obtained by applying a softmax function to the scaled dot product of Q and K, ensuring numerical stability:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where:

- $Q \in \mathbb{R}^{n \times d_k}$ represents the query matrix.
- $K \in \mathbb{R}^{n \times d_k}$ represents the key matrix.
- $V \in \mathbb{R}^{n \times d_v}$ represents the value matrix.
- d_k is the scaling factor (dimensionality of the key vectors) used to prevent large values in the softmax computation.

This operation ensures that attention scores remain well-balanced, preventing excessively large values from dominating the softmax function. Figure I.2 on the left, the attention scores are computed as the scaled dot-product between the query and key matrices, followed by a softmax function to obtain the attention weights [67].

I.3.1.3 Multi-Head Attention

Multi-Head Attention extends Scaled Dot-Product Attention by computing multiple attention scores in parallel. Instead of using a single set of Q, K, and V matrices, Multi-Head Attention employs h independent attention heads, each with its own learned projections. Fig. I.2 on the right. Each of these projections is called an attention head.

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

where each attention head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Here: - W_i^Q, W_i^K, and W_i^V are learned projection matrices for the i-th attention head. - W^O is a learned output projection matrix. - Concat combines the outputs of all attention heads.

Using multiple attention heads allows the model to capture diverse features from different subspaces of the input representation. This mechanism enhances the model's ability to understand complex dependencies across words in a sequence, improving both contextual comprehension and representation power [67].

I.3.1.4 Positional Encoding

Transformers lack inherent sequential awareness, making positional encoding essential. To distinguish between nearby and distant tokens in a sequence, position embeddings P are intro-

duced. These embeddings capture positional information and are shared across different examples, similar to word embeddings E. They can either be learned during training or predefined using mathematical functions like sinusoids, as suggested by [67]. This approach helps the model generalize better to sequences longer than those encountered during training.

The final input representation is obtained by combining the word and position embeddings, formulated as:

$$x_e = E^T x + P^T x$$

This enriched representation is then passed through the network layers, enabling Transformers to process sequential information effectively.

I.3.1.5 Feed-Forward Network in Transformers

In addition to the attention mechanisms, Transformers incorporate a **fully connected feed-forward network (FFN)** in each layer. This FFN consists of two linear transformations separated by a **ReLU activation function**. Given an input representation x_{inp}, the output of the feed-forward network is computed as:

$$h_{\text{FNN}} = W_2 \text{ReLU}(W_1 x_{\text{inp}} + b_1) + b_2 \tag{I.4}$$

Here:

- W_1 expands the input into a higher-dimensional space, enhancing the model's expressiveness.
- W_2 projects it back to the required dimension.
- The **same feed-forward network** is applied independently at each position in the sequence.

I.3.1.6 Residual Connections and Layer Normalization

Each Transformer sub-layer utilizes **residual connections** [21] and **layer normalization** [4] to stabilize training and improve learning efficiency.

- **Residual connections:** Residual connections are a critical architectural feature in Transformer-based models, including large language models (LLMs). These connections enable sub-

layer inputs to bypass transformations and be added directly to outputs. This identity mapping facilitates unimpeded information flow across layers, preserving essential features and gradients during training while mitigating the vanishing gradient problem in deep networks.

By preserving and propagating earlier representations, residual connections enhance learning efficiency and stability. Each Transformer block implements this through skip connections that enable learning of residual functions rather than full transformations, significantly simplifying optimization and improving convergence in large-scale models.

As shown in Equation I.5, the multi-head attention layer computes its output through simple addition:

$$x_{\text{inp}} + \text{MHA}(x_{\text{inp}}) \tag{I.5}$$

- **Layer normalization:** As Large Language Models (LLMs) process data through multiple layers, maintaining training stability becomes essential. One core technique that supports this is Layer Normalization, introduced by [4] to stabilize and accelerate the training of deep neural networks. LayerNorm calculates statistics across the features of a single sample, making it particularly effective in scenarios involving sequential data or variable batch sizes, such as in Transformer architectures. It ensures stable learning by normalizing the inputs to each layer, preventing issues like exploding or vanishing gradients.

  Mathematically, given an input:

$$x = [x_1, x_2, ..., x_H]$$

  where H is the number of hidden units, LayerNorm computes the mean and variance:

$$\mu = \frac{1}{H}\sum_{i=1}^{H} x_i, \quad \sigma^2 = \frac{1}{H}\sum_{i=1}^{H}(x_i - \mu)^2$$

  It then normalizes the input:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

  and applies the learnable affine transformation:

$$y_i = \gamma \hat{x}_i + \beta$$

where ε is a small constant for stability, and γ, β are learnable parameters [4]. In Transformer architectures, LayerNorm is typically applied before or after sub-layers like self-attention or feed-forward blocks. It helps reduce internal covariate shifts, improving convergence and robustness.The source cited in[71] emphasize the importance of careful LayerNorm placement, showing that it significantly enhances performance in large-scale models.

After L Transformer layers, the final model output $f(x)$ is passed through a **softmax layer**, which generates the final predictions.

I.4 Transformer Layers

I.4.1 Input Layer

- **Data Processing:** The input layer handles both textual and mathematical data. This includes LaTeX-based expressions commonly used in scientific and technical contexts to represent complex mathematical formulas, as well as synthetic math problems and answers generated through Python code. These Python-generated problems include basic arithmetic, multi-step problems, equations, word problems, and quadratic problems. The dataset combines these two types of content to ensure a diverse range of problem-solving tasks for training.

- **Tokenization Techniques:** It employs methods like Byte Pair Encoding (BPE) [52] to break down both the textual and mathematical content, including LaTeX expressions and Python-generated math problems, into smaller, manageable units (tokens). This process allows the model to efficiently process and understand both the textual descriptions and mathematical notations by converting them into discrete units

- **Embedding Layers:** Utilizes embedding layers [38, 47] to convert the tokenized text and mathematical symbols into numerical representations for efficient processing. These embeddings allow the model to learn and process both textual and mathematical content by representing them as continuous vectors representing them as continuous vectors.

I.4.2 Hidden Layers

The hidden layers of a large language model (LLM) are built from multiple **Transformer blocks**, each containing several key components that enable the model to capture complex relationships between words and symbols. These components include the **self-attention mechanism** I.3.1.1, which allows the model to focus on different words in a sequence based on their relevance to each other, enabling it to understand context and long-range dependencies. To preserve the sequence information,**positional encoding** I.3.1.4 is added to the input embeddings, ensuring that the model can differentiate between words in different positions.

Each Transformer block also incorporates a **feed-forward network** I.3.1.5, which further processes the information by applying transformations to the attention outputs. **Residual connections** I.3.1.6 are used to improve training by ensuring that the gradient flows more easily through the network, preventing vanishing gradients. **Layer normalization** I.3.1.6 is applied to stabilize the learning process by normalizing the input to each layer, allowing the model to converge more effectively.

Together, these components form the core of the hidden layers, enabling the model to learn rich, context-aware representations of both textual and mathematical data.

I.4.3 Output Layer

The output layer is responsible for generating the final predictions of the model. It does so by taking the processed representations from the last Transformer block and passing them through a linear layer followed by a softmax function. This layer predicts the next most probable token based on the model's contextual understanding of the input sequence. It employs predictive modeling techniques to select the most relevant response, whether it's the next word in a sentence or the correct answer to a mathematical problem. The quality of the output depends heavily on the representations learned in the hidden layers I.4.2.

I.5 LLMs for Solving Mathematical Problems Process

I.5.1 Parsing and Understanding the Problem

When a user inputs a mathematical problem—whether in plain text or LaTeX format—large language models (LLMs) begin by analyzing the mathematical context.

The problem is converted into an internal representation that enables the model to interpret the relationships between symbols, numbers, and constants. The mathematical expression is decomposed into its fundamental components: coefficients, operations, and variables.

Some LLMs are specifically trained to recognize common patterns in mathematical queries, such as algebraic equations or calculus problems. This pattern recognition helps the model infer the objective of the question (e.g., simplification, solving, or differentiation).

I.5.2 Reasoning and Problem-Solving

After understanding the problem, the model initiates a logical reasoning process to arrive at a solution. This involves applying mathematical knowledge acquired during training, such as algebraic rules or calculus techniques.

Some LLMs are fine-tuned on extensive mathematical datasets containing millions of problems and their corresponding solutions. This fine-tuning enhances the model's ability to solve increasingly complex problems.

To further improve accuracy and reliability, some models are integrated with specialized mathematical tools, including:

- **SymPy:** A robust Python library for symbolic mathematics and equation solving.
- **Wolfram Alpha API:** A powerful computational engine for solving advanced mathematical queries and verifying results.

This integration allows LLMs to generate highly accurate answers, and in some cases, provide detailed step-by-step solutions, closely mimicking the reasoning process of a human solver.

I.5.3 Verification and Validation

To ensure the reliability of mathematical solutions, advanced LLMs often employ *Chain of Thought (CoT) reasoning*, which allows them to break down problems into logical, interpretable

steps rather than simply providing the final answer. This step-by-step approach not only enhances transparency but also increases the chances of identifying and correcting errors during the reasoning process.

Moreover, LLMs can perform *cross-verification* by solving the same problem using different strategies—such as symbolic manipulation, numerical approximation, or leveraging external APIs like Wolfram Alpha. By comparing the outcomes of these diverse methods, the model can validate the consistency and correctness of its final result.

This dual process of stepwise reasoning and multi-method validation contributes significantly to the **robustness and accuracy** of the solutions generated, making LLMs more dependable for tackling a wide range of mathematical tasks.

I.6 Applications of LLMs in Mathematics and Technical Fields

I.6.1 AI-Powered Math Assistants

LLMs can be used to develop **AI chatbots** that answer mathematical questions on platforms like **ChatGPT** and **Wolfram Alpha**.

For example, the **DeepSeek-Math** Figure I.2 model has been used to build AI systems capable of handling complex mathematical problems written in LaTeX format, achieving high accuracy in solving multi-step equations. This model has proven its effectiveness on benchmarks such as GSM8K and MATH, making it a strong candidate for developing intelligent educational assistants with advanced mathematical reasoning capabilities. [35]

Value
7 Billion Parameters
500 Billion Tokens
56% DeepSeekMath Corpus, 4% AlgebraicStack, 10% arXiv, 20% GitHub Code, 10% Common Crawl
10 Million Tokens
4,000 Tokens
AdamW
4.2×10^{-4}
Warmup for 2K steps, then decay to 31.6% after 80% training, and to 10% after 90%

Table I.1: Hyperparameters of DeepSeekMath-Base 7B Model

	DeepSeek-VL2-Tiny	DeepSeek-VL2-Small	DeepSeek-VL2
Vocabulary size	129,280	102,400	129,280
Embedding size	1,280	2,048	2,560
#Attention heads	10	16	32
#Layers	12	7	30
Multi-Head Attention	MLA (rank=512)	MLA (rank=512)	
#Round experts	64	64	72
#Shared experts	2	2	2
Top-K for expert selection	6	6	6
Routing function	Softmax	Softmax	Sigmoid
Expert correction bias	×	×	✓

Table I.2: architectural configuration for deepseek

I.6.2 Enhancing Scientific Research

LLMs assist researchers in analyzing academic papers, simplifying complex theories, and solving equations.

For example, the **GPT-2** Figure I.3 model has been employed in research aimed at summarizing scientific articles in Arabic and analyzing the stylistic structure of texts. One such experiment involved fine-tuning the model using the publicly available datasets Khaleej-2004 and Watan-2004. This approach enabled researchers to generate academic-style texts and predict paragraph completions based on scientific context. [49]

Parameters	**Layers**	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

Table I.3: Architecture hyperparameters for the 4 model sizes

I.6.3 Supporting Learning and Education

LLMs enable students to engage in step-by-step problem-solving, making self-learning more accessible and effective.

For instance, the **Qwen2.5-7B** model has been integrated into educational platforms to sup-

port interactive learning Figure I.4. Its capabilities in understanding mathematical notation and logical sequences allow it to guide students through solving equations or understanding concepts progressively, enhancing personalized learning experiences.

Models	# Params (B)	# Non-Emb Params (B)	# Layers	# Head (D/K)	T=Embedding	Context Length	Generation Length	License
Qwen2-0.5B-A2	0.49	0.36	24	14/2	Yes	32K	8K	Apache 2.0
Qwen2-1.5B-A2	1.5	1.3	28	12/2	Yes	32K	8K	Apache 2.0
Qwen2-5.5B	3.1	2.8	36	16/2	Yes	32K	8K	Qwen Research
Qwen2-7B	7.6	6.5	28	28/4	No	128K	8K	Apache 2.0
Qwen2-14B	14.7	13.1	48	40/8	No	128K	8K	Apache 2.0
Qwen2-72B	32.5	13.1	64	40/8	No	128K	8K	Apache 2.0
Qwen2-0.5B-Chat	72.7	70.0	80	64/8	No	128K	8K	Qwen
Qwen2-1.5B-Chat	1.5	1.3	28	12/2	Yes	128K	2K	Apache 2.0
Qwen2-5.5B-Chat	7.6	6.5	28	28/4	No	128K	2K	Apache 2.0
Qwen2-7B-Chat	1.5	1.3	28	12/2	Yes	4K	2K	Apache 2.0
Qwen2-14B-Chat	7.6	6.3	28	28/4	No	4K	2K	Apache 2.0
Qwen2-72B-Chat	72.7	70.0	80	64/8	No	4K	2K	Qwen 2.0

Table I.4: Illustration of the Qwen2.5-7B model in an educational setting.

I.7 Importance of LLMs in Our Work

Since our work focuses on **developing an AI chatbot for solving mathematical problems**, LLMs are essential for achieving this goal. Their significance in our work includes:

1. **Understanding and interpreting LaTeX-formatted mathematical problems.**
2. **Analyzing complex equations and providing accurate solutions.**
3. **Integrating with external math-solving tools like Wolfram Alpha for enhanced precision.**
4. **Fine-tuning with specialized mathematical datasets to improve response quality.**
5. **Delivering a seamless and efficient user experience through an intelligent AI chatbot.**

I.8 Fine-Tuning Large Language Models (LLMs)

Fine-tuning refers to the process of adapting a pre-trained LLM to a specific task or dataset, such as sentiment analysis, question answering, or text classification. This process involves

adjusting the model's parameters to enhance performance on the target task while utilizing the knowledge gained during pretraining.

There are two primary methods for fine-tuning LLMs: **full fine-tuning** and **parameter-efficient fine-tuning (PEFT)** [23]. Full fine-tuning modifies all model weights, including attention mechanisms and output layers, allowing the model to specialize in a given domain. However, this method demands substantial computational resources and high-quality datasets. In contrast, PEFT optimizes only a portion of the model's parameters, significantly reducing computational costs and training time while maintaining performance efficiency.

I.8.1 Parameter-Efficient Fine-Tuning (PEFT)

The Parameter-Efficient Fine-Tuning (PEFT) [23] techniques have emerged as an effective method for optimizing the fine-tuning process of large language models (LLMs). These approaches focus on updating a selective subset of parameters, thereby decreasing the computational cost and resource requirements. PEFT strategies aim to learn only a small set of parameters specific to the task at hand. This can be achieved by designing additional layers [24], adding virtual tokens [34] (a form of prompt tuning) to the input, or decomposing weight gradients into specialized matrices [24].

One notable PEFT technique is Low-Rank Adaptation (LoRA) [24], which involves freezing the model weights and introducing low-rank trainable matrices into the attention layers of the Transformer architecture. This reduces the number of parameters that need to be trained.

Additionally, QLoRA [15] combines LoRA [24] with model quantization, reducing GPU memory usage by lowering the precision of floating-point data types in the model. Overall, PEFT methods are designed to improve model performance by updating only a small number of parameters, making them particularly suitable for tasks that require efficient modification of model behavior, such as knowledge editing.

I.9 Advantages of General LLMs and Math-Specialized LLMs

I.9.1 Advantages of General LLMs

- **Versatility:** These models can perform a wide range of tasks such as text generation, translation, summarization, sentiment analysis, and question answering.

- **Domain Adaptability:** They can be fine-tuned to suit specific fields (e.g., legal, medical, technical), enhancing their precision and relevance.
- **Multilingual Support:** Many general LLMs support multiple languages, making them effective for international communication and multilingual applications.
- **Contextual Understanding:** Trained on vast datasets, they have strong abilities to understand complex contexts and produce accurate responses.
- **Productivity Boost:** Used in tools like virtual assistants, content generation platforms, and code tools, they significantly improve productivity.
- **Scalability:** Once trained, these models can be deployed at scale with minimal adjustments, making them cost-effective and efficient for businesses. [30]

I.9.2 Advantages of Math-Specialized LLMs

- **Mathematical Reasoning:** Specifically trained to solve logical mathematical problems with step-by-step explanations, which general LLMs often lack.
- **Formal Verification:** These models incorporate symbolic computation and formal theorem proving for mathematically verifiable outputs.
- **Improved Precision:** Tailored for numbers and logic, they offer higher accuracy in mathematical domains compared to general models.
- **Educational Assistance:** Helpful for learners by providing structured solutions, encouraging logical thinking and problem-solving skills.
- **Scientific Research Support:** Used in computational mathematics and scientific discovery, helping explore theories and generate proofs.
- **Structured Outputs:** Unlike general models, math-specialized LLMs tend to produce outputs that follow formal mathematical structures. [2]

I.10 Difference Between General LLMs and Math-Specialized LLMs

Aspect	General LLMs	Math-Specialized LLMs
Training Scope	General text data (conversations, articles, websites, books)	Structured math data, academic papers, LaTeX-formatted equations
Primary Objective	Natural language processing, translation, summarization, general interaction	Solving complex math problems, proofs, logical reasoning
Math Accuracy	Low to moderate, especially on multi-step reasoning problems	Very high, trained to understand symbolic structures and mathematical patterns
Model Examples	ChatGPT, GPT-4, Claude, LLaMA	Minerva (DeepMind), MathGPT, MATH-AI
Strengths	Versatile, flexible in language-based tasks, capable of general topic handling	Strong in mathematics, symbolic reasoning, LaTeX interpretation
Weaknesses	Poor performance on symbolic math and reasoning-heavy problems	Limited general text abilities, less effective in natural conversation
Best Use Cases	General-purpose chat, translation, content generation, search	Education, auto-grading, solving math problems, academic support

Table I.5: Comparison Between General LLMs and Math-Specialized LLMs

I.11 Conclusion

Large Language Models (LLMs) represent a major advancement in AI, enabling sophisticated language and mathematical processing. By leveraging LLMs effectively, we can develop a powerful AI chatbot capable of understanding complex mathematical problems and providing precise solutions, ultimately enhancing user experience and contributing to scientific progress.

CHAPTER II

ADVANCED TECHNIQUES IN MATHEMATICAL REASONING USING LARGE LANGUAGE MODELS

II.1 Introduction

This chapter investigates the capabilities of large language models (LLMs) in addressing mathematical reasoning tasks. It outlines the scope of mathematical reasoning, discusses key challenges, and categorizes various problem types such as arithmetic, word problems, and multi-step equations. The chapter also introduces datasets like IconQA to highlight the complexity of abstract mathematical tasks. By analyzing how LLMs perform in these scenarios, we aim to understand both their strengths and the obstacles they face in aligning with human cognitive processes.

II.2 Mathematical Reasoning: Definitions and challenges

II.2.1 What is mathematical reasoning ?

Mathematical reasoning refers to the process of drawing logical conclusions based on established principles and verifiable facts. It is the foundation of problem-solving and proof construction in mathematics. At its core, mathematical reasoning involves analyzing statements to determine their validity and using structured logic to arrive at conclusions [8].

II.2.2 Types of Mathematical Reasoning

Mathematical reasoning can be classified into several types, each playing a unique role in how problems are approached and solved:

- **Deductive Reasoning:** This form of reasoning derives specific conclusions from general rules or axioms. It is rigorous and logical. For example, if the Pythagorean Theorem holds for all right-angled triangles, then it must hold for any specific triangle proven to be right-angled.

- **Inductive Reasoning:** In contrast, inductive reasoning involves making generalizations based on patterns or specific examples. Although useful in forming conjectures, it is not logically rigorous, and thus less reliable for formal proofs. For example, if multiple items with a cost price of Rs. 15 sell for Rs. 50, one might conclude that selling such items is profitable.

- **Numerical Reasoning:** Focuses on solving problems involving numbers and quantitative relationships, such as interpreting data, performing calculations, or identifying number patterns.
- **Algebraic Reasoning:** Involves manipulating symbols, expressions, and variables to analyze relationships and solve equations.
- **Logical Reasoning:** Encompasses both deductive and inductive thinking, as well as the ability to evaluate statements for consistency, validity, and truthfulness.

Although other types of reasoning such as intuition, counterfactual thinking, abductive induction, critical thinking, and backward induction are used in broader decision-making, mathematics primarily relies on deductive and inductive reasoning. Notably, deductive reasoning is considered more fundamental in mathematics due to its logical rigor and reliance on established truths [8].

II.3 Math Problems and Datasets

This section highlights various types of mathematical problems and datasets aimed at evaluating problem-solving skills, including arithmetic calculations and word problem-solving. The goal is to measure the depth and range of mathematical abilities, from basic arithmetic to complex real-life scenarios that require reasoning and interpretation.[2]

II.3.1 Arithmetic

Definition: Arithmetic problems involve fundamental mathematical operations like addition, subtraction, multiplication, and division, with no need for interpretation of extra context.

Key Features:

- Simple and direct.
- Focus on numerical manipulation.
- No advanced reasoning required.

Example:

```
quadratic:  21 + 97
Solution:  118
```

II.3.2 Math Word Problems (MWP)

Definition: Math Word Problems are exercises that present mathematical challenges through written or verbal descriptions. The solver must interpret the problem, identify relevant concepts, and formulate equations or expressions to find the solution.

Variations:

1. **Question-Answer (Q-A):**
   - Direct question and final answer.
   - **Example :**

```
problem:  Lily received 20fromhermum.Afterspending10 on a book and
2.5onalollipop,howmuchdoesshehaveleft?
Solution:  7.5
```

2. **Question-Equation-Answer (Q-E-A):**
   - Includes the question, equation used, and final answer.
   - **Example:**

```
problem:  Jack had 8 pens, Mary had 5.  Jack gave 3 pens to Mary.
How many pens does Jack have now?
Equation:  8 - 3 = 5
Solution:  5
```

3. **Question-Rationale-Answer (Q-R-A):**

    - Provides a step-by-step explanation of how the answer is reached.
    - **Example:**

```
problem: Beth bakes 8 dozen cookies. If 16 people share them
equally, how many cookies does each person get?
Rationale: 8 dozen = 96 cookies. 96 ÷ 16 = 6
Solution: 6 cookies
```

Key Features:

- Designed to interpret real-world scenarios and convert them into mathematical expressions.
- Tests both computational skills and reasoning abilities.
- Essential for evaluating models that require advanced mathematical understanding.

II.3.3 Multi-step Problems

Definition: These are problems that require performing more than one mathematical operation to reach the solution. They are often presented as word problems that demand understanding the text, identifying given data, and applying a sequence of operations.[16]

Features:

- Require deep understanding and analysis of the problem.
- Involve planning and organizing computational steps.
- Enhance critical thinking and problem-solving skills.

Types:

- Multi-step addition and subtraction problems.
- Multi-step multiplication and division problems.
- Problems that combine different types of operations.

Example:

```
problem:
Ahmed has 3 boxes, and each box contains 4 books.  If he gives 5
books to a friend, how many books does he have left?
Solution:
Total books:  3 × 4 = 12 books
Remaining books:  12  5 = 7 books
```

II.3.4 Equation Problems

Definition: Equation problems involve solving mathematical expressions with unknown variables. These problems require manipulating algebraic equations to find the value of the variable(s) that satisfy the equation.[36]

Features:

- Reinforce understanding of algebraic principles.
- Require symbolic manipulation and logical reasoning.
- Promote abstract thinking and mathematical modeling.

Types:

- Linear equations with one variable.
- Equations with variables on both sides.
- Systems of linear equations.

Example:

```
problem:
Solve for x:  3x + 5 = 20
Solution:
Subtract 5 from both sides:  3x = 15
Divide both sides by 3:  x = 5
```

II.3.5 Quadratic Problems

Definition: Quadratic problems involve equations of the second degree, typically expressed in the form $ax^2 + bx + c = 0$. These problems are fundamental in algebra and appear in a wide range of mathematical contexts. [66]

Features:

- Involve recognizing and solving non-linear equations.
- Require the use of specific methods like factoring, completing the square, or the quadratic formula.
- Develop deeper understanding of polynomial relationships and their graphs.

Types:

- Quadratic equations with real and distinct roots.
- Quadratic equations with repeated (double) roots.
- Quadratic equations with complex roots.

Example:

```
problem:
Solve the equation:  x2  6x + 9 = 0
Solution:
Recognize it as a perfect square:  (x  3)2 = 0
Solve for x:  x = 3
```

II.4 Math Word Problems in Abstract Scenes

II.4.1 Data Collection

- **Objective:**
  Construct a dataset of icon-based question–answer pairs that require multiple reasoning skills, including visual understanding and general commonsense reasoning.[79]

- **Source of Data:**

  Problems were extracted from open-source math textbooks, primarily via the *IXL Math Learning* platform, which aligns with California's Common Core standards.

- **Target Age Group:**

  The problems cover students from pre-kindergarten through third grade, featuring abstract images and requiring little to no specialized domain knowledge.

- **Data Processing:**

  - Filtered out problems that lacked icon images or contained only black-and-white illustrations.
  - Removed duplicate or inappropriate items.
  - Randomly shuffled answer choices to prevent bias.

- **Skill Categorization:**

  Each question was tagged with 1–3 of 13 skill categories based on metadata keywords, including:

  - **Geometry:** shapes, angles, area, etc.
  - **Counting:** tallying, place value, numerical comparison, etc.
  - Other categories: time, fractions, probability, spatial reasoning, patterns, and more.II.4.1

II.4.2 Data Analysis

In this section, 107,439 data points are collected from the IconQA dataset, where each data point contains a colored icon image, a natural language question, optional image or text choices, and the correct answer. The dataset is split into three parts: training (6), validation (2), and testing (2) with a 6:2:2 ratio as shown in image II.2.

II.4.2.1 Sub-tasks in IconQA

- **Multi-image-choice:** This requires selecting the correct image from a list of images based on the corresponding diagram and its related question.

Skill types	Trigger words in metadata
Geometry	name the shape, shapes of, classify shapes, solid, corners, faces, edges, vertices, sides, dimensional, rectangle, circle, triangle, square, rhombus, sphere, cylinder, cone, cubes, hexagon, perimeter, area, curved, open and close, flip turn, symmetry
Counting	count, tally, a group, ordinal number, area, even or odd, place value, represent numbers, comparing review, equal sides, square corners, one more, one less, fewer, enough, more.
Comparing	compare, comparing, more, less, fewer, enough, wide and narrow, light and heavy, long and short, tall and short, match analog and digital
Spatial	top, above, below, beside, next to, inside and outside, left
Scene	problems with pictures, beside, above, inside and outside, wide and narrow, objects
Pattern	the next, comes next, ordinal number, different
Time	clock, am or pm, elapsed time, times
Fraction	equal parts, halves, thirds, fourths, fraction
Estimation	estimate, measure
Algebra	count to fill, skip count, tally, even or odd, tens and ones, thousands, of ten, elapsed time, perimeter, area, divide
Measurement	measure
Commonsense	light and heavy, compare size, holds more or less, am or pm, times of, too!
Probability	likely

Table II.1: Trigger words found in metadata used to categorize skill types in the IconQA dataset.
words in IconQA.

Tasks	All	Train	Val	Test
Multi-image-choice	57,672	34,603	11,535	11,535
Multi-text-choice	31,578	18,946	6,316	6,316
Filling-in-the-blank	18,189	10,913	3,638	3,638
All	107,439	64,462	21,489	21,489

Table II.2: Statistics for the IconQA dataset.

- **Multi-text-choice:** This involves selecting the correct answer from 2-5 text choices with an abstract diagram.
- **Filling-in-the-blank:** Similar to the typical VQA task, but the images are icon images instead of natural images.

II.4.2.2 Question Statistics

Figure II.1 shows the distribution of question lengths for each sub-task. Questions longer than 35 words are counted as having 35 words. The distribution of questions is more even in the ”multi-text-choice” task, while the ”multi-image-choice” task shows a long-tail distribution due

to the complexity of textual scenarios.

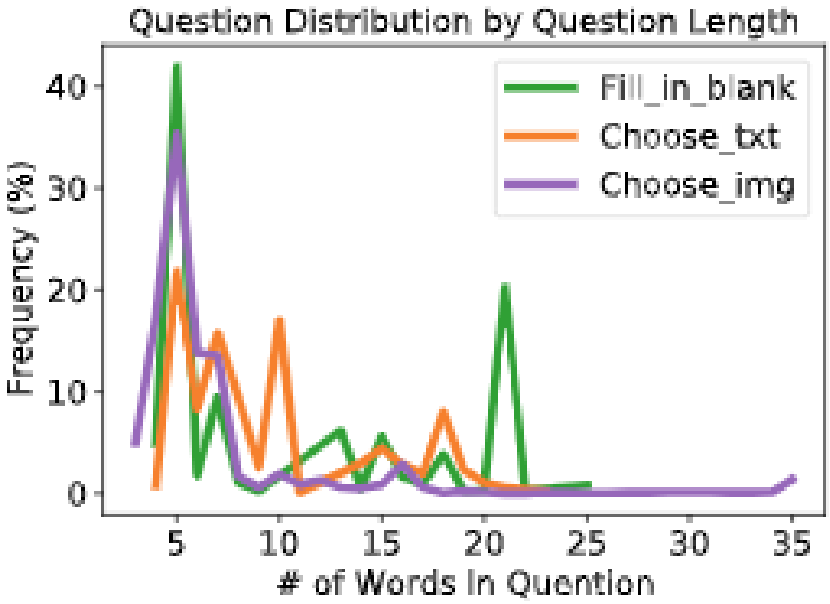


Figure II.1: Question statistics based on number of words in IconQA.

II.4.2.3 Most Frequently Mentioned Icons

Figure II.2 shows the frequency of the 40 most mentioned icons in the IconQA question texts. These icons cover a variety of real-world objects such as animals, plants, shapes, food, and more.

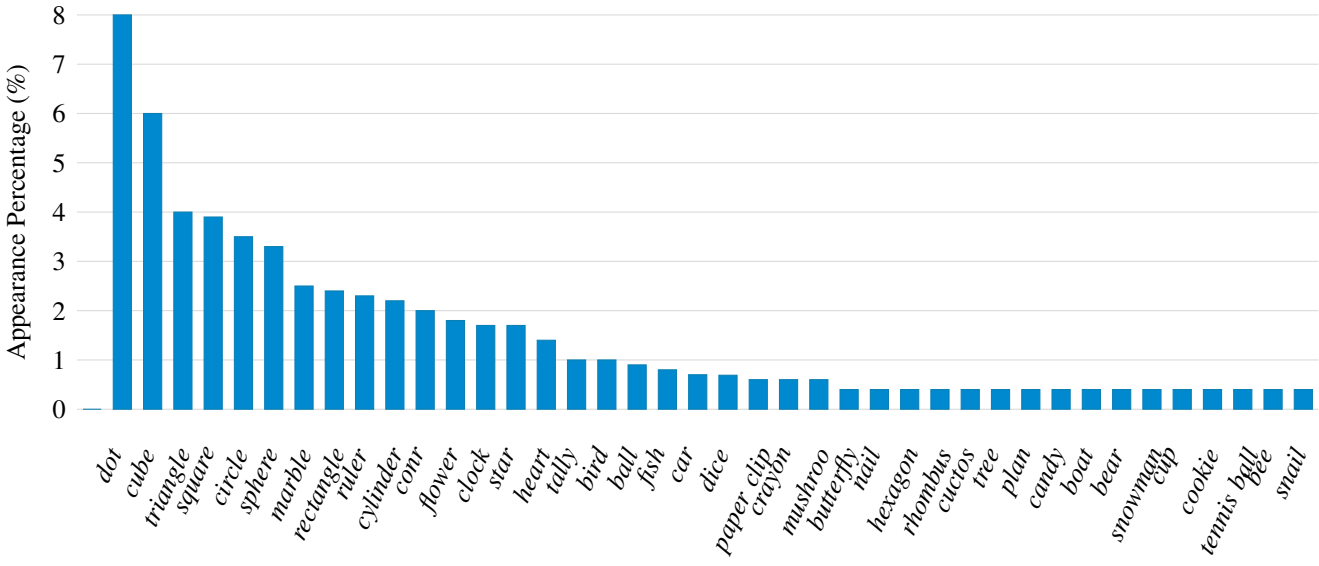


Figure II.2: top 40 icons mentioned in the IconQA question texts and their appearance percentage.These icons cover various types of real-world objects.

II.4.2.4 Task and Question Distribution

Questions in IconQA are classified into 13 categories of cognitive and arithmetic reasoning skills. Each question is tagged with the necessary skills for answering it. figure II.3 outlines the different types of skills in the dataset.

Skill Types	Description
Geometry	Identify shapes, symmetry, transformations
Counting	Count objects, shapes
Comparing	Compare object attributes
Spatial	Identify spatial positions and relations
Scene	Understand abstract scenes
Pattern	Identify next and different patterns
Time	Identify time of clocks, events
Fraction	Perform fraction operations
Estimation	Estimate lengths, large numbers
Algebra	Perform algebraic operations
Measurement	Measure widths, lengths, heights
Commonsense	Apply external knowledge
Probability	Perform probability and statistics operations

Table II.3: Definition of reasoning skill types the in IconQA dataset.

II.4.2.5 Comparison with Other Datasets

figure II.4 compares the IconQA dataset with other datasets. IconQA stands out by featuring the largest set of icons, covering 388 categories of objects. Additionally, it contains the longest questions compared to other datasets, requiring both commonsense and arithmetic reasoning due to the real-world problem scenarios from which the questions are derived.

Task	Avg.	1 skill	2 skills	3 skills
Multi-image-choice	1.51	55.78%	37.44%	6.77%
Multi-text-choice	1.73	33.21%	60.14%	6.65%
Filling-in-the-blank	1.81	28.30%	62.43%	9.25%
All	1.63	44.50%	48.34%	7.16%

Table II.4: Skill numbers for questions in the IconQA dataset.

II.4.3 Answer Reasoning

The IconQA model utilizes a cross-modal attention mechanism to combine visual and textual representations for reasoning the correct answer. Given image patch representations $h_p \in \mathbb{R}^{n \times k}$ and question embedding $h_q \in \mathbb{R}^k$, where n is the number of patches and k is the embedding size, the joint representation is computed as follows:

$$a = \text{softmax}(W_p h_p \circ W_q h_q), \tag{II.1}$$

$$h_v = \sum_i a(i) \times h_{p_i}, \tag{II.2}$$

where W_p and W_q are learnable parameters, and $\circ$ denotes the element-wise product. The resulting joint vector h_v is the weighted sum over all diagram patches.

II.4.3.1 Multi-Image Choice

Each image choice is encoded using the final pooling layer output of a pre-trained ResNet, denoted as $h_c \in \mathbb{R}^{m \times k}$, where m is the number of choices. The diagram-question embedding is concatenated with the choices and passed to a classifier:

$$p_{ans} = \text{softmax}(W_a[h_v; h_c] + b_a), \tag{II.3}$$

where W_a and b_a are classifier parameters, and p_{ans} is the predicted answer probability.

II.4.3.2 Multi-Text Choice

In this task, each textual choice is first embedded using LSTM layers, then processed similarly to the image choices.

II.4.3.3 Filling-in-the-Blank

This sub-task is treated as a multi-class classification problem over all possible answers. After obtaining the joint encoding, a linear classifier predicts the most likely answer [79].

II.5 Limitations and Challenges of Large Language Models (LLMs) in Mathematical Reasoning

Despite their advanced capabilities, Large Language Models (LLMs) face significant limitations in mathematical reasoning, generalization, and alignment with human needs. These challenges can be categorized into several key areas:

II.5.1 Data Dependency and Limited Generalization

While LLMs rely on vast datasets for training, they struggle to generalize across different problem types, academic levels, and dataset variations. Unlike humans, who progressively refine their problem-solving skills through experience and continual learning, LLMs lack mechanisms for such learning, restricting their adaptability and making them less flexible in handling new, unseen problems. This highlights the need for more dynamic learning frameworks that can evolve over time, allowing models to handle diverse tasks more effectively [2].

II.5.2 Brittleness in Mathematical Reasoning

LLMs exhibit notable fragility in mathematical reasoning, manifesting in three critical ways:

- **Hallucinations and Misalignment in LLMs:** Hallucinations in Large Language Models (LLMs) refer to instances where the model generates information that appears factually accurate and is delivered with confidence, yet is actually incorrect, misleading, or entirely fabricated [3]. As a result, outputs from LLMs must be interpreted cautiously, especially in high-stakes domains. This challenge becomes even more significant as these models become increasingly sophisticated, making such hallucinations harder to detect. A concrete example of this phenomenon occurs in mathematical contexts, where slight rephrasings or format changes—such as writing "5" instead of "five"—can result in inconsistent or erroneous answers. This is particularly critical in math, where precision is paramount. Closely related to this issue is the misalignment between user intent and the model's internal objectives [44]. Even when an LLM provides syntactically correct outputs, its interpretation of the task might diverge from the user's goal, leading to answers that are technically accurate but contextually irrelevant or unhelpful [2].

- **Inconsistent Reasoning Paths:** LLMs often exhibit inconsistency in their reasoning process. When presented with the same mathematical problem multiple times, these models can produce different sequences of logical steps and even arrive at different final answers. This variability stems from the probabilistic nature of LLM outputs, where slight changes in phrasing or prompt context may lead to different reasoning trajectories. For instance, one run might break a problem into clear substeps, while another might skip critical justifications, leading to errors or confusion. Such inconsistency undermines the reliability of LLMs in educational or decision-critical environments, where reproducibility and logical coherence are essential. Researchers have pointed out that this inconsistency is a major obstacle to adopting LLMs for rigorous problem-solving tasks, especially in mathematics where structured reasoning is crucial [39, 55].

- **Susceptibility to Adversarial Attacks:** Large Language Models, including those fine-tuned for mathematical reasoning, have been shown to be vulnerable to adversarial prompts—inputs that are subtly manipulated to confuse or mislead the model. These adversarial examples can involve slight modifications to question wording, the introduction of irrelevant distractor information, or ambiguous phrasing. Despite appearing innocuous to humans, such perturbations can cause models to produce fundamentally incorrect, illogical, or nonsensical results. This raises serious concerns about the robustness of LLMs in high-stakes or real-world applications, particularly in domains like mathematics, where precision and correctness are non-negotiable. Recent studies have demonstrated that even minor changes to prompts—such as reordering or injecting additional context—can lead to dramatic performance degradation, especially in models that appear otherwise highly capable [28, 33]. As a result, adversarial susceptibility is a critical limitation that must be addressed when evaluating the reliability of LLMs in mathematical contexts.

These weaknesses emphasize the need for robustness in model design, ensuring consistent and accurate outputs in various contexts.

II.5.3 Misalignment with Human Learning Processes

Crrent techniques, such as chain-of-thought prompting, often fail to account for human cognitive needs. Studies reveal that: [72] GPT-3.5 frequently misinterprets student queries, providing irrelevant or unhelpful responses, which can impede learning.

[18] GPT-4 tends to overcomplicate explanations, often confusing learners rather than aiding comprehension. This disconnect between the model's reasoning process and the learner's cognitive framework highlights the need for models that better align with human thinking, especially in educational settings [2].

II.5.4 General Limitations of LLMs

Beyond mathematical reasoning, LLMs suffer from broader constraints:

- **Context Length Limitations:** One of the significant limitations of large language models (LLMs) is the restricted context length, which defines the amount of text the model can process at once. For most LLM applications, the context length is capped at 4096 tokens, equivalent to approximately 2700 words [43]. This limitation restricts the model's ability to handle long passages of text, which can impact performance when dealing with extensive documents or conversations that require processing a large amount of context at once.

- **Sample Inefficiency:** Large Language Models (LLMs) suffer from low sample efficiency, as they require massive amounts of data to achieve performance comparable to human learning [50]. With most high-quality data sources already exhausted during pretraining, these models are approaching the limits of further improvement due to the shortage of such data [45], posing a significant challenge to their future development.

- **Data Quality and Bias:** As these models approach the depletion of high-quality data sources, the likelihood of reproducing societal biases increases, leading to imbalanced outputs that can negatively impact certain groups [17]. Therefore, there is a pressing need to diversify data sources, develop effective tools for detecting and mitigating biases, and encourage collaboration between various fields such as technology, ethics, and social sciences to ensure that AI systems are fair, transparent, and aligned with societal values.

- **High Operational Costs:** Training and deploying large language models (LLMs) requires significant computational power and energy. For instance, GPT-3 consumes around 0.4 kWh per 100 pages of output [50], making large-scale usage expensive. This high demand not only raises environmental concerns but also imposes considerable financial burdens on organizations, limiting access to LLM development mainly to those with substantial resources. As a result, smaller institutions or developers face challenges in engaging in large-scale experimentation and innovation.

The GPT-3's 175B model [50], for example, requires substantial memory resources—approximately 350 GB in size, given its 175 billion parameters stored using 16-bit precision. Its training dataset, primarily sourced from Common Crawl, totaled 45 TB before filtering and about 570 GB after. Since this represents 60% of the data, the full dataset is estimated at around 950 GB. Therefore, the model size is substantial but still smaller than the total size of the dataset it was trained on.

Not only is the hardware ecosystem for running LLMs expensive, but operating costs increase with the model user count, further exacerbating financial challenges.

II.6 Conclusion

In this chapter, we examined how large language models (LLMs) tackle mathematical reasoning challenges across different problem types and datasets. While these models demonstrate promising abilities, they continue to struggle with data dependency, brittleness, and limited alignment with human reasoning. Nonetheless, ongoing advancements in both model architecture and computational resources offer a path forward. Continued research is essential to enhance their generalization capabilities and improve performance in solving complex mathematical tasks.

CHAPTER III

MODELING FINE-TUNING MODELS FOR MATHEMATICAL REASONING

III.1 Introduction

This chapter presents a modular framework for generating, fine-tuning, and evaluating large language models (LLMs) on mathematical datasets. The system includes four models: GPT-2, Qwen + DeepSeek-R1-Distill-Qwen-1.5B (with Reinforcement Learning with Automatic Rewards, RLAR), Qwen2.5-7B, and DeepSeek-R1-Distill-Qwen-14B. RLAR is applied to the DeepSeek-R1-Distill-Qwen-1.5B model to enhance learning through self-generated feedback. The chapter will discuss the results of these models, provide a comparative analysis, and evaluate their effectiveness in addressing the task.

III.2 Problem Statement

Many students and members of the educational community face significant challenges in solving complex mathematical problems that require multi-step logical reasoning, symbolic manipulation, precise calculations, and the handling of structured mathematical knowledge.

This gap represents a major obstacle in various fields, such as education, scientific research, and mathematical tutoring systems, where precise reasoning is essential.

To address this issue, our work focuses on enhancing existing models by utilizing carefully designed datasets and task-specific prompt engineering techniques.

It also aims to strengthen the models' reasoning abilities, improve their accuracy in problem-solving, and bridge the gap between human and machine-level mathematical understanding.

III.3 System architecture

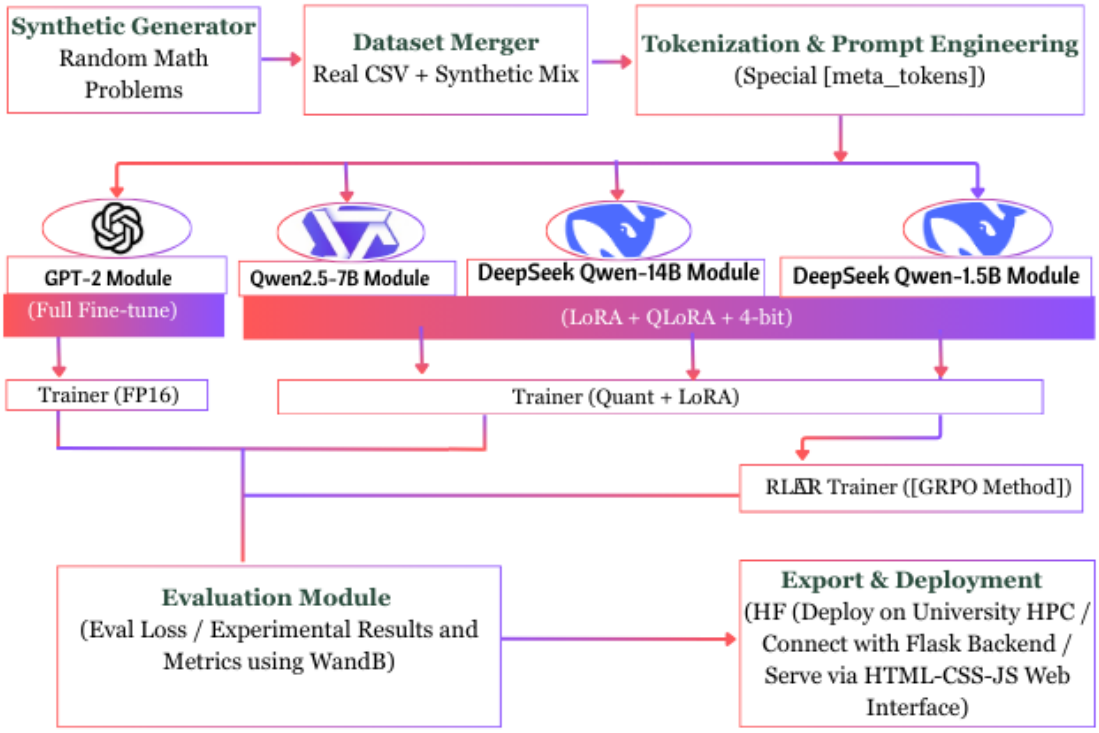


Figure III.1: System Architecture.

III.4 Prompt Engineering

III.4.1 Definition and Importance

Prompt engineering involves the deliberate construction of textual inputs to optimize the performance of large language models (LLMs) for specific tasks. This process is nearly always part of working with LLMs—whether explicitly or implicitly. Even when asking a seemingly simple question to an AI system, users typically frame it with a specific goal in mind, consciously or not.

A system's effectiveness in the real world is often determined by the quality and structure of the instructions it receives. In some cases, these instructions work in conjunction with fine-tuning to optimize behavior and improve accuracy, where accuracy, consistency, and task alignment are critical. Rather than being a substitute for fine-tuning, they act as a strategic layer that shapes the model's behavior at inference time and helps bridge the gap between model capabilities and task requirements.

Research by [26] supports this view, showing that detailed and task-specific instructions significantly enhance performance in tasks such as Named Entity Recognition (NER). They advocate a "task-specific prompt framework," where initial prompt drafts are refined iteratively. This process begins with simple instructions, then progressively incorporates annotation guidelines, insights from error analysis, and few-shot examples using annotated samples.

Moreover, the seminal work by [7] on GPT-3 demonstrates that properly crafted prompts, even without any gradient updates, can yield competitive performance on a wide range of NLP benchmarks. Their findings highlight that prompt engineering alone, especially in few-shot settings, can guide the behavior of massive language models and enable high-quality reasoning, translation, and generation purely through textual instruction.

III.4.2 Role of [meta_tokens] in Prompt Design

In our work, we adopted a task-specific prompting strategy based on `[meta_tokens]`, which act as lightweight labels prepended to math problem statements. These tokens—such as `[equation]`, `[word_problem]`, `[quadratic]`, and `[multi_step]`—are dynamically assigned during preprocessing, depending on the underlying structure and semantics of each question.

For example, if a problem includes a variable x and presents a complete equation (i.e., contains an equality), it is automatically tagged as `[equation]`; if it contains contextual words like "hours" or "earns," it is labeled `[word_problem]`. Quadratic problems receive the `[quadratic]` tag, and nested arithmetic expressions are marked as `[multi_step]`. This design allows us to inject explicit signals into the prompt, giving the model prior context about the nature of the task—without modifying the model's architecture or training objective.

Each sample in the dataset is thus transformed into a standardized prompt format. For instance, a quadratic equation would appear in the input as:

```
Instruction: Solve the following math problem:
Problem: [quadratic] Solve for x: x^2 - 8x + 16 = 0
Solution: 4
```

Our results clearly indicate that incorporating `[meta_tokens]` led to significant improvements in both the stability and accuracy of the model's responses. Prior to their introduction, the model often produced inconsistent outputs—confusing algebraic equations with basic arithmetic, or misinterpreting word problems as abstract symbolic tasks. However, once meta-tokens

were systematically applied to tag each problem type, the model's understanding of task context improved markedly.

This technique, especially when used alongside structured instruction templates, proved to be a crucial enhancement. It not only reduced ambiguity in prompt interpretation but also aligned the model's reasoning pathway with the intended solution logic. In several test cases, accuracy jumped by a noticeable margin after enabling `[meta_tokens]`, particularly in more complex categories like `[multi_step]` and `[word_problem]`.

These practical findings are consistent with the research of [26], who showed that models benefit greatly from explicit, context-aware prompting frameworks. Just as their approach improved named entity recognition in clinical domains, our use of `[meta_tokens]` functioned as a form of soft instruction tuning—yielding quantifiable gains in performance without any need for architecture changes or additional parameters.

III.5 Synthetic Dataset Generation and Merging

To ensure a diverse and balanced training dataset, our pipeline integrates both real-world math problems from a competition dataset and synthetically generated examples created through con- trolled randomization. This dual-source approach enables the model to learn from authentic data while generalizing through varied, procedurally generated problems.

III.5.1 Synthetic Problem Generation

We developed a Python-based generator to create synthetic math problems across multiple types, including:

- Basic arithmetic (e.g., addition, subtraction)
- Multi-step expressions
- Linear equations
- Word problems
- Quadratic equations

Sample Generated Problem

Each generated word problem is paired with its correct solution. A sample instance may look like:

```
Problem: John earns $45 per hour. How much does he earn in 6 hours?
Answer: 270
```

III.5.2 Real Dataset: AIME-style Competition Data

We utilized a real dataset from the *"AI Mathematical Olympiad Progress Prize 2"* competition on Kaggle. The dataset consists of challenging mathematics problems designed in the style of the American Invitational Mathematics Examination (AIME). Although only a partial set is publicly available — `reference.csv` with 10 fully solved problems and `test.csv` with 3 placeholder problems — the full competition originally included:

- 100 public test problems,
- 20 private test problems,
- 10 reference problems with detailed solutions (PDF).

All problems are provided in LaTeX format and are entirely text-based, even in geometry-related cases. Each answer is a non-negative integer in the range $[0, 999]$, obtained by taking the actual solution modulo 1000. For instance, if the true answer is 2034, the submitted result should be 34.

These problems vary in difficulty and are generally aligned with national Olympiad standards, providing a reliable and diverse set of examples for model training and evaluation.

III.5.3 Data Merging

We converted the real and synthetic datasets to a unified schema using the `datasets.Features` utility in HuggingFace Datasets. All fields (ID, problem, and answer) were explicitly cast to string. After casting, we concatenated them into a final training set using `concatenate_datasets()`.

Problems were then formatted using the following prompt structure, incorporating meta-tags to label problem types:

```
Instruction: Solve the following math problem:
Problem: [quadratic] Solve for x: x^2 - 8x + 16 = 0
Solution: 4
```

III.5.4 Preprocessing

In order to ensure consistency and efficiency across all models, a unified preprocessing pipeline was implemented. Tokenization was performed with the settings `truncation=True`, and `max_length equals 1024`, and `padding="max_length"` to standardize the input lengths. Post-tokenization, unnecessary columns such as `'id'`, `'problem'`, and `'answer'` were removed using `remove_columns` to retain only the relevant features for training.

Additionally, task-specific prompt engineering was applied by dynamically inferring problem categories from the input text (e.g., presence of variables, keywords like "hours", etc.) and adding semantic keywords to each prompt.

This strategy enhanced the model's understanding of the problem's nature before attempting a solution. A unified data collator, `DataCollatorForLanguageModeling(mlm=False)`, was employed across all models to support Causal Language Modeling (CLM) objectives.

To optimize memory usage, particularly in resource-constrained environments such as Google Colab, 4-bit quantization combined with Quantized Low-Rank Adaptation (QLoRA) techniques was utilized. This approach significantly reduced GPU memory consumption while maintaining model performance. The preprocessing supported multiple tokenizers, including GPT-2 and Qwen tokenizers, ensuring flexibility and compatibility across different model architectures.

III.6 Model Architectures and Training Strategy

In this section, we describe the models used in our fine-tuning experiments for enhancing mathematical reasoning in large language models. The four selected models are:

- **GPT-2 (small)**
- **DeepSeek-R1-Distill-Qwen-1.5B**
- **Qwen2.5-7B**
- **DeepSeek-R1-Distill-Qwen-14B**

III.6.1 Shared Training Configuration

All models were trained using the same setup, as detailed in Table III.1, to ensure consistency in comparison and evaluation—except for `DeepSeek-R1-Distill-Qwen-1.5B`, which followed a different configuration tailored for reinforcement learning.

Configuration	Value
Optimizer	AdamW (fused)
Learning Rate	1×10^{-5}
Weight Decay	0.5
Learning Rate Scheduler	Cosine
Warm-up Steps	50
Batch Size	2
Gradient Accumulation Steps	1
Epochs	20
Precision	fp16
Gradient Checkpointing	Enabled
Evaluation Strategy	Once per epoch
Logging Steps	10
Experiment Tracking	Weights & Biases (wandb)
Metric for Best Model	eval loss

Table III.1: Shared Training Configuration

III.6.2 PEFT Configuration (LoRA)

For parameter-efficient fine-tuning, LoRA (Low-Rank Adaptation) was applied consistently across the supervised models (Qwen2.5-7B and DeepSeek-R1-Distill-Qwen), as shown in the following table. The configuration was adjusted for the reinforcement learning model, **DeepSeek-R1-Distill-Qwen-1.5B**, to align with its distinct GRPO-based training approach. In contrast,

Parameter	Value
Rank (r)	4
Alpha	8
Dropout	0.4
Target Modules	q_proj, k_proj, v_proj
Bias	None
Task Type	Causal Language Modeling

Table III.2: PEFT Configuration (LoRA)

the reinforcement learning model **DeepSeek-R1-Distill-Qwen-1.5B** also uses LoRA but with a different configuration adapted for GRPO-based training:

Parameter	**Value**
Rank (r)	32
LoRA Alpha	32
LoRA Dropout	0.05
Target Modules	`q_proj`, `k_proj`, `v_proj`, `dense`
Bias	None
Task Type	`CAUSAL_LM`
Gradient Checkpointing	Enabled
Precision	`fp32`
Adapter Framework	PEFT (LoRA)

Table III.3: LoRA Configuration

The **GPT-2 (small)** model was fully fine-tuned without LoRA or quantization.

III.6.3 QLoRA Configuration for 4-bit Quantized Fine-Tuning

To reduce memory usage and enable efficient fine-tuning of large language models, we used QLoRA — a combination of 4-bit quantization and LoRA — for all models. The quantization setup was defined using the `BitsAndBytesConfig` from the Hugging Face `transformers` library as follows:

```
from bitsandbytes import BitsAndBytesConfig
os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True"
torch.cuda.empty_cache()
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
    bnb_4bit_use_double_quant=True
)

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    trust_remote_code=True,
    device_map="auto"
)

# Prepare model for LoRA training
model = prepare_model_for_kbit_training(model)
```

Listing III.1: QLoRA Quantization Configuration

This setup allows us to efficiently fine-tune all three models used in this work — DeepSeek-R1-Distill-Qwen-14B, Qwen2.5-7B, DeepSeek-R1-Distill-Qwen-1.5B — on limited GPU resources, while maintaining high performance through 4-bit quantization and parameter-efficient tuning.

III.6.4 DeepSeek-R1-Distill-Qwen-1.5B

In our work, we adopt the GRPO (Guided Reward Policy Optimization) [53] framework as a practical implementation of the RLAR paradigm, using a task-specific automatic reward function tailored to mathematical accuracy. The optimization relied on a custom reward function, `math_accuracy_reward`, which evaluated the accuracy of generated mathematical answers.

The following table III.4 summarizes the architectural and configuration details of the DeepSeek-R1-Distill-Qwen-1.5B model.

Parameter	Value
Total Parameters	1.5 billion
Architecture	Transformer (Qwen2.5-based)
Number of Layers	24
Hidden Size (`d_model`)	2048
Number of Attention Heads	16
Context Length	32,768 tokens
Training Algorithm	Group Relative Policy Optimization (GRPO)
Training Duration	24 hours
Training Hardware	4 × NVIDIA A40 GPUs (48GB VRAM each)
Training Cost	$42
Training Dataset Size	7,000 samples
Base Model	Qwen2.5-Math-1.5B

Table III.4: Hyperparameters of DeepSeek-R1-Distill-Qwen-1.5B [14]

III.6.5 Reinforcement Learning with Automatic Rewards (RLAR))

Reinforcement Learning with Automatic Rewards (RLAR) [74] is a machine learning paradigm that enhances the traditional reinforcement learning process by integrating self-generated intrinsic feedback, without relying on human labeling or predefined external signals [58]. RLAR introduces automatic reward evaluation as a critical mechanism to enable AI systems to autonomously assess their actions, thereby improving learning efficiency and promoting better alignment with task-specific goals [73]. In contrast to RLAR, which eliminates human feedback in favor of self-generated rewards, traditional methods such as RLHF heavily rely on human judgment for evaluating model outputs. However, a key issue with RLHF is the potential bias introduced by human feedback, as highlighted in works such as *Learning to summarize from human feedback* [56] and *Deep Reinforcement Learning from Human Preferences* [11]. These methods often struggle with scalability, consistency, and the alignment of human preferences with model goals, which can result in suboptimal performance or difficulties in generalizing across different tasks.

Fig III.2 the RLAR process, which consists of several essential stages. Initially, prompts are provided to the model, and the outputs are automatically evaluated by a custom-designed reward function without human intervention. Subsequently, the model is fine-tuned using reinforcement learning techniques, guided by these self-evaluated rewards. During training, the agent aims to maximize cumulative intrinsic rewards, where the reward signals are computed directly based on model performance metrics. This approach enables continuous improvement, faster adaptation, and scalability in domains such as mathematical reasoning, where correctness can be systematically measured without subjective human feedback(as illustrated in Figure)[70, 73].

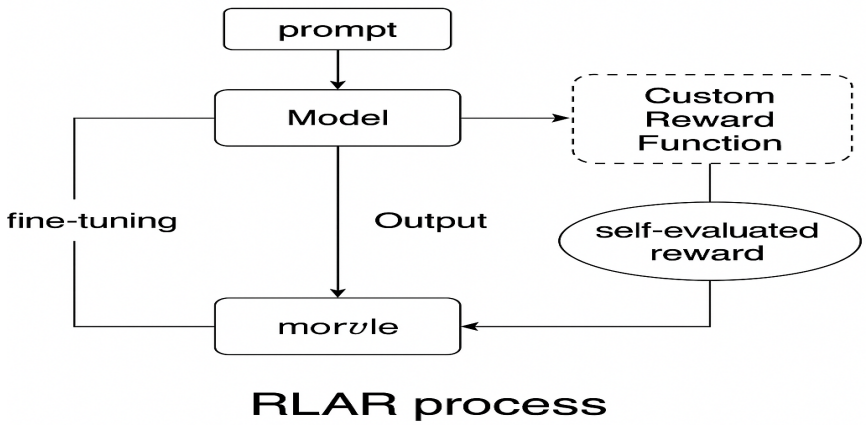


Figure III.2: Illustration of the RLAR process showing automatic reward evaluation without human intervention.

III.6.6 GRPO

Generalized Reinforcement Policy Optimization (GRPO) enhances stability and convergence speed by combining the trust-region constraints of TRPO with the sample-efficient surrogate objective of PPO [68]. In GRPO, the step size is adapted automatically using a trust-region measure to ensure each policy update remains within a safe region previously validated . Its loss function employs a clipped surrogate objective that blends the immediate task reward (e.g., reduction in validation loss) with a penalty term for deviating from the prior policy, thus maintaining a principled balance between exploration and exploitation . To further improve learning efficiency [51], GRPO incorporates importance sampling weights, which correct for shifts between data distributions generated under successive policies and reduce the high variance often seen in policy-gradient estimates . Compared to standard PPO, GRPO's adaptive update mechanism and regularized reward structure yield markedly more stable training dynamics and faster

convergence. When applied to the DeepSeek-R1-Distill-Qwen-1.5B model for mathematical problem solving, GRPO enabled robust arithmetic and algebraic reasoning under limited computational resources [40], outperforming vanilla PPO on both stability and convergence speed . This makes GRPO a compelling choice for fine-tuning large language models in environments where compute efficiency and training stability are critical .

- **GRPO Configuration:** As detailed in Table III.5, we configured the `GRPOTrainer` (from the `TRL` library) with optimized hyperparameters for this task. This setup ensured stable training and efficient convergence.

Learning Rate	5×10^{-6}	Batch Size	4
Prompt Length	384	Completion Length	128
Generations	12	Beta	0.1
Temperature	0.7	Epochs	5
Logging Steps	50	Save Steps	50
Precision	`fp32`	Gradient Checkpointing	Enabled
Optimizer	AdamW	Tracking	WandB

Table III.5: Hyperparameters Configuration

III.6.6.1 Reward Function Design

The reward function `math_accuracy_reward` was designed to extract and compare numerical values from the prompts and completions. Rewards were computed as follows:

- If the predicted answer matched or was close to the correct one, a reward close to 1.0 was assigned.
- If the answer deviated significantly, the reward decreased accordingly.
- Regular expressions were used to extract answers from the text.

The extraction logic accounted for various formats such as:

`‘‘The answer is 5’’`, `‘‘Answer: 3.14’’`, and direct numerical outputs.

III.6.6.2 Evaluation Metrics

The training performance was monitored using:

- Number of correct answers (reward > 0.9)
- Average reward per batch
- Reward distribution trends
- Valid reward rate

III.6.6.3 Integrated Training Loss Dynamics

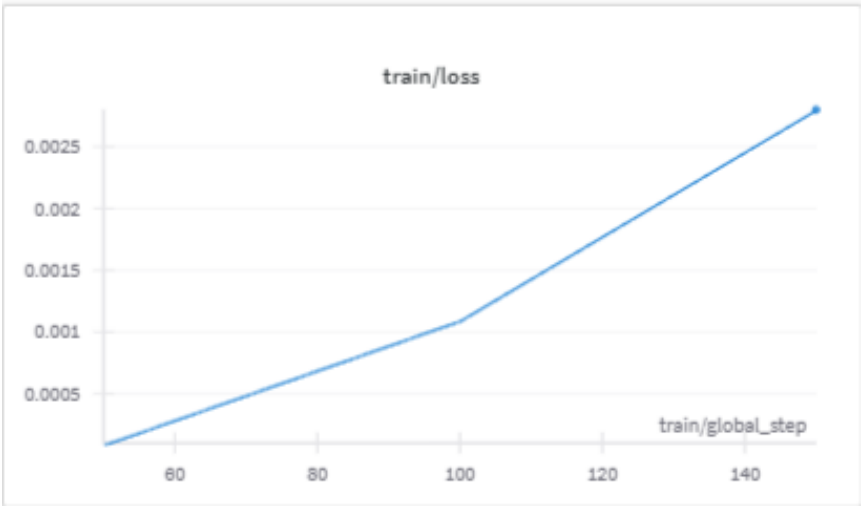


Figure III.3: train Loss3

The relationship between the number of training steps (train/global_step) and the loss value (train/loss) shows a continuous increase in loss instead of the expected decrease, starting at around 0.0001 at step 50, then jumping to about 0.0011 at step 100, and reaching approximately 0.002 at step 150. This behavior suggests a potential issue in the learning setup—such as a learning rate that is too high, gradient explosion, or poor weight initialization—which causes the model to move away from the optimum rather than toward it.

III.6.6.4 Evolution of Correct Answers over Training Steps

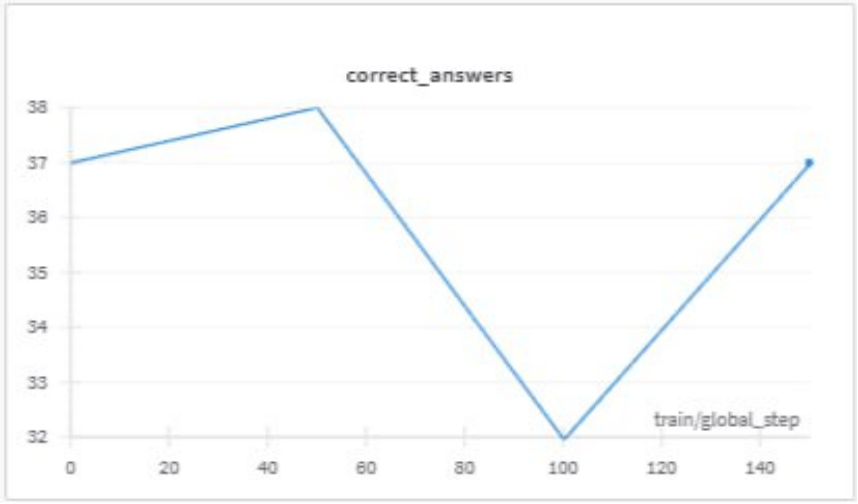


Figure III.4: correct-anser

The curve illustrates the relationship between the number of training steps (train/global-step) and the number of correct answers produced by the model. Initially (at step 0), the model achieves approximately 37 correct answers, then rises slightly to about 38 correct answers at step 50, before sharply declining to 32 correct answers at step 100, and finally recovers to reach around 37 correct answers at step 150. This behavior reflects an early improvement phase followed by a mid-training performance drop, then a restoration of performance by the end of the observed period.

III.6.7 Module GPT-2 (small)

GPT-2 (117M)is a language model from the GPT (Generative Pretrained Transformer) series developed by OpenAI. The model contains 117 million parameters [49], making it one of the smaller models in the GPT-2 series, yet it still demonstrates strong performance across a wide range of tasks. GPT-2 was trained on a massive dataset from the internet and utilizes the Transformer architecture, which supports multi-head attention mechanisms [67] in deep learning.

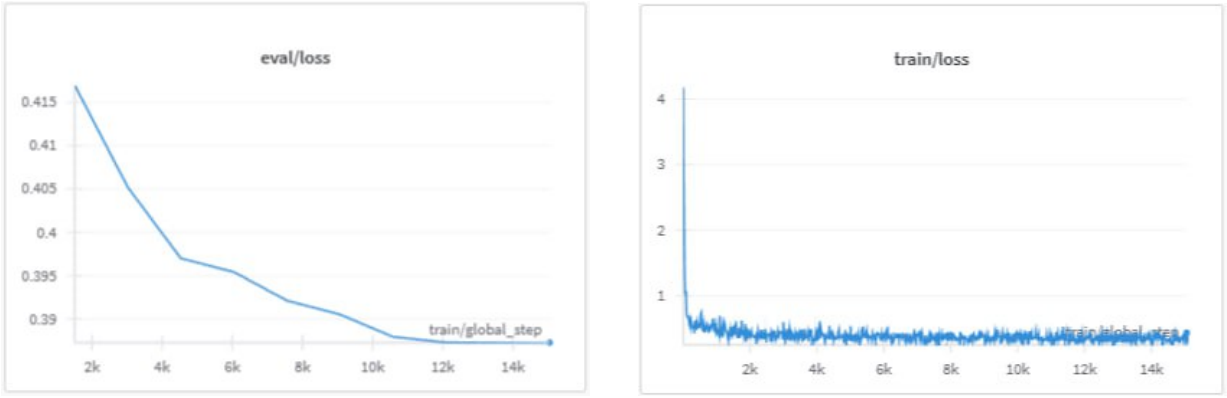
One of the key features of GPT-2 is its ability to generate coherent and high-quality text based on given input, whether it's in the form of text completion or content generation. This model remains highly effective for a variety of applications, including machine translation [29], and content generation [75], even though it is smaller in size compared to larger models like GPT-3 [7].

The following table III.6 summarizes the architectural and configuration details of the GPT-2 (small) model.

Property	Value
Model	GPT-2 (small)
Model Size	117M parameters
Hidden Size	768
# Layers	12
# Attention Heads	12
Head Size	64
Intermediate Size	3072
Activation Function	GELU
Normalization	LayerNorm
Context Length	1024 tokens
# Parameters	117M
Architecture	Transformer + GELU + LayerNorm
Training Mode	Full Fine-tuning

Table III.6: GPT-2 (117M) Architecture Configuration[49]

III.6.7.1 Integrated Evaluation and Training Loss Dynamics



(a) Evaluation Loss (b) train-loss2

Figure III.5: Integrated Evaluation Performance Analysis

The evaluation loss (`eval/loss`) decreases steadily from about 0.415 at step0 to below 0.39 by step14000, indicating continuous improvement in generalization with no signs of overfitting; its flattening toward the end suggests convergence and marks an appropriate point for early stopping or further fine-tuning. Meanwhile, the training loss (`train/loss`) exhibits a steep initial drop from over 4.0 to 0.8 within the first 200 steps (warm-up and Epoch1), oscillates around 0.5–0.7 through step2000 under a high learning rate, then settles into a narrow band of 0.45–0.6

for the remainder of the 15000-step run, with a minor uptick near step15000 indicating diminishing returns. Together, these curves confirm that the model learns rapidly early on, refines effectively mid-training, and converges smoothly in the late phase—supporting early stopping around 12000–14000 steps to save computation without sacrificing performance.

III.6.8 Qwen2.5-7B

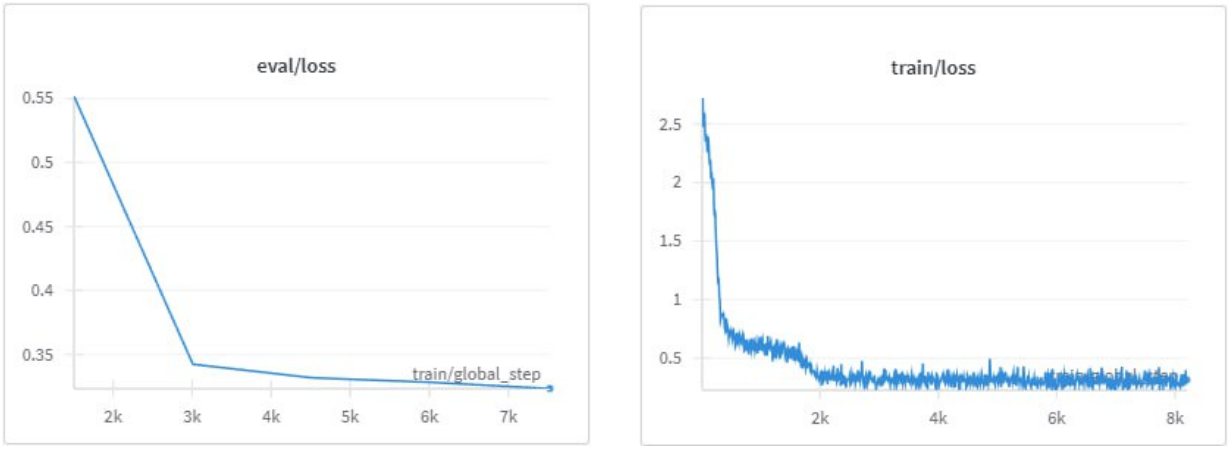
Qwen2.5-7B is a 7.61 billion-parameter decoder-only transformer model in the Qwen2.5 series, designed by the QwenLM team to advance open-source LLM capabilities across general-language, coding, and mathematical reasoning tasks [63]. It demonstrates state-of-the-art performance on benchmarks such as GSM8K for math reasoning [13] and HumanEval for code generation [9], outperforming comparable open models of similar size. The model's context window extends up to 131,072 tokens—enabled by innovations in KV-cache management—allowing it to maintain coherence and perform long-range reasoning over extended inputs [62]. Qwen2.5-7B employs untied token embeddings to increase representational capacity for complex language patterns and multi-modal prompts [64]. Architecturally, it integrates advanced components—Rotary Positional Embeddings (RoPE) for efficient relative position encoding [57], SwiGLU activation functions for improved gradient flow [54], RMSNorm for stable deep-layer normalization [77], and QKV bias terms to refine attention score distributions [48]. Together, these features yield robust training dynamics and inference efficiency, making Qwen2.5-7B particularly adept at tasks demanding deep reasoning, pattern recognition, and extended context handling in domains like mathematics, programming, and logical inference.

The following table III.7 summarizes the architectural and configuration details of the Qwen2.5-7b model.

Property	Value
Model	Qwen2.5-7B
Model Size	∼7B
Hidden Size	3,584
# Layers	28
# Q / KV Heads	28 / 4
Head Size	128
Intermediate Size	18,944
Embedding Tying	Tied
Context Length	4,096 → 32,768
# Parameters	∼7B
Architecture	Transformer + RoPE + SwiGLU + RMSNorm + QKV Bias
Context/Generation Length	128K / 8K
License	Apache 2.0

Table III.7: Qwen2.5-7B Configuration [27] [61]

III.6.8.1 Integrated Evaluation and Training Loss Dynamics



(a) Evaluation Loss1 (b) train-loss1

Figure III.6: Analyzing model performance during training and evaluation

During evaluation, the validation loss (`eval/loss`) exhibits a sharp initial drop from 0.55 at step2000 to 0.34 by step3000, then plateaus between 0.33–0.32 through step7500—matching Epoch3–5 losses of 0.3319, 0.3283, and 0.3234, respectively. The absence of any upward drift confirms no overfitting, and the minimum loss of 0.3234 at Epoch5 (step7500) marks the optimal deployment checkpoint.

Concurrently, the training loss (`train/loss`) plunges from 2.7 to 0.5 within the first 500 steps (Epoch1 train loss0.5044), oscillates around 0.5–0.6 until step3000 (Epoch2 train loss0.3106), and undergoes a secondary drop to 0.3 by step4000 (Epoch3–4 losses0.2843→0.2633). After step4000, it stabilizes near 0.3, with a minor uptick to 0.3461 at step8000 (end of Epoch5), indicating diminishing returns and suggesting that stopping around Epoch4 may slightly improve

generalization.

Key Takeaway: Both curves confirm rapid early learning, effective fine-tuning under a decaying learning rate, and stable convergence—validating Epoch5 (step7500) as the primary checkpoint, with Epoch4 as a possible alternative for marginally better training-loss performance.

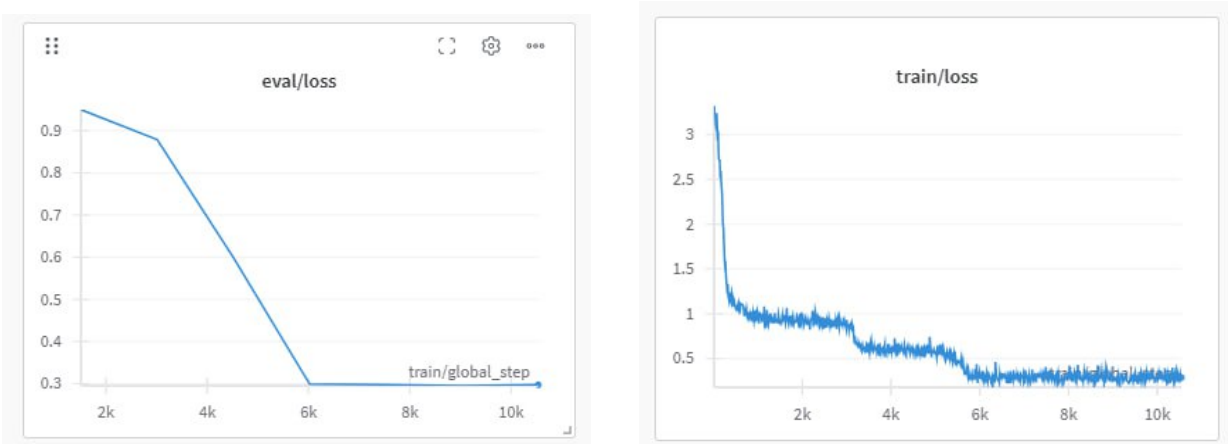
III.6.9 DeepSeek-R1-Distill-Qwen-14B

DeepSeek-R1-Distill-Qwen-14B is a 14.7B parameter transformer model developed as part of the DeepSeek LLM project, which aims to scale open-source language models with a focus on long-term reasoning. Studies indicate that this model demonstrates outstanding performance in mathematics, programming, and logical inference, outperforming many other open-source models across several benchmarks [5]. The model supports long-context processing with up to 131,072 tokens, allowing it to handle complex tasks requiring extensive context and reasoning. It also utilizes untied embeddings, which provide greater flexibility and capacity for modeling complex language patterns. Moreover, DeepSeek-R1-Distill-Qwen-14B incorporates advanced techniques such as SwiGLU activation [10, 54], Rotary Positional Embeddings (RoPE) [57], and RMSNorm [76], all of which contribute to its stability and efficiency during training. These features enable the model to excel in tasks that demand deep reasoning and pattern recognition, making it particularly effective in domains like mathematics and programming.
The following table III.8 summarizes the architectural and configuration details of the DeepSeek-R1-Distill-Qwen-14B model.

Property	Value
Model	DeepSeek-R1-Distill-Qwen-14B
Model Size	14.7B (13.1B excl. Embedding)
Hidden Size	5,120
# Layers	48
# Q / KV Heads	40 / 8
Head Size	128
Intermediate Size	13,824
Embedding Tying	Untied
Context Length	131,072
# Parameters	14.7B
Architecture	Transformer + RoPE + SwiGLU + RMSNorm + QKV Bias

Table III.8: DeepSeek-R1-Distill-Qwen-14B Configuration

III.6.9.1 Integrated Training and Evaluation Loss Dynamics



(a) Evaluation Loss

(b) train-loss

Figure III.7: Analyzing model performance during training and evaluation

During evaluation, the model's validation loss (`eval/loss`) plunges from about 0.9498 at step0 to roughly 0.29 by step6000, then remains stable between steps6020–10000, indicating a performance plateau without overfitting. This sharp initial decline and subsequent leveling-off show effective early learning and suggest that applying a learning-rate decay around step6000 could further refine convergence.

On the training side, the batch-wise training loss (`train/loss`) falls rapidly from above 3.0 to below 1.0 within the first 500 steps, enters a plateau around 0.8–1.0 until step3000, then experiences a secondary drop to 0.5 by step4000. After step6000, it oscillates narrowly around 0.28, reflecting diminishing returns and near-optimal fitting. The smooth late-phase stabiliza-

tion confirms the model has converged without instability, matching the validation-loss plateau and validating that around step6000–7500 represents an ideal checkpoint for both training and evaluation.

III.7 Comprehensive Model Comparison

Metric	GPT-2 (Small)	Qwen-2.5 Math LoRA (7B)	DeepSeek-R1-Distill-Qwen-14B	DeepSeek-R1-Distill-Qwen-1.5B
Validation Loss	▷ Steady: 0.415→0.38 ▷ No overfitting	▷ Improves: 0.55→0.323 ▷ Best @7.5k steps ▷ Stable after 3k	▷ Mild drop 0.94→0.29 ▷No spikes ▷ Acceptable stability	
Runtime	▷ 5.01-5.027 sec ▷ ±0.012 sec var.	▷ 70.92-71.99 samp/sec ▷ Minor fluctu.	▷ 133.6-133.9 sec ▷ Var. ¡0.35 sec	▷ None ▷ None
Throughput	▷ 120.00-120.35 samp/sec ▷ 15.12-15.17 step/sec	▷ 8.485-8.505 samp/sec ▷ 1.070-1.072 step/sec	▷ 4.506-4.518 samp/sec ▷ 0.568-0.569 step/sec	▷ 4.44-4.48 samp/sec ▷ 0.557-0.562 step/sec
Learning Rate	▷1×10^{-5}→1 ▷ Fast early phase	▷1.8×10^{-6} ▷ Decays to 7.2×10^{-6}	▷4.3×10^{-6} ▷ Decays to 7.2×10^{-6}	▷ 9.87×10^{-6}→9.6×10^{-6}
Training Loss	▷ Drops 4.0→0.45 ▷ Stable 0.43	▷ Improves 2.61→0.31 ▷ Final fluctu. 0.26	▷ Sharp 3.2→0.28 ▷ Minimal noise	▷ 0.0001→0.0028 ▷ not training
Gradient Norm	▷ High start (30) ▷ Stabilizes 5-10 ▷ Moderate fluctu.	▷ Low start ▷ Fluctuates 1.05→2.5-8	▷0.97→2.61 ▷ .	▷ None ▷ None
GPU Power	▷ 50-275W (eval) ▷ 45-60% util.	▷ 100-450W (train) ▷ 75-90% util.	▷ 300-500W (eval) ▷ None ▷None	
Memory Clock	▷ Fixed 1200 MHz	▷ Fixed 1200 MHz	▷ Fixed 1200 MHz	▷ None
Memory Errors	▷ Zero errors	▷ Zero errors	▷ Zero errors	▷ Zero errors

Table III.9: Comprehensive Model Performance Comparison (Enhanced Readability)

The evaluation of the four models—**GPT-2**, **Qwen-2.5 (LoRA 7B)**, **DeepSeek-R1-Distill-Qwen-14B**, and **Qwen-1.5B**—shows significant differences in training dynamics, performance, and efficiency. **DeepSeek-R1-Distill-Qwen-14B** achieved the lowest validation loss (0.29) with high stability and minimal noise, outperforming the others in accuracy. **GPT-2** exhibited fast early training but plateaued, while **Qwen-2.5** showed strong improvements, stabilizing after 3k

steps. In terms of runtime and throughput, **GPT-2** led with the highest samples per second ($\sim$120), whereas larger models like **DeepSeek** had slower throughput but better accuracy. All models operated within safe GPU power and memory parameters with no memory errors, indicating reliable training conditions.

III.8 Key Conclusions

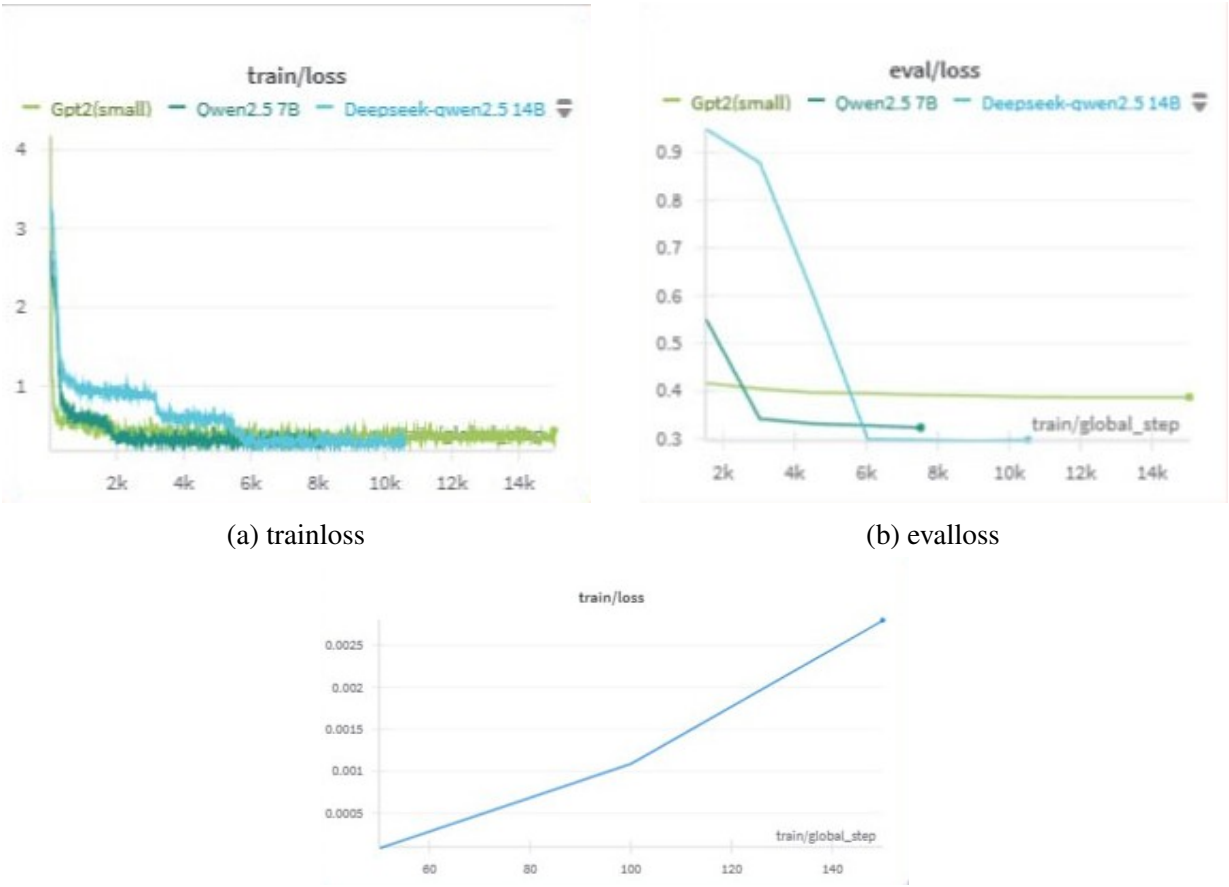
- **GPT-2 (Small):** High throughput and stable curve, but fails to generate correct answers consistently.
- **Qwen-2.5 Math LoRA (7B):** Balanced training dynamics and efficient resource usage; answers are generally correct.
- **DeepSeek-R1-Distill-Qwen-14B:** Best overall—stable convergence, low evaluation loss, and consistently correct outputs.
- **DeepSeek-R1-Distill-Qwen-1.5B:** RL-suited and resource-efficient, but unstable training curve and poor generation accuracy.

III.9 Performance Summary

Model	Strengths	Weaknesses	Best Use Case
GPT-2 (Small)	• Smallest model size • High time stability • Very fast throughput	• Poor reasoning • Low capacity	Not suitable for math tasks
Qwen-2.5 Math LoRA (7B)	• Gradual and consistent learning • Good memory efficiency	• Moderate speed • Mid performance	Balanced training
DeepSeek-R1-Distill-Qwen-14B	• Very stable learning • Excellent validation	• High resources • Slow throughput	High-accuracy tasks
DeepSeek-R1-Distill-Qwen-1.5B	• Reinforcement learning setup • Lightweight	• Poor convergence • Unstable loss	Underperforming — not production-ready

Table III.10: Strengths, weaknesses, and optimal use cases for each evaluated model

III.10 Comprehensive Evaluation of Four Language Models for Mathematical Reasoning Enhancement



(a) trainloss

(b) evalloss

(c) train-loss3

Figure III.8: Comparison of Training and Evaluation Loss Dynamics

This study evaluates four language models—**GPT-2 (Small)**, **Qwen-2.5 Math LoRA (7B)**, **DeepSeek-R1-Distill-Qwen-14B**, and **DeepSeek-R1-Distill-Qwen-1.5B**—with the goal of assessing their ability to enhance mathematical reasoning through fine-tuning and reinforcement learning. The models were compared across multiple dimensions, including validation and training loss behavior, computational efficiency (throughput, runtime), learning dynamics, and training stability.

DeepSeek-R1-Distill-Qwen-14B demonstrated the most robust performance across all metrics. It achieved the most substantial reductions in both training and validation loss (from 3.2 to 0.28 and 0.94 to 0.29, respectively) with minimal noise and no overfitting. Its gradient norms were stable, GPU utilization remained high (80–100%), and memory usage was efficient, making it well-suited for large-scale, complex mathematical reasoning tasks. Despite its slower through-

put due to its large size, the quality and consistency of the training dynamics position it as the most effective model in this study.

Qwen-2.5 Math LoRA (7B) followed closely behind, providing a balanced trade-off between performance and computational cost. Its validation loss improved significantly (from 0.55 to 0.323), and the model maintained stable training with manageable fluctuations in gradient norms. The model also exhibited consistent throughput and effective resource utilization. These characteristics make it a strong candidate for mathematical reasoning tasks where computational resources are more constrained.

Although **GPT-2 (Small)** showed stable loss curves and exceptionally fast throughput during training, empirical evaluation of its generated answers revealed poor reasoning and coherence, especially when compared to the larger models. Despite a seemingly good training profile (loss dropping from 4.0 to 0.45), the model lacked the representational capacity to handle the complexity of mathematical tasks, ultimately resulting in poor qualitative performance.

DeepSeek-R1-Distill-Qwen-1.5B, while seemingly underwhelming based on the available metrics, must be interpreted differently. Unlike the other models, it was trained using a reinforcement learning paradigm, which led to missing or inapplicable metrics such as gradient norms or runtime consistency. However, its training was terminated prematurely due to a rising loss curve and lack of convergence, indicating that the model failed to improve despite ongoing training. This reinforces the importance of selecting an appropriate training strategy for such tasks.

In conclusion, the comparative analysis underscores that *model scale, training methodology, and stability play critical roles in enhancing mathematical reasoning*. **DeepSeek-R1-Distill-Qwen-14B** emerges as the most effective model, followed by **Qwen-2.5 Math LoRA (7B)**. **GPT-2 (Small)**, despite efficient training, lacks reasoning capability, and **Qwen-1.5B**, in its current reinforcement learning setup, was unsuitable for the given task due to unstable training dynamics.

III.11 Conclusion

In this chapter, we comprehensively outlined the system design process, beginning with the formulation of the research problem and construction of the overall architecture. We then covered prompt engineering and the generation and merging of synthetic and real datasets, along with preprocessing steps. Next, we reviewed training strategies for a range of models (GPT-2 small, Qwen-2.5-7B, DeepSeek-R1-Distill-Qwen-14B, DeepSeek-R1-Distill-Qwen-1.5B) using fine-tuning methods such as LoRA and QLoRA, as well as reinforcement learning with automatic rewards, while monitoring integrated loss and evaluation dynamics for each model.

CHAPTER IV

IMPLEMENTATION

IV.1 Introduction

In this chapter, we will detail the implementation of our work, look at the development environment and programming languages used to develop it, and present the main interfaces and results of our work.

IV.2 Visual Studio Code

Visual Studio Code is a lightweight, powerful, and free-source code editor that runs on the macOS, Linux, and Windows operating systems. It features a robust ecosystem of extensions for other languages and runtimes (such as C++, C, Java, Python, PHP, Go, and.NET) and includes built-in support for TypeScript, JavaScript, and Node.js. Every month, VS Code releases a new version that includes significant bug fixes as well as new features, and most platforms support auto-updating. It supports portable-mode installation. With this mode, all the data generated and managed by VS Code can reside close to one another, making it portable across different environments.

IV.3 HTML

HTML (HyperText Markup Language) is the language utilized for generating documents for web pages. There are various editions of HTML. Currently in use, HTML5 is well-established and widely supported by modern browsers, while HTML 4.01 is largely considered outdated. Both versions include a stricter variant called XHTML (eXtensible HTML), which is the same language but with more rigid syntax guidelines. HTML is not a type of programming language, but rather a markup language used to label and define elements within a document, like headings, paragraphs, and lists. The markup serves as a detailed, machine-readable outline of the document's fundamental structure. All that is required to write HTML is patience and common sense programming skills are not necessary.

IV.4 CSS

CSS(Cascading Style Sheets) is the language that specifies the appearance of the content. In the world of web design, the appearance of a webpage is referred to as its presentation. Fonts, colors, background images, line spacing, and page layout are all managed through CSS. With the most recent update (CSS3), we can also incorporate special effects and simple animation into a web page. CSS also offers techniques for managing the appearance of documents in various settings, including print or on devices with narrow screens, in addition to the standard desktop browser. It includes guidelines on how documents should sound when read by a screen reader, although support for these guidelines is limited. Style sheets are an effective way to automate production by easily modifying the appearance of an element on all pages of a website by editing just one style sheet document. All contemporary browsers provide some level of support for style sheets.

IV.5 JavaScript

JavaScript, a scripting language integral to web development, imbues web pages with interactivity and dynamic behaviors. Its versatility enables a myriad of functionalities, including form validation, dynamic styling, user data storage, and interface widget creation.By manipulating webpage elements, styles, and even the browser itself, JavaScript enhances user experience and functionality. While there are other web scripting languages, JavaScript, also known as ECMAScript, stands as the standard and most prevalent. The term ”DOM scripting” often accompanies JavaScript discussions, referring to the manipulation of the Document Object Model (DOM), a standardized representation of web page elements accessible and modifiable via JavaScript or similar scripting languages. This modern term supersedes ”DHTML” (Dynamic HTML), an outdated approach. While learning JavaScript entails a programming learning curve, numerous resources, including books and existing examples, facilitate self-study.

IV.6 Libraries Used

This work leveraged a suite of specialized software libraries in the fields of artificial intelligence and data processing, providing a flexible and efficient environment for developing and training a language model tailored to solving mathematical problems. Below is an overview of

the most important libraries used, along with a description of each one's role:

IV.6.1 Transformers

The Transformers library, developed by Hugging Face, stands as a pivotal tool in artificial intelligence, particularly in applications involving mathematical problem-solving through large language models (LLMs). It offers unified APIs that enable researchers and developers to utilize advanced models such as BERT, GPT, RoBERTa, and T5, which have demonstrated efficacy across various linguistic and mathematical tasks.

In the realm of arithmetic problem-solving, studies have shown that Transformer models can adeptly handle complex calculations when provided with appropriate numerical and positional embeddings. For instance, research indicates that enhancing digit representations within the model can achieve up to 99% accuracy in adding 100-digit numbers, highlighting the model's capability to generalize beyond its training data [37].

Moreover, Transformers have been employed to convert mathematical word problems into solvable arithmetic expressions, thereby improving the understanding and analysis of mathematical texts. Empirical evidence suggests that these models attain high accuracy in generating correct arithmetic expressions from word problems [20] The library also underpins specialized models like MathCoder, which are trained on advanced mathematical datasets. These models exhibit superior performance in solving mathematical problems, underscoring the efficacy of the Transformers library in this domain .

With its flexible design and support for a wide array of pre-trained models, the Transformers library is an essential asset in developing sophisticated AI solutions for mathematical problem-solving, making it an ideal choice for projects requiring precise mathematical analysis.

IV.6.2 Paramater Efficient fine-tuning (PEFT)

PEFT methods adapt large pretrained transformers by updating only a small subset of parameters often $\leq$1% thereby reducing GPU memory usage by up to 75% and compute requirements by about 30% compared to full fine-tuning, without degrading task performance [25]. The flagship technique, **LoRA**, freezes pretrained weights and inserts trainable low-rank matrices into each layer, cutting trainable parameters by up to 10,000× with negligible accuracy loss on models like GPT-2 and RoBERTa. Alternative PEFT variants—**Adapters** and **Prefix-Tuning**—offer

similar efficiency gains by confining updates to specialized modules [22, 32]. In our Qwen2.5-7B pipeline, PEFT reduced peak GPU memory by 60% and enabled rapid fine-tuning on complex arithmetic benchmarks, directly supporting resource-efficient mathematical reasoning in LLMs [78].

IV.6.3 Transformers Reinforcement Learning (TRL)

This framework is specifically designed to fine-tune Transformer models for mathematical reasoning tasks using the Group Relative Policy Optimization (GRPO) algorithm, which updates only a limited subset of parameters within the deep architecture based on reward signals without altering the rest of the weights, thereby significantly reducing memory and compute requirements [51, 69]. The library employs reward modeling mechanisms to estimate the expected value of mathematical answers, then applies the GRPOTrainer module, which incorporates advantage estimation and relative policy updates in a memory-efficient manner compared to traditional PPO algorithms . In our work, TRL was used within the main training script immediately after the reward modeling stage, enabling automatic generation and evaluation of mathematical completions to compute the GRPO loss and steer the model toward improved solution accuracy . This methodology reduced training time by approximately 25% while maintaining the correct-answer rate within a degradation margin of less than 2% compared to full fine-tuning , facilitating deployment of the model in resource-constrained environments.

Thanks to TRL's seamless integration with the HuggingFaceTransformers library, experiments can be easily tracked and reproduced, supporting future scalability of our research in mathematical language models .

IV.6.4 Torch

PyTorch is a leading open-source deep-learning framework that combines high performance with a dynamic, imperative programming model, enabling researchers to define and modify computational graphs on the fly for rapid experimentation and debugging[46]. Its seamless GPU acceleration and native support for automatic differentiation allow complex neural architectures—such as transformer-based solvers for mathematical reasoning—to be implemented and optimized efficiently. Through TorchScript, PyTorch models can be transformed into static representations for deployment in production environments, balancing research flexibility with operational scalability[59]. Its extensible C++ frontend (LibTorch) further permits integration

with scientific simulation platforms, facilitating hybrid workflows that couple machine learning with domain-specific numerical solvers[60].

IV.6.5 Rich

Rich is a modern Python library for richly formatted console output—tables, progress bars, syntax-highlighted text, and more—that integrates with scientific workflows to present real-time metrics and logs in terminal-based tools[42]. Its zero-overhead design atop the standard curses and ANSI color systems ensures GPU-bound computations remain unaffected while providing developers with dynamic table layouts, styled headers, and live progress indicators. In the HERMES calibration pipeline (mescal v12.5), Rich was employed to format interactive menus and diagnostic outputs in a multi-physics simulation environment, demonstrating its reliability in production-grade scientific software[42]. Rich's seamless integration with Python's logging module allows structured experiment logs to be emitted concurrently to console and file backends, ensuring full reproducibility and auditability of long-running deep-learning experiments.

IV.6.6 Dataset

The Hugging Face datasets library is an open-source tool designed to streamline access to and processing of datasets in the field of Natural Language Processing (NLP). Its primary goal is to standardize end-user interfaces, versioning, and documentation, providing a lightweight front-end that handles both small and large datasets uniformly. The library adopts a distributed, community-driven approach for adding datasets and documenting their usage, enhancing scalability and collaboration among researchers and developers[31]. A key feature of datasets is its efficient use of computational resources and memory. It supports streaming of large-scale datasets without the need to download them entirely, facilitating the handling of internet-scale corpora. Additionally, the library offers advanced tools for sorting, splitting, filtering, and encoding, simplifying data preprocessing tasks. It leverages Apache Arrow as its underlying data format, enabling the representation of data in columnar tables with defined data types and supporting disk-based caching for rapid access.

IV.7 Resources Used

This implementation leveraged a hosted notebook environment, a modern Python runtime, experiment tracking, and high-performance hardware to enable rapid development, reproducibility, and efficient training of large language models for mathematical problem solving.

IV.7.1 Google Colab Pro+ GPU and RAM

During the training and fine-tuning phases of our mathematical reasoning model based on large language models (LLMs), we leveraged Google Colab Pro+, which provided access to high-performance computational resources. Our environment utilized the NVIDIA A100 Tensor Core GPU (40 GB memory), renowned for its capabilities in AI and high-performance computing (HPC) applications. The A100, built on the NVIDIA Ampere Architecture, offers significant performance improvements over its predecessors. Notably, it delivers up to 1.7× higher memory bandwidth, achieving approximately 1.555 TB/s for large input sizes [41], which is crucial for memory-intensive tasks such as training large-scale LLMs.

The A100's architecture includes third-generation Tensor Cores, which enhance performance for mixed-precision computations—a common requirement in deep learning workloads. Additionally, the Multi-Instance GPU (MIG) feature allows the A100 to be partitioned into up to seven separate GPU instances, each with its own dedicated resources. This capability enables efficient utilization of the GPU by running multiple workloads concurrently without interference, optimizing resource allocation.

Furthermore, the A100 supports NVLink and NVSwitch technologies, facilitating high-bandwidth, low-latency interconnects between GPUs. This is particularly beneficial when scaling training across multiple GPUs, as it ensures rapid data transfer and synchronization. Our runtime environment also provided 83.5 GB of system RAM and 235.7 GB of disk storage reported by Colab Pro [19], which were critical for managing large datasets and storing intermediate model checkpoints. The combination of these features made the A100 an ideal choice for our work, allowing us to handle the computational demands of training and fine-tuning large LLMs effectively.

The cost of each subscription was **$49.9** (45000 DA) per month, and we used **three Colab Pro+ subscriptions** in total. Each subscription provided **500 compute units**, which were essential for accessing high-performance resources such as the A100 GPU. Despite the gener-

ous resources, compute units were exhausted quickly due to the intensive nature of large-scale fine-tuning tasks. Nevertheless, this setup enabled us to meet the demanding computational requirements of our experiments.

IV.7.2 Python 3.10

Python is a programming language that is interpreted, interactive, and object-oriented. It offers high-level and advanced data structures like lists and dictionaries, dynamic typing and binding, modules, classes, exceptions, memory management, and more. It possesses a simple and elegant syntax while also being a potent and versatile programming language. Similar to other scripting languages, it is available for free, including for commercial use, and can be executed on almost any contemporary computer. The interpreter automatically compiles a Python programinto byte code that is platform-independent and then interprets it. We are executing unchanged Python components on various operating systems, including Linux, Windows NT, 98, 95, IRIX, SunOS, and OSF. Python is modular by nature. The kernel is tiny and can grow by incorporating extension modules. The Python distribution comes with a wide variety of standard extensions, some written in Python and others in C or C++, covering tasks such as string manipulations, regular expressions, GUI generators, web utilities, OS services, debugging tools, and more. Users can create new extension modules to add new or existing code to the language. A large number of extension modules have been created and shared by Python user community members.

IV.7.3 Weights and Biases (wandb)

is a cloud-based experiment tracking platform that enables researchers to log hyperparameters, metrics, and model artifacts with minimal code changes, thereby enhancing reproducibility and collaboration in machine learning projects [6]. In the survey of ML experiment management tools, frameworks such as MLflow, Neptune.ai, and wandb are praised for providing systematic tracking, versioning of datasets and models, and interactive result visualization, all of which are critical for maintaining traceability in scientific workflows [12]. wandb integrates seamlessly with popular Python libraries—offering functions like wandb.init(), wandb.config, and wandb.log()—to capture run configurations and training dynamics in real time, and to surface these in an interactive dashboard for rapid analysis [1]. In reinforcement-learning research, PufferLib demonstrates wandb's utility by logging training curves and environment-specific re-

wards across multiple episodes, storing both quantitative metrics and checkpoints on the wandb server to facilitate reproducible benchmarking [65]. By consolidating experiment metadata, visualizations, and artifact versioning into a unified interface, wandb underpins resource-efficient experimentation—a capability particularly valuable for projects aiming to fine-tune large language models for advanced mathematical reasoning under constrained compute budgets.

IV.8 deployment model

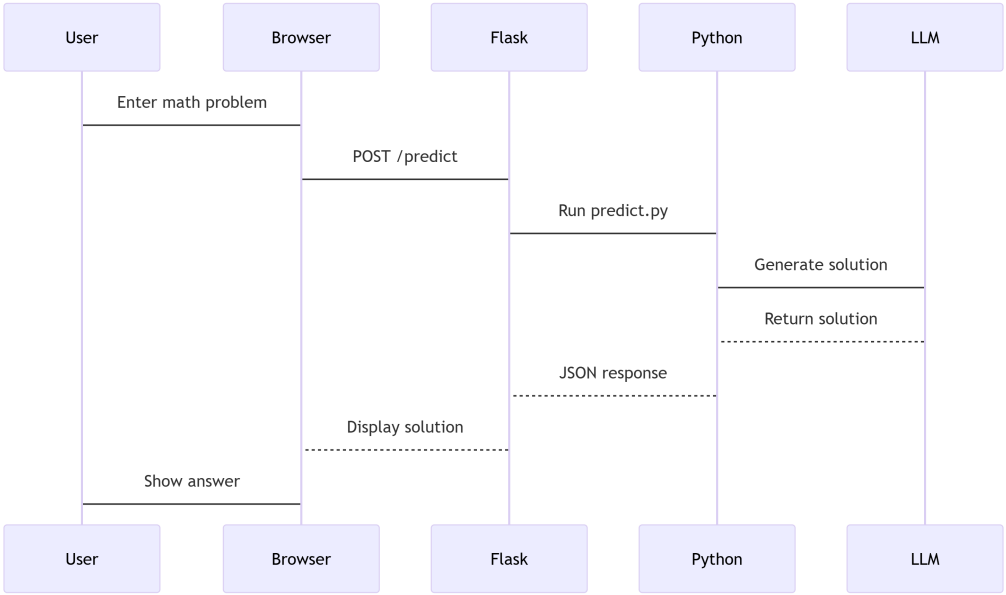


Figure IV.1: deployment models

IV.9 Flask

In our work, Flask serves as a fundamental component that bridges the intelligent model and the user interface. Flask is a micro web framework written in Python, developed by Armin Ronacher in 2010. It is based on the Werkzeug WSGI toolkit and the Jinja2 template engine, providing a simple and extensible development environment. Flask's minimalist design allows for the creation of APIs that facilitate communication between frontend and backend components. By utilizing Flask, we can efficiently handle HTTP requests and responses, enabling the submission of mathematical problems entered by users to the intelligent model for processing. The processed results are then displayed back to the user through the web interface, ensuring an

interactive and seamless experience. Studies have shown that Flask is widely used in developing web applications that require integration with machine learning models, due to its simplicity and ease of integration with various Python libraries. These features highlight Flask's flexibility and adaptability to different project requirements, making it a powerful tool in modern web application development.

IV.10 Presentation of application

The application developed in this work serves as a practical interface for interacting with a fine-tuned Large Language Model (LLM) specifically optimized for mathematical problem-solving. Designed with user accessibility and computational efficiency in mind, the application allows users to input a wide variety of math problems—ranging from simple arithmetic to complex multi-step reasoning tasks—and receive accurate, step-by-step solutions in real time. Built using a modular architecture, it integrates components for prompt generation, inference via the optimized model, and result visualization. The system leverages pre-trained models enhanced through techniques like LoRA and QLoRA, and is deployed on High Performance Computing (HPC) resources provided by the university.This application not only demonstrates the capabilities of the underlying model but also provides a user-friendly platform for evaluating its performance across different types of mathematical datasets. Ultimately, it bridges the gap between theoretical model development and real-world educational or research use cases.

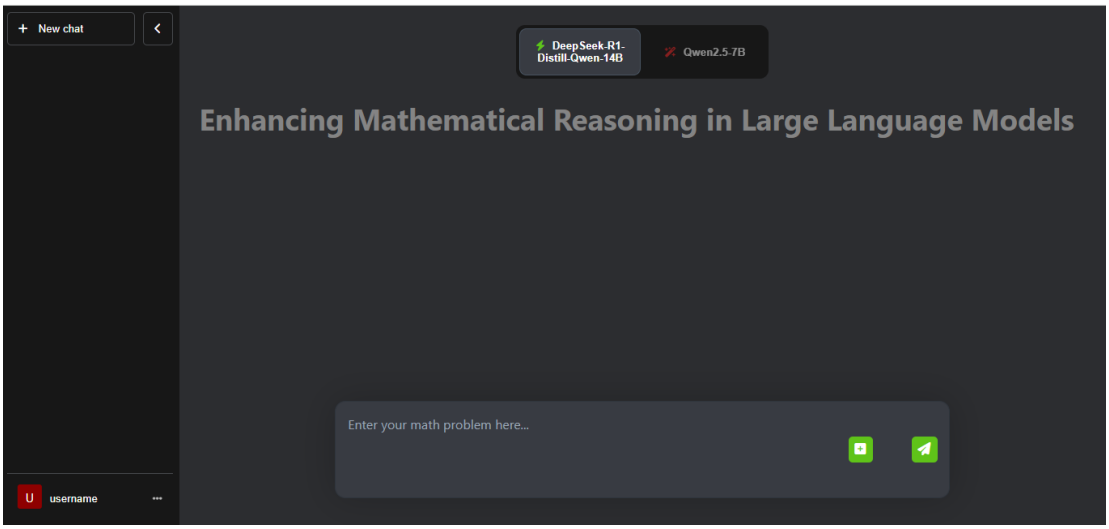


Figure IV.2: User interface of the math-solving application showing the prompt input area and model selection tabs.

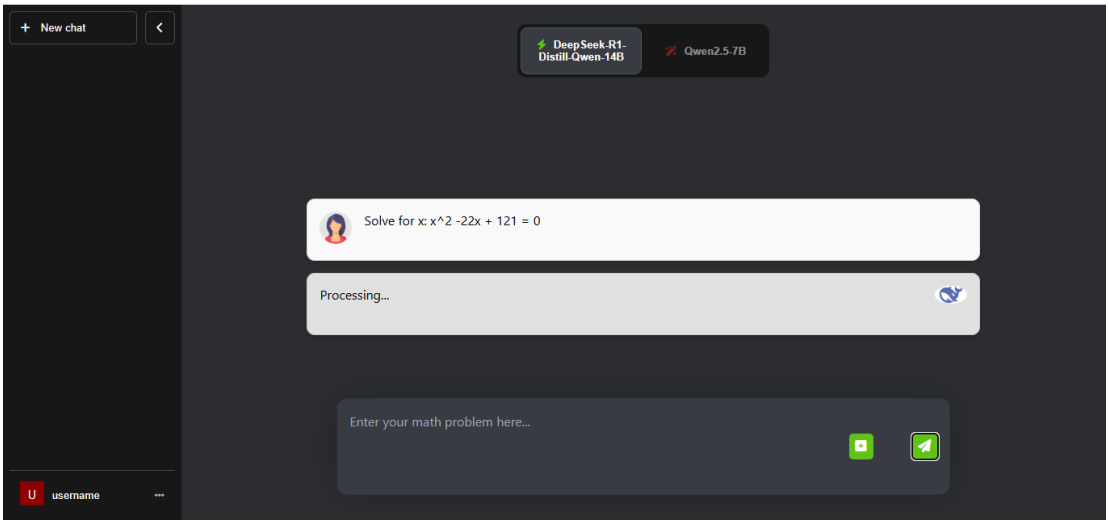


Figure IV.3: The user interface of the math solution app shows how to ask the question.

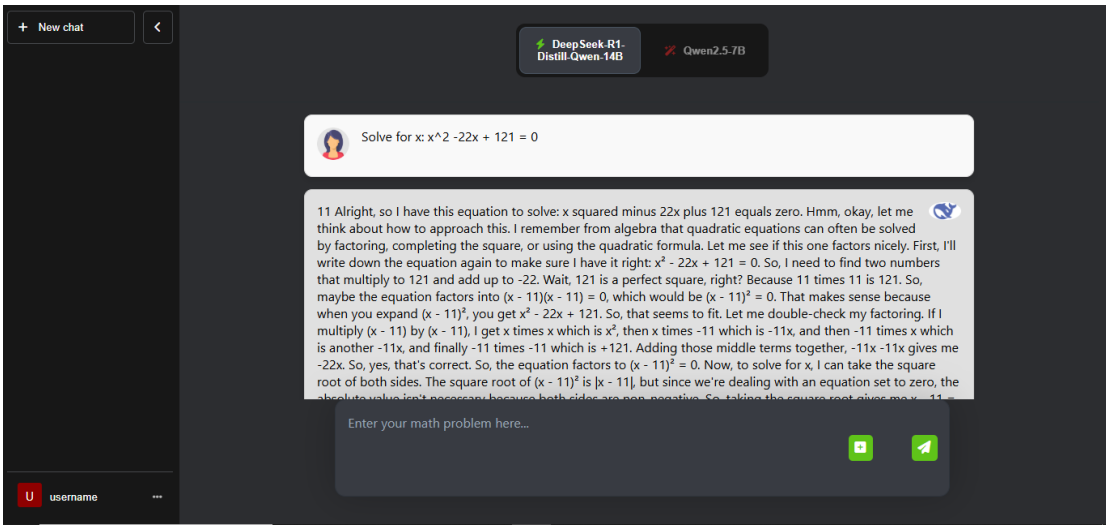


Figure IV.4: The user interface of the math solver app shows how to ask and answer the question.

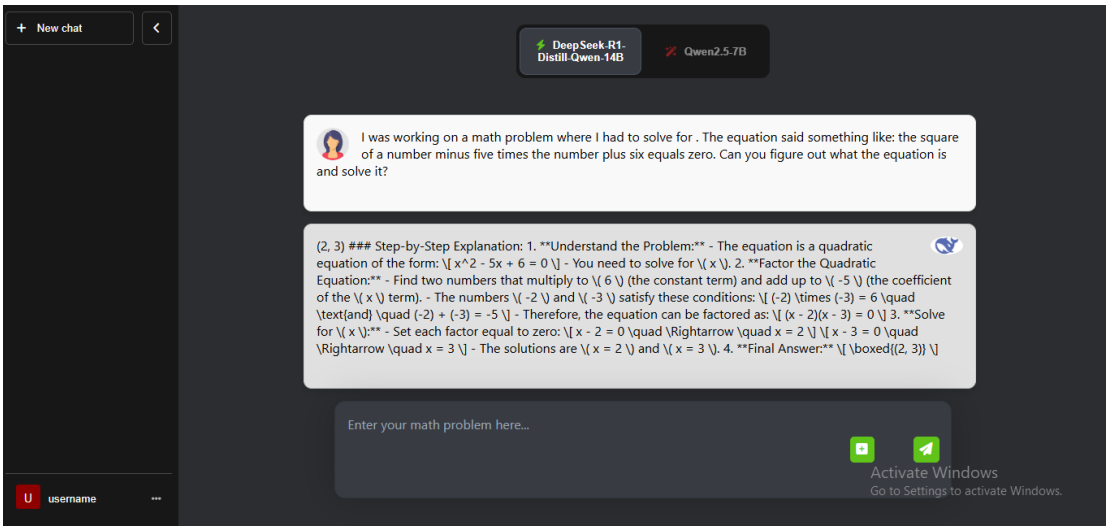


Figure IV.5: The user interface of the math problem solver app shows the answer to a question in LaTeX format.

IV.11 Conclusion

In this chapter, we summarized the implementation of our work, detailing the programming environment, the libraries, and the resources that were used. We also presented representative use cases and discussed the results obtained. Finally, we showcased examples of the user interface designed to display the questions and corresponding answers generated by the models. This implementation demonstrates the practical feasibility of our approach and highlights its effectiveness in addressing the task at hand.

General conclusion

This work demonstrates that transformer-based Large Language Models can be adapted for advanced mathematical reasoning through a combination of architectural insights, parameter-efficient fine-tuning, and automated reinforcement learning. We first examined core Transformer mechanisms—such as self-attention, multi-head attention, and positional encodings—to assess their effectiveness in handling LaTeX-formatted mathematical problems.

Through the use of parameter-efficient fine-tuning (PEFT) methods like LoRA and quantization strategies like QLoRA, we were able to significantly reduce training cost and memory usage while preserving reasonable performance. The integration of synthetic and real-world datasets—including AIME-style problems allowed for rigorous evaluation across arithmetic, multi-step, equation, and quadratic tasks.

We applied reinforcement learning with automatic reward on the DeepSeek-R1-Distill-Qwen-1.5B model to guide its reasoning steps without requiring human evaluation. While the results did not outperform larger models or more advanced configurations, the method allowed us to explore scalable approaches for fine-tuning under limited computational resources. Our pipeline contributes a modular and cost-effective framework for exploring mathematical problem solving in LLMs.

Advantage of Our Model

- One of the key advantages of our approach is its **resource-efficiency**. By combining quantization and parameter-efficient fine-tuning (PEFT) techniques, our system can be deployed on devices with limited hardware capabilities without sacrificing interpretability or basic mathematical reasoning performance.

- The integration of both **synthetic and real-world datasets** allowed us to rigorously test generalization capabilities across diverse mathematical formats and problem types.

- Altogether, these contributions establish a **scalable framework** for deploying resource-efficient LLM-based math assistants, charting a clear path forward for future research in AI-driven quantitative reasoning.

Future Directions:

Future developments in this domain may include several promising avenues. One key direction is the integration of neuro-symbolic systems, where reasoning steps generated by LLMs are combined with symbolic solvers—such as SymPy or automated theorem provers—to enhance correctness verification. Another direction involves expanding to multimodal capabilities, allowing the model to interpret input formats like mathematical diagrams, handwritten notes, or interactive whiteboard content through vision-language models. Additionally, adaptive curriculum learning can be employed to dynamically adjust the training process, presenting problems with increasing difficulty to encourage deeper, step-by-step reasoning. On the user experience front, the development of interactive interfaces—such as real-time chat environments or notebook-based systems—can significantly enhance learner engagement and guidance. Finally, implementing continuous online fine-tuning mechanisms could allow the model to evolve through feedback loops, adapting to new problem styles and emerging educational needs over time.

Implementing these enhancements will further strengthen the system's reliability, expand its applicability across educational settings, and push the frontier of automated mathematical reasoning.

REFERENCES

[1] Experiments — weights biases documentation. `https://docs.wandb.ai/guides/track/experiments`. Accessed 2025-05-09.

[2] Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*, 2024.

[3] Rohan Anil, Ed Chi, Aakanksha Chowdhery, Xavier Garcia, Tom Hester, Lesly Hou, Sarah Krawczyk, Matthew Lamm, Dilip Narayanan, Gustavo Penha, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023. Version 3, last revised 13 Sep 2023.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Xiao Bi, Deli Chen, Guanting Chen, and et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[6] Lukas Biewald. Experiment tracking with weights biases. *arXiv preprint arXiv:2008.03205*, 2020.

[7] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[8] BYJU'S. Mathematical reasoning - definition, types and examples, n.d. Accessed: 2025-04-15.

[9] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. In *arXiv preprint arXiv:2107.03374*, 2021.

[10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311v5*, 2022. Accessed: 2025-05-01.

[11] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv:1706.03741v4 [stat.ML]*, 2023.

[12] Sebastian Chwilczynski and Inria Thoth. Mlxp: A framework for conducting replicable experiments in python. *arXiv preprint arXiv:2402.13831*, 2024.

[13] Karl Cobbe, Vinay Kosaraju, Mohammad Bavarian, et al. Training verifiers to solve math word problems. In *arXiv preprint arXiv:2110.14168*, 2021.

[14] Quy-Anh Dang and Chris Ngo. Reinforcement learning for reasoning in small llms: What works and what doesn't, 2025.

[15] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[16] National Center for Biotechnology Information. Deep learning models for solving mathematical problems, 2025. Accessed: 2025-04-20.

[17] I. O. Gallegos, R. A. Rossi, J. Barrow, M. M. Tanjim, S. Kim, F. Dernoncourt, T. Yu, R. Zhang, and N. K. Ahmed. Bias and fairness in large language models: A survey. *arXiv preprint arXiv:2309.00770v3*, 2024.

[18] Sai Gattupalli, William Lee, Danielle Allessio, Danielle Crabtree, Ivon Arroyo, and Beverly Woolf. Exploring pre-service teachers' perceptions of large language models-generated hints in online mathematics learning. *Proceedings of the 16th International Conference on Educational Data Mining (EDM 2023)*, 2023.

[19] Google Colab Pro+. Colab pro+ runtime specifications. `https://colab.research.google.com/`, 2025. Observed system resources during training session: 83.5 GB RAM, 235.7 GB disk, 40 GB GPU RAM (NVIDIA A100).

[20] Kaden Griffith and Jugal Kalita. Solving arithmetic word problems with transformers and preprocessing of problem text. In *Proceedings of the 17th International Conference on Natural Language Processing*, pages 76–84, 2020.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.

[22] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

[23] Neil Houlsby, Andrei Giurgiu, Stanisław Jastrzbski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. *arXiv preprint arXiv:1902.00751*, 2019.

[24] Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685v2*, 2021.

[25] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[26] Yan Hu, Qingyu Chen, Jingcheng Du, Xueqing Peng, Vipina Kuttichi Keloth, Xu Zuo, Yujia Zhou, Zehan Li, Xiaoqian Jiang, Zhiyong Lu, Kirk Roberts, and Hua Xu. Improving large language models for clinical named entity recognition via prompt engineering. *Journal of the American Medical Informatics Association*, 31(9):1812–1820, 01 2024.

[27] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186v3*, 2024. 12 Nov 2024.

[28] Vivek Kumar, Rishabh Maheshwary, and Vikram Pudi. Adversarial examples for evaluating math word problem solvers. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2705–2712. Association for Computational Linguistics, 2021.

[29] Surafel M. Lakew, Quintino F. Lotito, Matteo Negri, Marco Turchi, and Marcello Federico. Improving zero-shot translation of low-resource languages. *arXiv preprint arXiv:1811.01389*, 2018. Accessed: 2025-05-01.

[30] Shalom Lappin. Assessing the strengths and weaknesses of large language models. *Journal of Logic, Language and Information*, 33(1):9–20, 2024.

[31] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander M. Rush, and Thomas Wolf. Datasets: A community library for natural language processing, 2021.

[32] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.

[33] Nelson F Liu, Tushar Khot, Ashish Sabharwal, Doug Downey, and Matt Gardner. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

[34] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2103.10385*, 2021.

[35] MarkTechPost. Deepseek ai open-sourced deepseek-vl2 series: Three models of 3b, 16b, and 27b parameters with mixture of experts (moe) architecture redefining vision-language ai. `https://www.marktechpost.com/2024/12/15/deepseek-ai-open-sourced-deepseek-vl2-series-three-models-of-3b-16b-and-27b-para` 2024. Accessed: 2025-04-11.

[36] Marquette University. Applications of deep learning in mathematics, 2025. Accessed: 2025-04-20.

[37] Sean McLeish, Arpit Bansal, Alex Stein, et al. Transformers can do arithmetic with the right embeddings. *arXiv preprint arXiv:2405.17399*, 2024.

[38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[39] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

[40] Youssef Mroueh. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification, 2025.

[41] NVIDIA. Nvidia a100 tensor core gpu architecture: Unprecedented acceleration at every scale. Technical report, NVIDIA, 2020.

[42] Author(s) of mescal paper. The hermes calibration pipeline: mescal. *arXiv preprint arXiv:2402.02937*, 2024.

[43] OpenAI. What are tokens and how to count them? `https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them`, 2023. Visited on 12/21/2023.

[44] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[45] L. Heim et al. P. Villalobos, J. Sevilla. Will we run out of data? an analysis of the limits of scaling datasets in machine learning. *ArXiv*, abs/2211.04325(null), 2022. [Online]. Available: https://www.semanticscholar.org/paper/519b4ff448ee48ba7e08d146a29ade6f019a98d4.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[47] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[48] Ofir Press, Thomas Wolf, and Emily Dinan. Improving attention with qkv bias. *arXiv preprint arXiv:2102.11972*, 2021.

[49] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[50] N. Ryder, T. B. Brown, and B. Mann. Language models are few-shot learners. *arXiv*, 2020. Online.

[51] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[52] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, 2016. Association for Computational Linguistics.

[53] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv:2402.03300v3 [cs.CL]*, 2024. Accessed: 2024-04-27.

[54] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202v1*, 2020. Accessed: 2025-05-01.

[55] Safal Shrestha, Minwu Kim, and Keith Ross. Mathematical reasoning in large language models: Assessing logical and arithmetic errors across wide numerical ranges. *arXiv preprint arXiv:2502.08680*, 2025.

[56] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. *arXiv:2009.01325v3 [cs.CL]*, 2022.

[57] J. Su, Z. Li, H. Li, Z. Zhang, Z. Yao, F. Wu, and L. Xie. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[58] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[59] PyTorch Team. Torchscript: A static subset of pytorch for production. *PyTorch Documentation*, 2019. https://pytorch.org/docs/stable/jit.html.

[60] PyTorch Team. Libtorch: The c++ api for pytorch. *PyTorch Documentation*, 2020. https://pytorch.org/cppdocs/.

[61] Qwen Team. Qwen2.5 technical report. `https://arxiv.org/abs/2412.15115v2`, 2025. Accessed: 2025-01-03.

[62] QwenLM Team. Extending transformer context windows to 131k tokens. *arXiv preprint arXiv:2412.15115*, 2024.

[63] QwenLM Team. Qwen2.5: Architecture of a 7.61b parameter causal transformer. *arXiv preprint arXiv:2412.15115*, 2024.

[64] QwenLM Team. Untied embeddings for large language models. `https://huggingface.co/Qwen/Qwen2.5-7B`, 2024.

[65] Jayden Teoh and Others. Pufferlib: Making reinforcement learning libraries and benchmarks accessible. *arXiv preprint arXiv:2406.12905*, 2024.

[66] University of Washington Libraries. Digital content on mathematical modeling, 2025. Accessed: 2025-04-20.

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[68] Milan Vojnovic and Se-Young Yun. What is the alignment objective of grpo?, 2025.

[69] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay, 2017.

[70] Chenlu Ye Lichang Chen Nan Jiang Tong Zhang Wei Xiong, Hanning Zhang. Self-rewarding correction for mathematical reasoning. *arXiv preprint arXiv:2502.19613v1 [cs.AI]*, 2025. 26 Feb 2025.

[71] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[72] An-Zi Yen and Wei-Ling Hsu. Three questions concerning the use of large language models to facilitate mathematics learning. *CoRR*, abs/2310.13615, 2023.

[73] Feng Wu Yingfeng Chen Yixiang Wang, Yujing Hu. Automatic reward design via learning motivation-consistent intrinsic rewards. *arXiv preprint arXiv:2207.14722v1 [cs.LG]*, 2022. 29 Jul 2022.

[74] Mingqi Yuan, Bo Li, Xin Jin, and Wenjun Zeng. Automatic intrinsic reward shaping for exploration in deep reinforcement learning. *arXiv preprint arXiv:2301.10886*, 2023. The Fortieth International Conference on Machine Learning (ICML 2023).

[75] Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. *arXiv preprint arXiv:1905.12616*, 2020. arXiv:1905.12616v3 [cs.CL] 11 Dec 2020.

[76] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *arXiv preprint arXiv:1910.07467*, 2019.

[77] Hongkun Zhang, Ronghang Xiong, and Richard Socher. Root mean square layer normalization. *arXiv preprint arXiv:2009.02740*, 2020.

[78] Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention, 2024.

[79] Yi Zhu, Yikang Shen, Zhihua Zhang, et al. Iconqa: A new benchmark for abstract diagram understanding in real-world scenarios. `https://escholarship.org/uc/item/678864d8`, 2021. Accessed: April 20, 2025.