Mathematical Word Problem Solving

Swayam Agrawal
CSE
IIIT Hyderabad

Pratham Thakkar
CSE
IIIT Hyderabad

Yash Kawade
CSE
IIIT Hyderabad

Abstract

Mathematical problems are often stated in words in different scenarios, thus requiring problem solvers to extract information from the text and formulate in mathematical language to get the problem's answer. This project explores advanced Natural Language Processing (NLP) approaches to solve MWPs, comparing traditional models like RNNs and Transformers, math-specific architectures like Graph-to-Tree & GTS, and large language models (LLMs) such as Gemma and Mistral. We also evaluate the impact of reasoning paradigms like Chain-of-Thought (COT) prompting.

Our findings highlight trade-offs among these methods: while Graph-To-Tree excels in structured reasoning, LLMs demonstrate strong generalization, especially when fine-tuned or enhanced with Few-shot prompting or COT strategies.

This study underscores the potential of classical NLP approaches and LLMs for MWPs, emphasizing the need for fine-tuning and more complex reasoning strategies.

1 Introduction

Mathematical reasoning is an integral aspect of human intelligence, deeply embedded in everyday tasks like calculating expenses, measuring quantities, or solving complex problems. Consequently, solving Mathematical Word Problems (MWPs)—which require translating natural language into structured mathematical expressions—has been a long-standing challenge in Artificial Intelligence (AI) research. Early approaches predominantly relied on rule-based systems, which, while effective for narrowly defined problems, struggled with scalability and diversity.

The advent of machine learning in the 2010s shifted the focus to statistical methods and feature engineering for MWPs. More recently, deep learning has emerged as the dominant approach, driven by advances in computation and the availability of large-scale datasets. Despite this progress, MWPs remain challenging due to the inherent semantic gap between natural language and mathematical logic, compounded by the nuances of mathematical operators like the commutative laws of addition and multiplication.

In this work, we focus on advancing the performance of NLP models in solving MWPs. Our study evaluates traditional architectures like RNNs and Transformers, math-specific models like Graph-to-Tree & GTS, and cutting-edge large language models (LLMs) such as fine-tuned Gemma and Mistral. Additionally, we integrate reasoning techniques like Chain-of-Thought (COT) prompting to enhance model interpretability and consistency. We analyze the strengths and weaknesses of these methods to better understand how they perform on MWPs, with the goal of investigating the present mathematical reasoning skills of various NLP techniques.

2 Related Work

Solving Mathematical Word Problems (MWPs) has been an area of interest for decades & has garnered significant attention within the fields of Artificial Intelligence (AI) and Natural Language

Processing (NLP). Early approaches primarily utilized rule-based systems, which, while effective for specific problem types, lacked scalability and flexibility in handling diverse linguistic constructs. As machine learning gained traction in the 2010s, researchers began to explore statistical methods, including semantic parsing and feature engineering, to enhance MWP-solving capabilities. Notable contributions in this area include the work by Kushman et al. (2014), who developed equation template systems to extract mathematical relationships from text, and Ling et al. (2017), who employed latent variable models to capture underlying mathematical logic. Despite these advancements, the accuracy of these models remained limited, with state-of-the-art results achieving only around 36% on benchmark datasets.

The introduction of deep learning has revolutionized MWP solving methodologies. Cheng et al. investigated various deep learning architectures, including Recurrent Neural Networks (RNNs) and Transformers. Their findings indicated that Transformers outperformed traditional RNN models due to their ability to capture long-range dependencies and contextual information through self-attention mechanisms. This shift towards deep learning has also led to the development of specialized models like Graph-to-Tree (G2T) and Graph-to-Sequence (GTS), which leverage graph representations to model complex relationships within MWPs, thus enhancing performance on structured reasoning tasks.

In recent years, large language models (LLMs) have emerged as powerful tools.These models benefit significantly from fine-tuning on specific datasets and employing Few-shot prompting techniques. Additionally, reasoning paradigms like Chain-of-Thought (COT) prompting have shown promise in improving model interpretability and consistency by guiding models through their reasoning processes.

3 Dataset

The MAWPS (Math Word Problem Solver) dataset from Patel et al. (2021) is designed to help develop and evaluate natural language models capable of solving math word problems. Each entry in the dataset contains a word problem, the numerical values involved, the corresponding equation, and the correct solution.

The dataset contains the following fields:

- **Question**: The word problem expressed in natural language.
- **Numbers**: The numerical values extracted from the question.
- **Equation**: The mathematical equation representing the problem's solution.
- **Answer**: The correct solution to the equation.
- **Body**: The main context of the word problem (excluding the question).
- **Ques_Statement**: The main question posed in the word problem.

```
Question: Mary is baking a cake. The recipe wants number0 cups of flour. She alr
eady put in number1 cups. How many cups does she need to add?
Numbers: 8.0 2.0
Equation: - number0 number1
Answer: 6
Body: Mary is baking a cake . The recipe wants number0 cups of flour . She alrea
dy put in number1 cups.
Ques_Statement: How many cups does she need to add?
```

Figure 1: Dataset sample

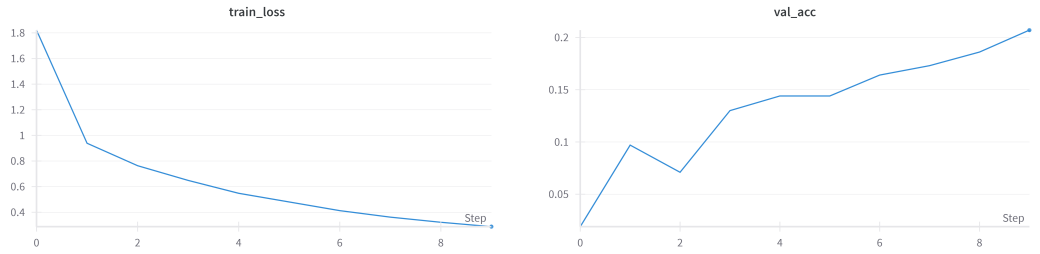
4 Approaches and Results

4.1 Baseline: Feed-Forward Network as Encoder & LSTM Decoder

We construct a simple model based on the Seq2Seq architecture by simply using a Feed-Forward Network that maps the input embeddings to their hidden representations. The LSTM Decoder is provided with the average of these hidden representations as its initial hidden state. During decoding, an attention mechanism (Luong et al., 2015) assigns weights to individual hidden representations of the input tokens.

1. **Input Embedding Transformation:**
$$h_i = \text{FFN}(e_i), \quad i = 1, 2, \ldots, n$$
2. **Hidden State Initialization:**
$$h_{\text{init}} = \frac{1}{n} \sum_{i=1}^{n} h_i$$
3. **Attention Mechanism:**
$$a_t^i = \text{softmax}(h_i^\top W_a s_t), \quad c_t = \sum_{i=1}^{n} a_t^i h_i$$
4. **Decoder Update:**
$$s_t = \text{LSTM}(x_t, [s_{t-1}, c_t])$$
5. **Prediction:**
$$P(y_t \mid y_{<t}, x) = \text{softmax}(W_o s_t + b_o)$$

This model when provided with non-contextual RoBERTa embeddings is able to achieve 20.7% accuracy on the ASDiv-A dataset. This is expected as well since the model does not have access to word-order information.



(a) Train Loss curve.

(b) Validation Accuracy curve.

Figure 2: Train Loss and Validation Accuracy curves.

4.2 Baseline: RNN as Encoder & LSTM Decoder

Our baseline model employs an encoder-decoder architecture enhanced with an attention mechanism. The encoder processes the input sequence—comprised of the MWP text—and generates hidden states that encapsulate contextual information. The decoder then utilizes these hidden states along with attention weights to focus on relevant parts of the input when generating the output mathematical expression.

1. **Hidden State Update:** For each time step t:
$$h_t = \sigma(W_h x_t + U_h h_{t-1} + b_h)$$
2. **Final Hidden States:** The final hidden states from the encoder for each input token are stored as:
$$H = [h_1, h_2, \ldots, h_T]$$

3. **Attention Context Vector:** The context vector a_t at decoding step t:

$$a_t = \sum_{i=1}^{m} \alpha_{t,i} h_i$$

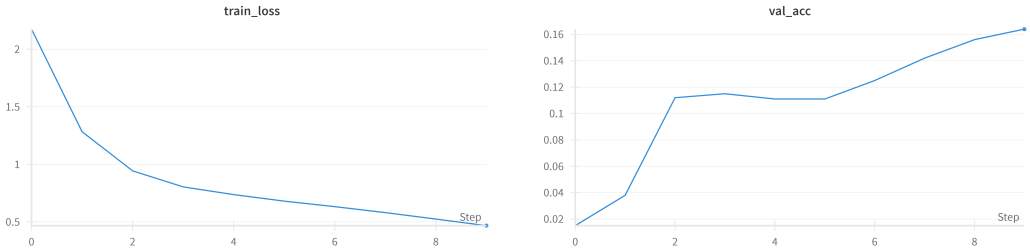
4. **Attention Weights Calculation:** The attention weights $\alpha_{t,i}$ are computed as:

$$\alpha_{t,i} = \text{Softmax}\left((h_{\text{dec},t})^T W_{\text{att}} h_i\right)$$

5. **Output Generation:** The output y_t at decoding step t:

$$y_t = \text{Softmax}\left(W_{\text{out}}[a_t; h_{\text{dec},t}]\right)$$

The attention mechanism allows the model to dynamically weigh different parts of the input sequence during decoding, which is crucial for accurately translating natural language into mathematical equations. We obtain an accuracy of 16.4% on the validation data of the ASDiv-A dataset.



(a) Train Loss curve.

(b) Validation Accuracy curve.

Figure 3: Train Loss and Validation Accuracy curves.

4.3 Baseline: Transformer

We construct a Seq2Seq model entirely based on the Transformer architecture from Vaswani et al. (2023). The encoder maps the input embeddings to contextualized hidden representations using self-attention layers, capturing relationships across all tokens in the input. The decoder generates the output sequence autoregressively, attending to both the encoder's representations and its previously generated tokens.

The Transformer Encoder uses multi-head self-attention and position-wise feed-forward layers to transform input embeddings. The Transformer Decoder, initialized with positional encodings, incorporates multi-head self-attention and cross-attention layers to process past outputs and attend to encoder outputs. The final prediction is made through a linear projection followed by a softmax function.

When provided with non-contextual RoBERTa embeddings, this model achieves 29.5% accuracy on the ASDiv-A dataset, outperforming LSTM-based methods due to its ability to handle long-range dependencies and fully parallelize computations.

1. **Input Embedding with Positional Encoding:** For each input token x_i in the sequence, the embedding is combined with its positional encoding:

$$z_i = \text{PE}(x_i) + e_i, \quad i = 1, 2, \ldots, n$$

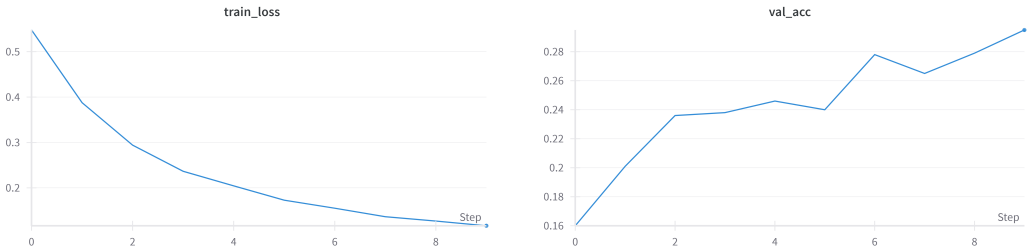
2. **Multi-Head Self-Attention:** The attention mechanism in each Transformer layer calculates:

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

Where Q, K, V are computed as:

$$Q = ZW_Q, \quad K = ZW_K, \quad V = ZW_V$$

3. **Transformer Encoder Layer:** The hidden state for each token after the encoder layer is updated as:
$$h_i = \text{LayerNorm}(\text{FFN}(\text{Attention}(Q, K, V) + z_i))$$
4. **Decoder Self-Attention:** For each decoding step t, the decoder self-attention is calculated as:
$$\text{Self-Attention}(Q_D, K_D, V_D) = \text{Softmax}\left(\frac{Q_D K_D^\top}{\sqrt{d_k}}\right) V_D$$
5. **Cross-Attention (Encoder-Decoder):** The decoder uses cross-attention to attend to the encoder's hidden states:
$$\text{Cross-Attention}(Q_D, K_E, V_E) = \text{Softmax}\left(\frac{Q_D K_E^\top}{\sqrt{d_k}}\right) V_E$$
6. **Decoder Layer Update:** The hidden state at decoding step t after self-attention and cross-attention is:
$$s_t = \text{LayerNorm}(\text{FFN}(\text{Self-Attention}(z_t) + \text{Cross-Attention}(H)))$$
7. **Output Prediction:** The probability distribution over the output vocabulary at decoding step t is:
$$P(y_t \mid y_{<t}, x) = \text{Softmax}(W_{\text{out}} s_t + b_{\text{out}})$$



(a) Train Loss curve.

(b) Validation Accuracy curve.

Figure 4: Train Loss and Validation Accuracy curves.

4.4 Goal-Driven Tree-Structured (GTS)

The Goal-Driven Tree-Structured (GTS) model from Xie and Sun (2019) is a novel approach to solving math word problems (MWPs) that generates an expression tree in a human-like, goal-driven manner. Unlike sequence-to-sequence models like RNNs and Transformers, which generate solution expressions from left to right, the GTS model decomposes the problem into a hierarchy of subgoals, mimicking how humans solve problems.

1. **Problem Understanding**: Identifies the final goal of the Math Word Problem (MWP), e.g., calculating the number of baggies from given cookie counts.

2. **Goal Decomposition**: Breaks the goal into recursive sub-goals linked by operators, e.g., "total cookies ÷ cookies per bag", with further decomposition as needed.

3. **Expression Tree Generation**: Builds an expression tree where nodes represent goals/sub-goals, leaves represent known quantities, and internal nodes are operators.

4. **Subtree Embedding**: Uses a recursive neural network to encode subtrees, capturing mathematical meaning for richer sub-goal representation.

5. **Contextual Information**: Leverages attention to extract problem-relevant details for predicting tokens and guiding decomposition.

The GTS model solves problems intuitively by breaking them into smaller steps, creating clear and logical tree structures that connect numbers and operations. It achieves an accuracy of 29.6% after 10 epochs, ensuring valid mathematical expressions while focusing only on relevant numbers and avoiding irrelevant or incorrect predictions.

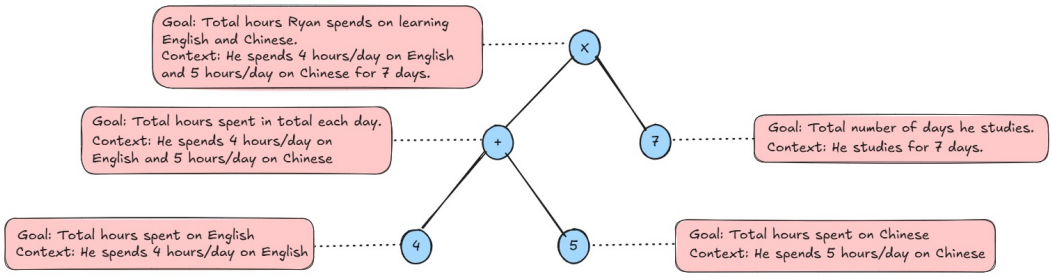
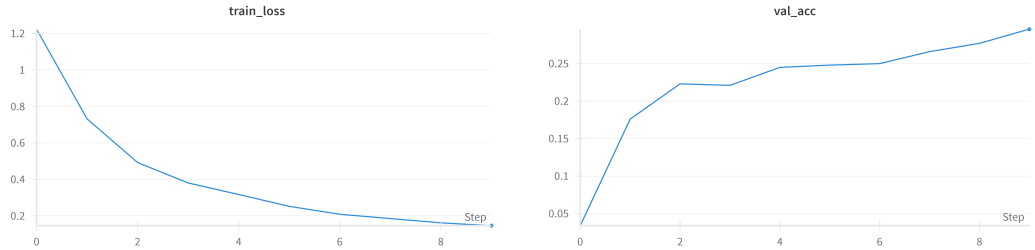


Figure 5: The expression tree for a sample problem.



(a) Train Loss curve.

(b) Validation Accuracy curve.

Figure 6: Train Loss and Validation Accuracy curves.

4.5 Graph to Tree learning

The Graph2Tree model from Zhang et al. (2020) is an extension of the GTS approach that improves the representation of Math Word Problems (MWPs) through graph-based encoding. While GTS effectively decodes expressions in a tree-structured manner, its performance can be further enhanced by enriching the representation of the problem itself. Graph2Tree addresses this by incorporating two novel graph structures that capture both the semantic and numerical relationships between quantities in MWPs.

- **Quantity Cell Graph**: This graph links each quantity to its associated nouns, adjectives, verbs, units, and rates. These relationships are extracted using dependency parsing, constituency parsing, and part-of-speech tagging. This graph helps the model understand the meaning of quantities within the context of the problem. For instance, in a problem involving "15 aquariums for saltwater animals," the Quantity Cell Graph would link the quantity "15" to the noun "aquariums" and the context "saltwater animals."
- **Quantity Comparison Graph**: This graph encodes the numerical relationships between quantities by adding directed edges between them based on their magnitudes. If one quantity is larger than another, a directed edge points from the larger to the smaller quantity. This heuristic helps prevent the model from generating expressions that subtract a larger number from a smaller one, leading to negative results, which are often unrealistic in the context of MWPs.

These two graph structures are then used as input to a graph transformer encoder, which learns a rich representation of the MWP that captures both semantic and numerical relationships between quantities. This representation is then fed into the GTS decoder, which generates the solution expression tree.

By enriching the representation of the problem with these two graphs, Graph2Tree provides the GTS decoder with more informative input, allowing it to generate more accurate and realistic solution expressions, achieving an accuracy of 36.1% after 10 epochs.

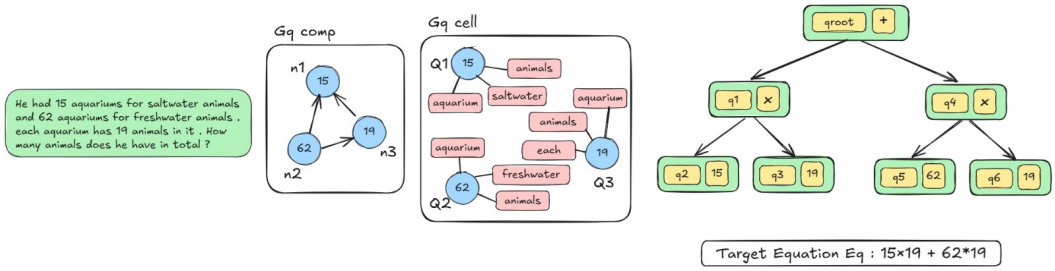
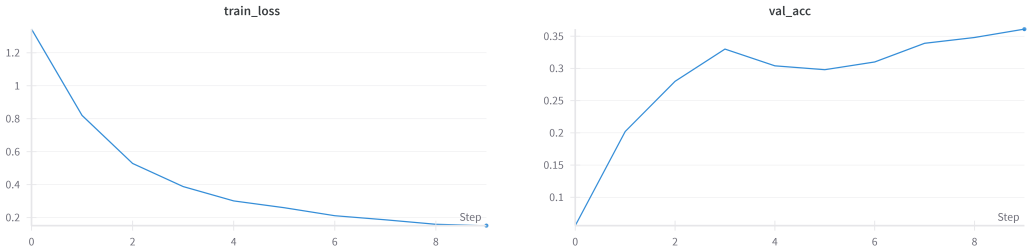


Figure 7: Overview of the Graph2Tree model.



(a) Train Loss curve.

(b) Validation Accuracy curve.

Figure 8: Train Loss and Validation Accuracy curves.

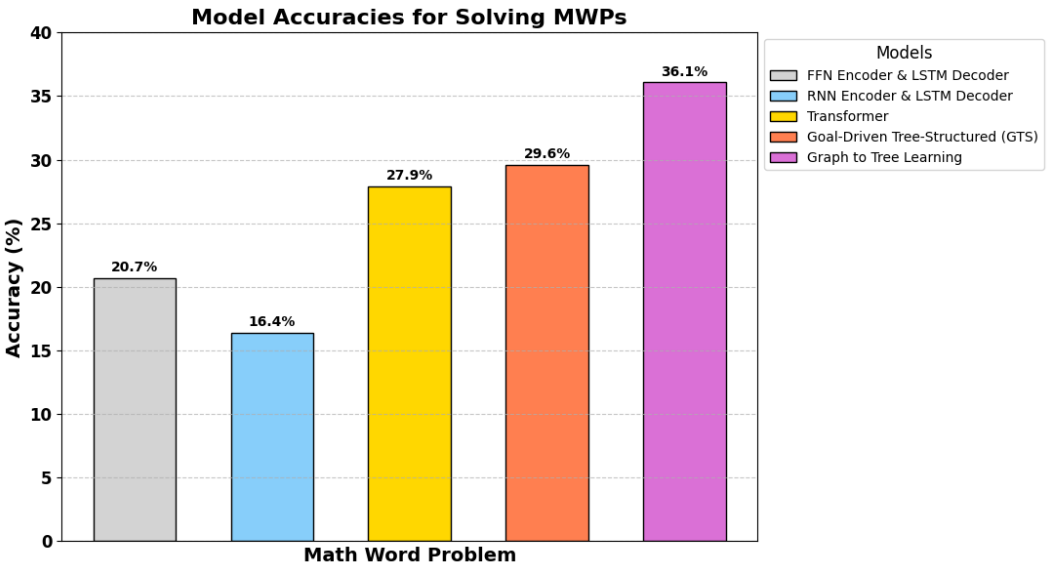


Figure 9: Comparison of Baseline Models.

4.6 LLM: Gemma-2-9B

4.6.1 Direct Evaluation of Gemma-2-9B

The direct evaluation of the pre-trained Gemma-2-9B model Team and et al. (2024) was conducted on the MAWPS-ASDIV dataset without any fine-tuning. This evaluation aimed to understand the model's baseline performance in solving mathematical word problems.

Evaluation Setup: The model was used in its pre-trained form with no additional training or optimization specific to the MAWPS-ASDIV dataset. During inference, the model was provided with problem statements in natural language and was tasked with predicting the numerical answer directly. The generated responses were compared against the ground truth answers for accuracy evaluation.

Prompting Strategy: To guide the model's response format, the following prompt was used for each problem:

```
Solve the following math problem. Provide only the numerical
answer in the format: "the answer is [number]" without any
explanation.
```

This ensured consistent output formatting, which is critical for evaluating model accuracy.

Results: The direct Gemma-2-9B model achieved an accuracy of approximately **64%** on the dataset. This baseline performance reflects the model's inherent capabilities derived from pre-training but also highlights the challenges of solving mathematical word problems without domain-specific fine-tuning.

Observations:

- The model performed well on problems involving basic arithmetic operations and straightforward question phrasing.
- Performance degraded on problems requiring multi-step reasoning or involving implicit mathematical relationships.
- Output consistency was maintained due to the strict prompting strategy, but some numerical errors were observed in cases of complex reasoning.

4.6.2 Fine-Tuning of Gemma-2-9B using Standard Prompting

To improve the baseline performance of Gemma-2-9B on the MAWPS-ASDIV dataset, we employed fine-tuning with domain-specific training examples. The fine-tuning process involved using the standard MAWPS-ASDIV problems and answers to adapt the model for mathematical word problem-solving.

Fine-Tuning Setup: The fine-tuning process was implemented using the **UnSloth** framework, a lightweight and efficient library for scalable fine-tuning of large language models. This framework significantly reduced computational costs and wall-clock time by optimizing gradient computations and minimizing redundancy.

Evaluation Post Fine-Tuning: After fine-tuning, the model was evaluated on the MAWPS-ASDIV dataset. The results showed a significant improvement in accuracy compared to the direct Gemma evaluation. The fine-tuned model achieved an accuracy of **67%**, outperforming the baseline by nearly 3%.

Observations:

- Fine-tuning enhanced the model's ability to interpret complex question phrasing and solve multi-step problems.
- The use of UnSloth accelerated the training process, allowing effective fine-tuning with minimal resource overhead.
- The model demonstrated improved numerical precision and fewer systematic errors.

4.6.3 Few-Shot Prompting Fine-Tuning

Few-shot prompting fine-tuning inspired by Brown et al. (2020) was applied to the Gemma-2-9B model to further enhance its performance on the MAWPS-ASDIV dataset. This method combined the benefits of providing task-specific examples during inference with standard fine-tuning techniques, enabling the model to generalize better on mathematical word problems.

Fine-Tuning Strategy: Few-shot prompting involved including several task-specific examples within the prompt to guide the model during training and inference. These examples were presented in the following format:

```
Here are a few examples for your reference:
```

- `Question: Tom has $100. He spends 40% of the money on books. How much money does he have left now? Answer: the answer is 60.0`

This strategy not only provided contextual guidance to the model but also improved its understanding of task-specific nuances.

Evaluation Results: The fine-tuned model was evaluated on 50 samples from the MAWPS-ASDIV dataset due to lack of efficient compute. The results indicated an accuracy of **88%**, reflecting a significant improvement over previous methods, including direct evaluation and standard fine-tuning.

Observations:

- Few-shot prompting fine-tuning significantly enhanced the model's ability to solve complex multi-step problems.
- Accuracy improved due to better contextual understanding facilitated by the inclusion of task-specific examples.
- Some errors persisted in cases requiring implicit reasoning or multi-step logical deductions, though the error rate was markedly reduced compared to previous methods.

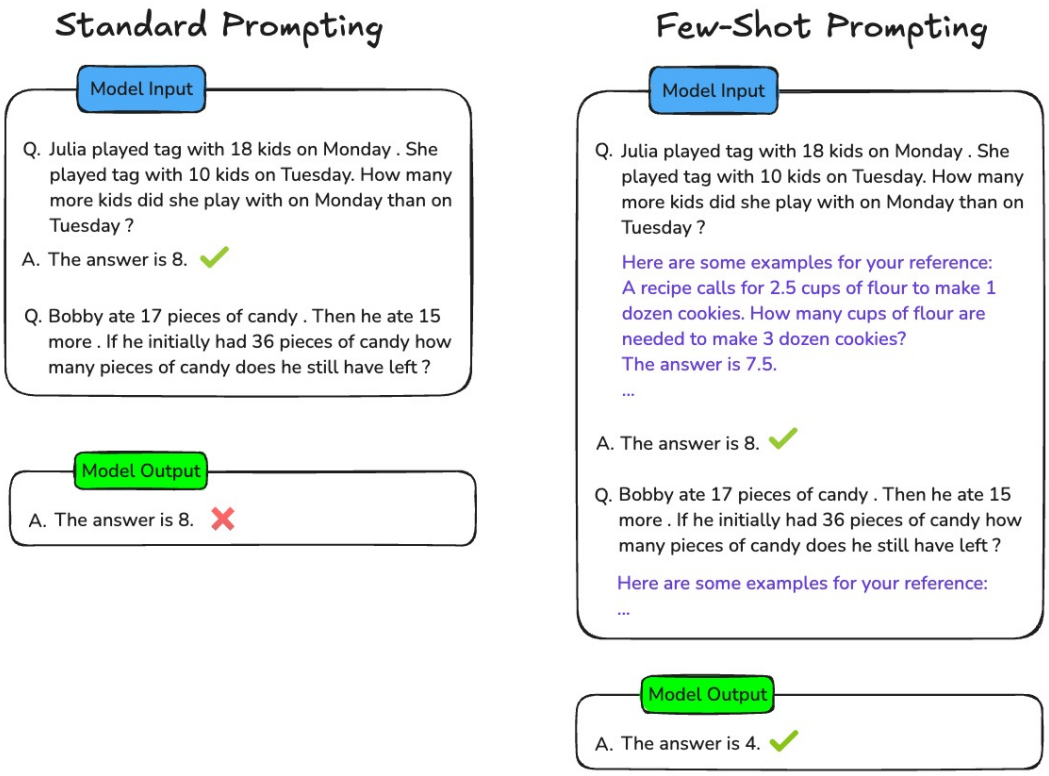


Figure 10: Comparison of Standard Prompting and Few-Shot Prompting results.

4.6.4 Inference-Side Chain-of-Thought (CoT) Prompting

Chain-of-Thought (CoT) prompting was applied to the Gemma-2-9B model during inference to improve its reasoning capabilities when solving mathematical word problems. CoT prompting encourages the model to generate step-by-step reasoning before arriving at the final answer, using intermediate reasoning paths to enhance accuracy and consistency. This approach is inspired by the work in Wei et al. (2023), where CoT prompting was introduced as a method to boost large language models' performance on reasoning tasks.

Prompting Strategy: To implement CoT prompting, the model was provided with the instruction to "think step-by-step" before producing the final numerical answer. This strategy guided the model to articulate intermediate reasoning steps, leading to improved accuracy in problems requiring multi-step calculations or logical deductions.

Evaluation Results: The inference-side CoT prompting was evaluated on the MAWPS-ASDIV dataset. The results demonstrated a significant improvement over direct evaluation of the model, achieving an accuracy of **94%** on the test set of 50 samples. This performance highlights the effectiveness of CoT prompting in enhancing the model's problem-solving capabilities without requiring additional fine-tuning.

Observations:

- The CoT approach enabled the model to handle complex reasoning tasks, such as multi-step arithmetic or problems involving implicit logical deductions, with greater accuracy.
- While the reasoning process occasionally introduced unnecessary verbosity, the final numerical answers were more accurate due to the structured step-by-step reasoning.
- Errors were still observed in problems which required more complex reasoning and thinking capabilities.

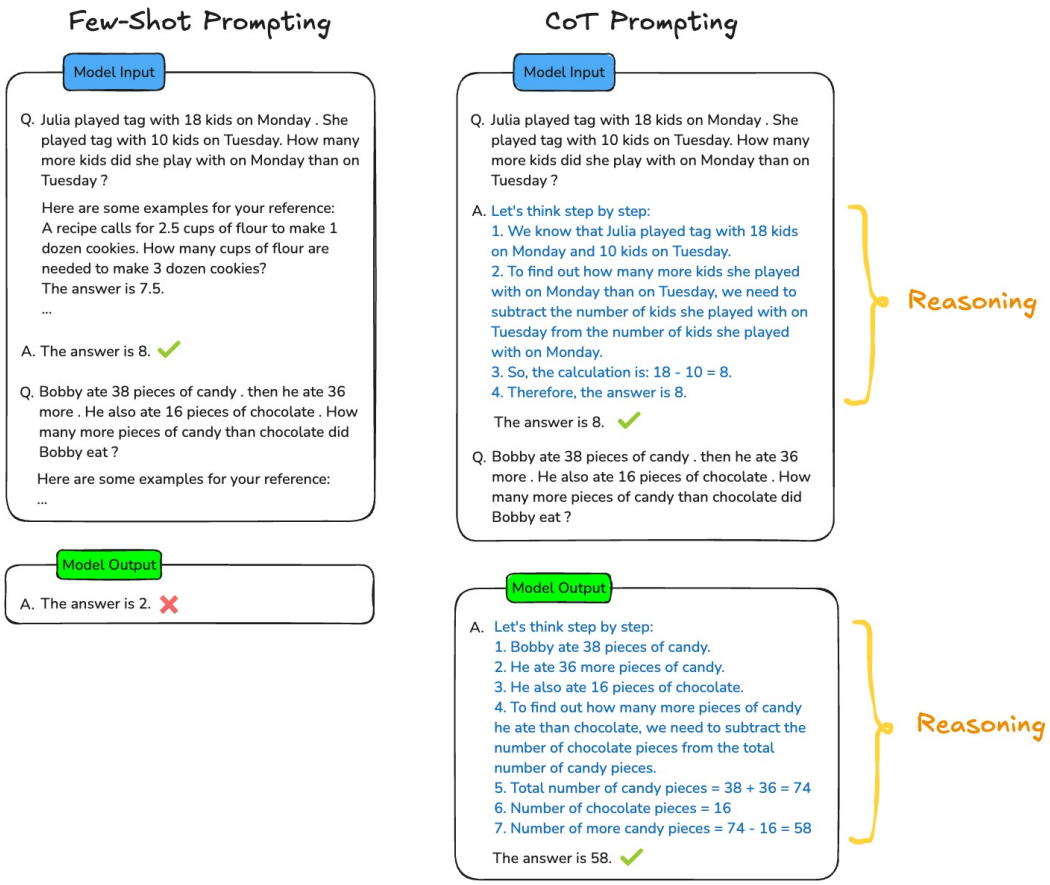


Figure 11: Comparison of Few-Shot Prompting and CoT Reasoning results.

4.6.5 Chain-of-Thought (CoT) + Few-Shot Prompting

Chain-of-Thought (CoT) reasoning combined with few-shot prompting was used during inference to further enhance the Gemma-2-9B model's performance on the MAWPS-ASDIV dataset. This approach leveraged the strengths of step-by-step reasoning and contextual guidance provided by task-specific examples, enabling the model to solve complex mathematical word problems with higher accuracy and consistency.

Prompting Strategy: The CoT + Few-Shot prompting method utilized a multi-part prompt, beginning with explicit instructions to "think step-by-step" for reasoning through each problem. The prompt also included several few-shot examples of solved problems, formatted to demonstrate the step-by-step reasoning process and the expected concise numerical answer.

Evaluation Results: This combined strategy was evaluated on the MAWPS-ASDIV dataset, achieving an accuracy of **96%**. This shows an improvement over both standalone CoT and few-shot prompting techniques, highlighting the positive effect of combining these approaches.

Observations:

- The combined approach improved the model's ability to handle ambiguous or multi-step problems that previously led to errors in reasoning or computation.
- Few-shot examples in the prompt provided additional clarity on task requirements, reducing the frequency of systematic errors observed in earlier methods.
- Despite occasional verbosity in reasoning steps, the model's output consistently adhered to the expected format, ensuring reliable evaluation of accuracy.

- Since the reasoning provided in the few-shot examples were concise and to-the-point, we could observe the same during inference on unseen examples as well in comparison to standalone CoT approach.

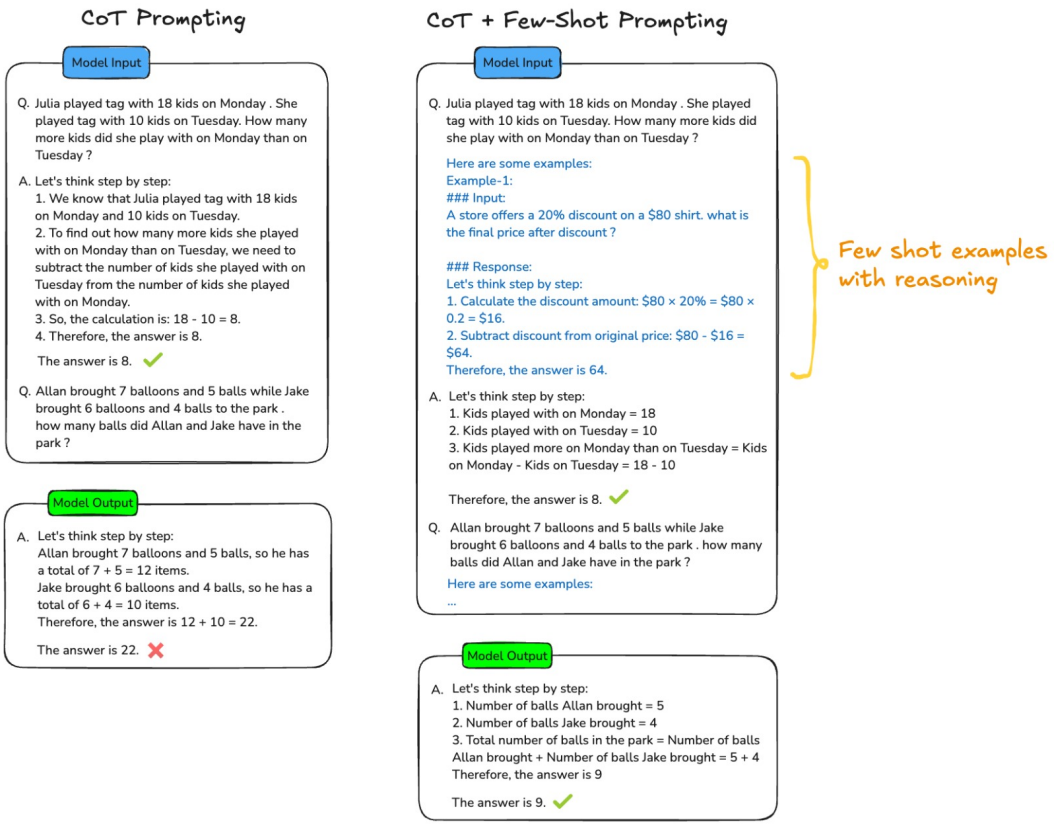


Figure 12: Comparison of standalone Inference side CoT reasoning and combined CoT reasoning + Few Shot prompting results.

4.7 LLM: Mistral-Instruct-7B

4.7.1 Direct Evaluation of Mistral-Instruct-7B

Similar to the evaluation conducted with the Gemma-2-9B model, we evaluated the pre-trained Mistral-Instruct-7B model on the MAWPS-ASDIV dataset without any fine-tuning. The *evaluation setup*, *prompting strategy*, and *methodology* were identical, with the model tasked to provide numerical answers to mathematical word problems.

Results: The Mistral-Instruct-7B model achieved an accuracy of approximately 40%, demonstrating relatively worse baseline performance compared to Gemma-2-9B.

Observations:

- Gemma-2-9B is better at handling of basic arithmetic operations and straightforward problems compared to Mistral-Instruct-7B.
- But inferencing was much faster for Mistral-Instruct-7B as compared to Gemma-2-9B
- Similar performance drop on multi-step reasoning and implicit relationships.
- Consistent output formatting was maintained, but numerical errors persisted in complex cases.

4.7.2 Fine-Tuning of Mistral-Instruct-7B using Fine Tuning

To improve the baseline performance of Mistral-Instruct-7B on the MAWPS-ASDIV dataset, we employed fine-tuning with domain-specific training examples. The fine-tuning process involved using the standard MAWPS-ASDIV problems and answers to adapt the model for mathematical word problem-solving.

Fine-Tuning Setup: To further optimize the Mistral-Instruct-7B model for solving mathematical word problems, we applied NF4 quantization and performed LoRA-based fine-tuning on the MAWPS-ASDIV dataset. This approach aimed to reduce the model's memory footprint while enhancing its domain-specific performance.

Evaluation Post Fine-Tuning: After fine-tuning, the model was evaluated on the MAWPS-ASDIV dataset. The results showed a significant improvement in accuracy compared to the direct Mistral-Instruct-7B evaluation. The fine-tuned model achieved an accuracy of **56%**, outperforming the baseline by nearly 15%.

Observations:

- Fine-tuning enhanced the model's ability to interpret complex question phrasing and solve multi-step problems.
- The model demonstrated improved numerical precision and fewer systematic errors & much faster inferencing as compared to Gemma.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, Baltimore, Maryland. Association for Computational Linguistics.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Gemma Team and et al. 2024. Gemma 2: Improving open language models at a practical size.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *International Joint Conference on Artificial Intelligence*.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020. Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3928–3937, Online. Association for Computational Linguistics.