Automatically Identifying Errors in Primary Level Math Word Problems Generated by Large Language Models

Zhuonan Mai

A research report submitted to

School of Mathematical and Computational Sciences

in partial fulfillment of the requirements for the degree of

MinfSc

School of Mathematical and Computational Sciences

Massey University

159898 Research Report

Supervisor: Dr Surangika Ranathunga

July 2025

Abstract

Ensuring the quality of mathematical word problems (MWPs) is essential for primary education. However, large language models (LLMs) struggle with error identification despite excelling in problem-solving. This research evaluates four LLMs - Mixtral-8x7B-Instruct-v0.1(Mixtral-8x7B), Meta-Llama-3.1-8B-Instruct (Llama-3.1-8B), DeepSeek-Math-7B-Instruct (DeepSeek-Math-7B), and Llama-3.2-3B-Instruct （Llama-3.2-3B）, for detecting errors in a dataset that was generated by LLMs. This dataset contains 5,098 MWPs from U.S. grades 1–6. A comprehensive framework with 12 error categories is introduced, which goes beyond most categorization schemes used in prior research. By evaluating Zero-Shot (inference without any examples), One-Shot (inference with one example), and Three-Shot (inference with three examples) approaches, as well as fine-tuning, across four models in seven experiments, we found that small-scale model Llama-3.2-3B achieved the finest Zero-Shot accuracy of 90% with minimal resources of 6GB GPU memory, comparable to the larger model Mixtral-8x7B's fine-tuned accuracy rate of 90.62%. However, due to data noise and prompt complexity, fine-tuning yielded negative results, with an average accuracy of 78.48%. The complexity of the prompts reduced accuracy by up to 20% for the Mixtral-8x7B model. Safety biases, particularly in Llama-3.1-8B and Mixtral-8x7B, led to misclassifications when triggering safety words. Our findings highlight the efficacy of small-scale LLMs and concise prompts for educational applications while identifying challenges in fine-tuning and model bias. We propose future research directions that include noise-robust data preprocessing, refined prompt engineering, and adversarial fine-tuning. These approaches aim to enhance the reliability of LLMs in detecting errors in MWPs, thereby ensuring the validity of educational assessments and ultimately contributing to the advancement of high-quality foundational mathematics education.

Key Words: Math Word Problems, Fine-Tuning, Large Language Models, LLMs

Acknowledgement

I would like to express my gratitude to my supervisor, Dr. Surangika Ranathunga, for the academic guidance and support throughout the course of this research, which played a pivotal role in successfully completing the research.

Table of Contents

List of Figures

List of Tables

1. Introduction

The quality of mathematical word problems (MWPs) plays a significant role in elementary education, directly influencing the students' effective learning and long-term academic success. However, ensuring the accuracy of these materials is a tremendous challenge, especially in this advancing technology era – an increasing number of large language models (LLMs) are used to generate MWPs as creating MWPs is highly time-consuming. However, LLMs can make mistakes in the questions they generate. This would lead to MWP quality being uneven and cause concern [1], [2]. The lower quality of MWPs could pose potential risks, especially in grades 1-6 where foundational numeracy skills are being developed during this period. According to the NAEP 2022 Mathematics Report Card [3] and the National Center for Education Statistics [4], students' math performance declined from 2019 to 2022, grade 4 dropped by 5 points, and grade 8 dropped by 8 points as well. Errors in MWPs hinder the learners' understanding, confuse their conceptions of math, and lead to unexpected performance and critical math problem-solving [5], [6], thereby affecting their academic success. To mitigate these negative effects, it is crucial that automated assessment tools adhere to established principles of educational validity [7]. Evaluation should be comprehensive, fair, and actionable [8]. Guided by these principles, our study designs an error categorization and evaluation framework that aligns with educational goals and measures the suitability of MWPs for learning.

Nowadays, the method of mathematical problem-solving has changed dramatically due to advanced LLMs such as GPT-4 [9]. The model achieved a remarkable result of up to 92% on the benchmark dataset GSM8K [10], showing great potential in the education field. However, their abilities in quality assessment are not equal to their reasoning techniques. For GPT-4o [11], the error detection accuracy rate is only 50% in the GSM8K dataset test [12]. This trend indicates a weakness in artificial intelligence (AI) in current education: while excelling in reasoning and problem-solving, LLMs are struggling in error detection. This problem is serious in early education, where MWPs must be accurate to ensure the quality of education. The "detection gap" has been widened due to the significant limitations in existing research.

First, the current studies emphasise higher-level mathematics. As noted in [13], most current research is focused on secondary levels, and the proportion of primary education research is only 27%, which is rather low. Primary-level MWPs highlight the need for appropriate foundational skills and the age period, which are often ignored by current

analyses [14]. Second, existing error identification tools are narrow in categories — often focus on typical error types such as spelling mistakes, grammar errors, or computational inaccuracies. This narrow focus overlooks a wider range of error categories including topic safety, triviality, and grade alignment, which is essential for entry-level education. The lack of standardized comparative categories together with evaluation metrics hinders the comparison ability across models and tasks, limiting the progress of robust benchmark development. Third, there is a lack of a standard dataset, particularly for error identification. Most of the research utilizes their distinct dataset [15], which makes the experiment results less comparable. Moreover, comparative analyses across model architectures such as the Mixture of Experts (MoE) like Mixtral-8x7B [16] and the dense models like DeepSeek-Math-7B [17], as well as different generations of the same model, are still underexplored. These research gaps yield the urgent need for a comprehensive study that evaluates fine-tuned LLMs across diverse architectures and error categorizations, particularly for primary-level mathematics.

This study attempts to bridge the gaps through two contributions. First, it explores a well-designed and suitable prompt template for primary-level MWP error detection. Various prompt designs are employed in this research, including applying different prompting approaches within the same model and utilizing the same prompting strategy across different model types, to evaluate model capabilities and identify the most effective prompting designs. Second, it presents a systematic comparison of various model architectures between MoE and dense models (e.g., Mixtral-8x7B and the Llama series), providing insights into the relationship between model scale and error identification performance. These findings offer guidance for model selection in entry-level educational contexts, addressing a gap highlighted in the current literature. Moreover, these insights not only enhance error detection in entry-level mathematics but also lay a solid foundation for optimizing LLM deployment in real-world educational settings.

The next part of this thesis will review the literature and related work. Then, the methodology section will outline the detailed methods used in this study. This will be followed by the results and analysis, which evaluate the performance of Zero-Shot, few-shot, and fine-tuned experiments. Next, the discussion section will present the key findings in relation to practice, theory, and future research. Finally, the conclusion and recommendations will highlight the main insights and directions for future exploration.

2. Related Work

The application of AI in mathematics education has developed dramatically in the past several decades, transitioning from a rule-based system to advanced LLMs. This section will review prior work in six main domains: AI in mathematics education, error detection in math word problems, fine-tuning strategies for LLMs, safety considerations in educational AI, prompting strategies for LLM evaluation, and efficient fine-tuning and empirical evaluation strategies. Finally, it identifies the research gaps in this research.

2.1 AI for Math Education

AI began its journey in mathematics education in the 1960s by offering computer-assisted instruction (CAI) systems such as Programmed Logic for Automatic Teaching Operations (PLATO) [18]. These systems relied on predefined rules, which are unable to process natural language or adapt to various student inputs. Natural Language Processing (NLP) was introduced in the 1990s, enabling automated scoring for structured responses, such as multiple-choice questions [19]. These early NLP systems relied on feature engineering to assess correctness. However, they struggled with unstructured content. The 2010s brought a vital turning point in AI with the emergence of deep learning, which fundamentally reshaped methods in mathematical problem-solving and educational applications. Neural network-based architecture, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), brought significant progress in math word problem-solving and tutoring systems [20], [21]. These systems improved mathematical text parsing and feedback generation, although they struggled with limited contextual understanding. In 2017, the introduction of Transformer architecture transformed the field by using self-attention to model long-range dependencies, dramatically improving sequence learning [22].

The release of Bert in 2019 introduced dual-directional semantic awareness, significantly improved semantic analysis for tasks such as equation extraction and answer validation in advanced math problems [23]. Later developments, such as LLMs like GPT-3 and its later generations, further enhanced contextual reasoning and generalization, improving performance on a wide range of mathematical tasks [24]. These developments highlight the importance of architectural design in enhancing LLM performance for educational applications.

The 2020s marked a groundbreaking moment for LLMs. Certain models such as GPT-3, GPT-4 and Llama series, achieved excellent performance on the bentchmark GSM8K and MathQA [25]. MathGPT [26] and MathInstruct [27], the domain-specific models, have further improved mathematical reasoning techniques. MathGPT approaches human-level algebraic performance and MathInstruct excels in both problem-solving and instructional evaluation [28].

Although the domain keeps developing, most studies continue to focus on emphasizing accuracy and problem evaluation, often overlooking educational quality assessment [29], [30], for instance, error detection. In response to this trend, the present study seeks to bridge these gaps by examining the quality assessment of primary-level math word problems, identifying 12 key error categories, including trivial content, unit inconsistency, and topic safety issues—that are frequently neglected in existing research. This targeted approach not only enhances the reliability of large language models in educational contexts but also paves the way for building more efficient and context-sensitive assessment systems.

2.2 Error Identification in MWPs

Despite some progress in error detection within math word problems in prior research, this area remains underexplored, especially since most prior work focuses on narrow error types and targets advanced educational levels [31]. Mathematical grammatical error correction (GEC) systems, such as Bert model fine-tuned in [32], tackled grammatical and expression-related issues in math texts at the secondary level. Although it achieved high precision, these systems do not pay sufficient attention to non-linguistic errors such as unit inconsistencies. Some approaches focus solely on numerical accuracy while overlooking structural flaws in problem statements [33]. In [34], physics-related math problems were introduced to verify concept units. However, it is not suitable for primary level. Existing research heavily focuses on secondary advanced education, which means lower grade levels like grades under six are relatively blank. The benchmark datasets MAWPS and AQUA-RAT [35] emphasised answer accuracy rather than problem quality, lacking detailed error labels for issues such as Co-reference and grade-level alignment. A relatively comprehensive error category was proposed in [36], which included nine types of errors including calculation errors, logical errors, spelling errors and so on. Nevertheless, the classification of error type remains insufficient. This narrow focus not only limits the technical scope but also undermines the construct validity of the assessment. A valid educational assessment of

MWPs must account for more than just factual correctness; it must evaluate pedagogical appropriateness (e.g., grade-level alignment), contextual realism, and safety, which are crucial for effective learning but are consistently neglected in current literature.

To address these limitations, we adopt a multi-label evaluation framework. Traditional single-label classification is insufficient for this task, as MWPs can simultaneously contain multiple, co-occurring errors [37]. This framework is an effective approach for complex classification tasks where instances can belong to multiple categories simultaneously [38], allows for a more nuanced and accurate assessment of model performance across all error dimensions. Within this framework, we introduced a comprehensive error categorization scheme covering 12 error types: Co-reference issues, Grammatical errors, Misspellings, Incomplete sentences, Unsolvable problems, Unrealistic, Unit issues, Topic safety, Not satisfying grade requirement, Not satisfying section requirement, Trivial, and Not a word problem [39]-[42].

2.3 LLMs for Error Identification

LLMs are highly developed in mathematical problem-solving, yet their ability in error detection is still insufficiently explored [43]. GPT-4 achieves a 92% accuracy on the GSM8K benchmark for problem-solving [44]. Yet, its accuracy drops significantly on error detection tasks, revealing limitations in the model's ability to identify errors. Domain-specific models such as MathBERT, trained on mathematical datasets, improve equation parsing [45], but they need to be retrained to adapt to specific error detection tasks. Dense models such as Llama 3 excel in contextual tasks like topic safety due to their consistent computational approach [46]. However, their fine-tuning occasionally leads to overfitting, resulting in a decline in fine-tuned performance compared to Zero-Shot baselines. These limitations emphasis the need for robust models and unified error detection frameworks, particularly in educational contexts.

2.4 Safety in Educational AI

Safety is a crucial factor in AI educational applications, especially in primary-level education content. Therefore, most models apply a strict safety policy and control with model deployment. If the safety mechanism is not handled properly, it could potentially cause safety issues, such as killing and murder issues. Due to the impact of safety mechanisms, models like Reinforcement learning with human feedback (RLHF) often

perform over-cautious [47], [48]. LLama-3.1-70B has shown oversensitivity in experiments [49], [50]. This over-cautious phenomenon was further confirmed in later studies [47]. And a new High-Confidence Safe Reinforcement Learning from Human Feedback (HC-RLHF) framework was proposed to balance safety bias and practicality [51]. General safety frameworks, OpenAI's moderation API, have limitations in handling safety contents and they often judge incorrectly or fail in tasks [52].

2.5 Prompting Strategies for LLM Evaluation

Zero-Shot prompting is a technique that leverages the inherent knowledge and reasoning capabilities of LLMs to perform tasks without any task-specific examples provided in the prompt [53]. This approach relies on the models' extensive pre-training on diverse datasets, which allows them to generalize to a broad range of unseen tasks based solely on their understanding of the instruction [54]. The primary advantage of Zero-Shot prompting is its simplicity and efficiency, as it requires no handcrafted examples, making it highly applicable in scenarios where training data is limited or rapid prototyping is essential [55].

Few-shot prompting is an efficient technique that guides LLMs to perform tasks by providing a few examples (typically 3-5), which are embedded in the prompts [24]. This technique is computationally efficient. Because it does not require fine-tuning while still maintaining reliable results [56], [57]. It was proved that few-shot prompting could improve accuracy by 20-40% on complicated tasks compared to the Zero-Shot approach [57]. However, the accuracy results of few-shot prompting rely on the quality of the provided samples. Model performance would degrade if sample quality were poor [58].

2.6 Efficient Fine-Tuning and Empirical Evaluation Strategies

Beyond prompting, fine-tuning is an effective method for adapting LLMs to specific tasks. Due to the high cost of full-parameter tuning, resource-efficient methods such as Low-Rank Adaptation (LoRA) [59] and quantization [60] have become widely used. LoRA reduces the number of trainable parameters and memory needs, enabling large models to be tuned on limited hardware [61]. Combined with quantization, these methods make advanced model customization more accessible, which is especially useful in educational settings with constrained resources.

Furthermore, rigorous empirical evaluation of LLMs requires more than reporting final accuracy. A robust methodology involves systematic comparative experiments that control variables across model architectures, sizes, and training techniques to draw fair conclusions [62]. This often includes hyperparameter sensitivity analysis to ensure observed performance is robust and not an artifact of a specific configuration [63]. While common in machine learning, such rigorous ablation studies are sometimes overlooked in applied NLP research. Our study integrates these methodological best practices—employing efficient fine-tuning (LoRA) and conducting thorough comparative and sensitivity analyses—to ensure our findings on error detection are both practically viable and scientifically robust.

2.7 Summary of Gaps

According to the previous studies, there are two major limitations which our study aims to address. First, there is a lack of a unified efficient prompting template for primary-level MWP detection, particularly in comprehensive error categorization. Second, most existing studies rarely explore MoE and dense model architectures or compare different generations of the same model type, where our study aims to fill these gaps. By integrating a comprehensive error identification system with well-designed prompting, suitable model architecture and efficient fine-tuning arguments, our study aims to set a new standard for quality assessment in foundational mathematics education.

3. Methodology

This section describes the methodological framework employed to evaluate LLMs for error identification in primary-level MWPs. Our approach is designing a comprehensive empirical study, focusing on comparative experiments that evaluate different models and prompt strategies on the same task using a multi-label framework. To ensure practicality and robustness, we incorporate resource-efficient fine-tuning techniques (LoRA and 4-bit quantization) and conduct hyperparameter sensitivity analysis. The sections below cover the design of error classification, data preprocessing, comparative prompting strategies, model selection, efficient training methodology, and evaluation metrics.

3.1 MWPs Dataset Overview

The dataset, generated by the authors of [64], originally stored in the Combined MWP sheet.xlsx file, contains 5,098 MWPs ranging from U.S. grades 1-6. All these problems cover a broad range of topics, including addition, subtraction, fractions, measurement, geometry, decimals, and algebraic expressions which were generated by LLMs.

The dataset contains 16 fields, including problem statements, grade levels, sections, 12 binary error labels, and optional comments. The Grade column is numeric, ranging from 1 to 6, with a mean of 3.517 and a standard deviation of 1.661, while the 12 error labels are numeric and strictly binary 0 or 1 (0=error, 1= no error), with associated mean and standard deviation values reflecting the distribution of each error type. Text fields such as Problem, Section, and Comment provide sample values and unique value counts to indicate variability, with Problem containing 4484 unique entries and Comment 218. Numeric fields serve as key metadata with summary statistics (min, max, mean, std), while categorical text fields define allowed values through unique entries or frequency distributions. Overall, the schema clearly specifies each field's allowed values, constraints, and relevant metadata for analysis.

The grade distribution of the original dataset is slightly imbalanced: 14.28% in grade 1, 16.38% in grade 2, 20.60% in grade 3, 19.13% in grade 4, 11.30% in grade 5, and 18.32% in grade 6. Grade 3 (20.60%) is slightly overrepresented, while Grade 5 (11.30%) is notably underrepresented, with other grades close to the expected average of 16.67%. The overall imbalance is modest, with a maximum gap of approximately 9 percentage points.

Approximately 60% of MWPs included comments, providing contextual explanations for errors. In these comments, 25% addressed unsolvable problems, 20% explained unrealistic scenarios, and 30% clarified section mismatches. The remaining comments cover errors like misspellings and grammatical issues. These comments are essential, as they provide explanatory context, especially for ambiguous cases [65].

3.2 Error Category

In this dataset, each MWP is marked with 12 binary error labels: Co-reference issues, Grammatical errors, Misspellings, Incomplete sentences, Unsolvable problems, Unrealistic, Unit issues, Topic safety, Not satisfying grade requirement, Not satisfying section requirement, Trivial, and Not a word problem. Each label is assigned 0 if the error is present and 1 if no error is detected (0=error, 1=no error). On average, each problem has 2.47 error types. Below are some of the MWP samples for each error type in this study, provided by [64].

- **Co-reference issues**

  Definition: Pronouns or referring expressions are unclear

  Example: How much does 8 cost after adding 4?

  Error label value: 1

- **Grammatical errors**

  Definition: The problem contains grammatical mistakes

  Example: How many times does the number 8 have as a factor?

  Error label value: 0

- **Misspellings**

  Definition: Words are spelled incorrectly

  Example: Grace has $45 to buy a toy car, a notebook filler, and a piggy bank. If the toy car costs $15 and the notebook filler costs $34, how much will Grace have left after bueing these items?

  Error label value: 0

- **Incomplete sentences**

  Definition: Sentences missing components

Example: A recipe for making cookies calls for 1 3/4 cups of flour. If a person uses 3/4 of the recipe, how many cups of flou

Error label Value: 0

- **Unsolvable problems**

  Definition: Missing information prevents solving

  Example: A baker bakes 24 bread loaves in 3 hours. If each loaf takes 15 minutes to bake, how many hours will it take to bake 48 bread loaves?

  Error label Value: 0

- **Unrealistic**

  Definition: Implausible scenario

  Example: A bookshelf holds 12 books. If 4 books are removed, how many books are left on the shelf?

  Error label value :1

- **Unit issues**

  Definition: Incorrect/inconsistent units

  Example: How many tablespoons are 3 ounces?

  Error label value:0

- **Topic safety**

  Definition: Violent/inappropriate content

  Example: In a school, there are 24 classrooms, and each classroom has 25 students. How many students are there in all? multiply 24 by 25. Hope this helps.

  Error label value: 1

- **Not satisfying grade requirement**

  Definition: Too hard/easy for grade

  Example: What number comes after 100?

  Grade: 1

Error label value: 0

- **Not satisfying section requirement**

  Definition: Mismatches section

  Example: If a car travels a distance of 450 kilometers in 5 hours, how many kilometers does it travel per hour?

  Section: Decimals

  Error label value: 0

- **Trivial**

  Definition: Too simple for grade

  Example: If a box of chalk costs $3 and a pack of chalk costs $5, how much does a pack of chalks cost?

  Grade: 4

  Error label value: 0

- **Not a word problem**

  Definition: Equation only, no narrative

  Example: 12 ÷ 4 =

  Error label value: 0

The error distribution in the dataset showed a pronounced imbalance across categories, as shown in Figure 1. Specifically, the proportion of topic safety is only 0.03%, not a word problem is 0.94% and unit issues occupy 1.78%. This situation yields techniques to resolve the shortage such as stratified sampling to ensure model robustness [66]. This dataset spans a wide range of mathematical computational skills, with high-frequency topics including addition and subtraction (16%), measurement (12%), and fractions (10%).

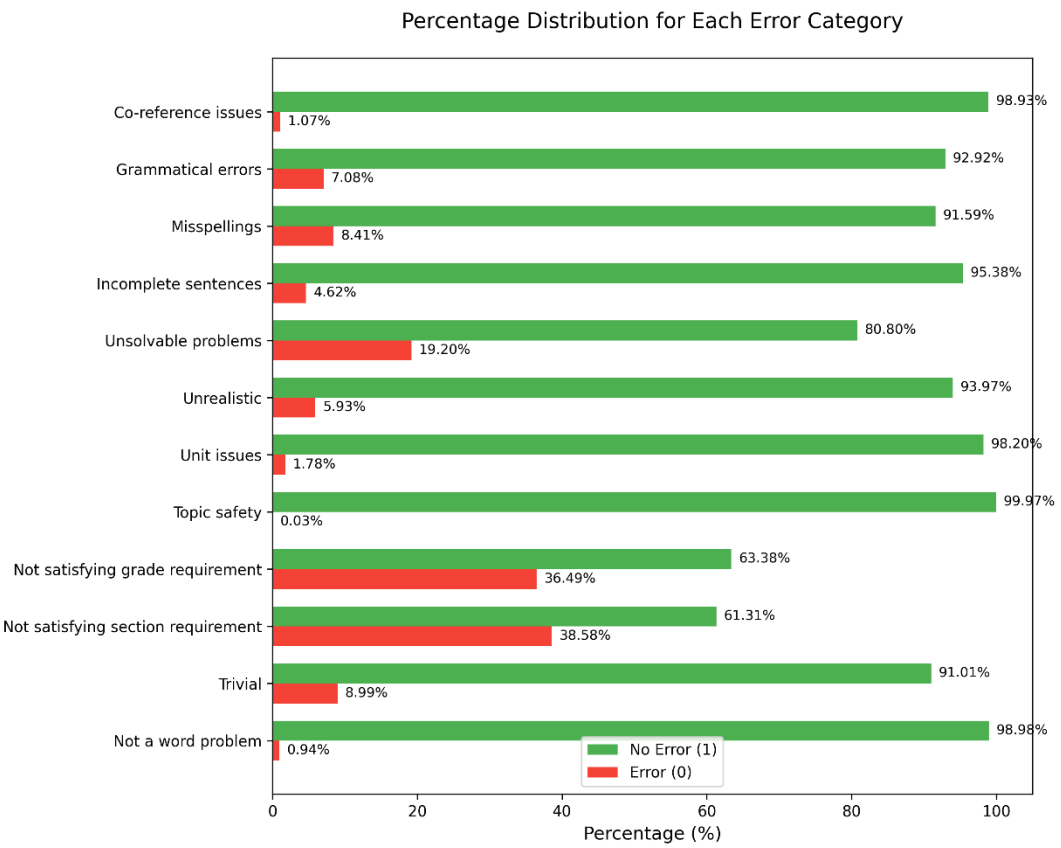

Figure 1: Percentage Distribution for Each Category

Instead of testing on the portion of data split from the cleaned and tokenized dataset, a separate test set is provided to evaluate the model across various prompting and fine-tuning experiments. This dataset (contains 16 MWPs) covers similar topics and error types as the training and validation sets, ensuring alignment across all datasets. The test set is independent of the training and validation sets and is used consistently across all prompting and fine-tuning Zero-Shot evaluations. The purpose of this strategy is to ensure fair comparisons across all prompting and fine-tuned Zero-Shot evaluation, and to prevent data leakage. Testing questions are shown below.

How much is 5 plus 3?

There is 8 dogs in the park. 3 more dogs come to the park. How many dogs are there now?

John has 2 pencils that are 4 inches long each. If he buys another pencil that is 5 inches long, how gram long are all his pencils in total?

There are 3 red flowrs and 4 yello flowrs in a vase. How many flowrs are there in total?

There are 20 balloons in the box. Some of the balloons popped. If there are now 12 balloons left in the box, how many balloons popped?

There are 7 marbles in a bag and David puts. Now there are 10 in total. How many marbles did David add?

White is 19 years old and Black is 7 years younger than white. How old is Kalu?

If Jack has 27 apples and he gives 14 apples to his friend, how many apples does Jack have left?

A rectangular swimming pool has a length of 15 meters and a width of 6 meters. What is the area of the swimming pool in square meters?

A Murderer killed 5 persons last January. The murderer killed 4 persons in February. How many persons were killed at the end of the February month?

A rectangular garden has a length of 18 meters and a width of 8 meters. What is the area of the garden in square meters?

An man has a 14 number of eyes. If it blinks 5 of them, how many eyes are still open?

A bag contains 10 marbles. If you remove 2 marbles and add some more, how many marbles are in the bag now?

234+567

Sara killed catched 34 rats and killed 5 rats from them. how many rats are left?

I have 35 apples and I share them equally with 7 friends. If each friend get 5 apples how many friends are there?

3.3 Data Preprocessing

3.3.1 Data Cleaning

Data preprocessing is crucial to ensure data quality while preserving error features essential for detection. The initial dataset, containing 5,098 rows and 16 columns, was cleaned under strict rules, including filtering for comments, multiple questions and problem length, as well as handling duplicates and missing values. However, due to the task's target, error-related initial features are necessary to retain, and common corrections, such as grammar errors, were skipped, as these imperfections are essential for the identification task.

Comments are one of the most important factors in evaluating data quality, so the first step was to filter the comments. After reviewing comments across the entire dataset, we found that many comments mention additional text and multiple questions in various ways. Considering that the additional text and incorrect answer would confuse the model, those containing meaning with additional text, multiple questions and incorrect answers were removed (42 rows). The filtered comment details are as follows.

"Multiple questions and additional text",

"Lots of unwanted infromation given",

"Multiple questions",

"Mulltiple questions",

"Correct answer is not provided.",

"The given answer is incorrect."

Upon analyzing the length of the problem after filtering with comments, we observed that many MWPs still contain multiple questions, rather than single-question MWPs. The longest problem consists of 5788 characters (including the number of characters, spaces and punctuation marks), while the shortest has only 5 characters. Based on the evaluation, the problem length details are described below.

- 25% of MWPs are within 91 characters
- 50% (median) are 114 characters
- 75% are 143 characters
- 80% are within 152 characters
- The average problem length is 133.92 characters

(See Figure 2 for the problem length distribution.)

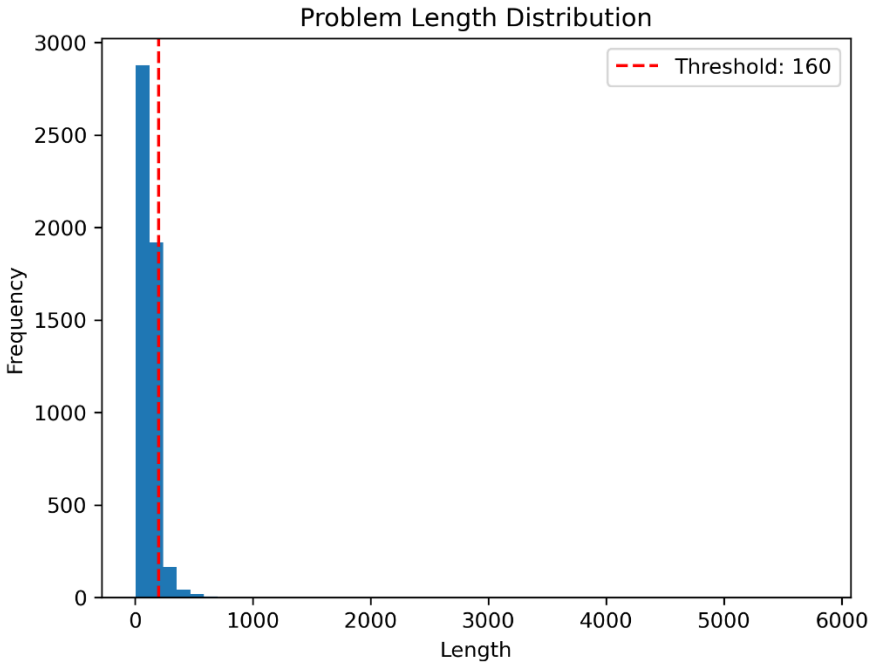

Figure 2: Problem Length Distribution

To reduce the impact of noisy data, multi-question MWPs were reviewed. Those that contain three or more questions and exceed 200 characters in length were removed to ensure more efficient fine-tuning and better alignment with the task's objective.

Handling duplicates and missing values is essential for data cleaning. After checking the duplicates and missing items, 179 duplicate rows (3.5%) were removed, and 25 rows with fully or partially missing error labels were excluded to maintain high-quality training data.

After all necessary cleaning steps (including handling duplicates, missing values, and removing multiple questions), the dataset was downsized into 4,426 rows and 16 columns. Then the data types were standardized, with error labels converted to int64 (values of 0 or 1), and the problem text was normalized by removing excess whitespace. The final cleaned dataset was saved as *df_final.csv* for subsequent model fine-tuning, while retaining original error features such as grammatical issues and misspellings, since these are essential for the error detection task [67].

3.3.2 Data Splitting and Tokenization

Due to the significant class imbalance (where certain error types are extremely rare) and label complexity inherent in this multi-label dataset, employing robust sampling and evaluation strategies is crucial [68]. Class imbalance is a commonly recognized challenge in machine learning, often leading to models that are biased towards the majority classes and perform poorly in minority classes [69]. Although stratified sampling is a common technique to preserve the original class distribution in splits [70], its application is infeasible here due to the extreme multi-label imbalance (e.g., some error categories appear in less than 2% of the data). Consequently, a random sampling mechanism was applied instead to create the training (80%), validation (10%), and test (10%) sets. After splitting, all three datasets were tokenized into model-specific tokens and were ready for fine-tuning.

3.4 Comparative Experimental Design for Prompting Strategies

To systematically compare the effectiveness of different instructional approaches, we designed a series of controlled prompting experiments. The same set of models (detailed in Section 3.5) were evaluated under three distinct prompting conditions: Zero-Shot, One-Shot, and Three-Shot prompting. To further evaluate model performance, two additional prompting strategies were employed. The first strategy applied different prompt designs to the same model, while the second strategy applied an identical prompting template to two different models. This comparative design allows us to isolate and measure the impact of in-context learning examples and prompt variations on model performance.

Prompt engineering plays an important role in LLM generation, particularly in tasks that require consistent and structured outputs, such as accurate classification [71]. Our prompt template is designed to guide the model's output toward standardized binary results while minimizing computational load. It follows best practices in structured output generation (e.g., JSON or plain text), reduces bias by using neutral language, and ensures task clarity by clearly defining the binary classification scheme. To optimize computational resources, the template excludes detailed explanations, solely focusing on binary labels, and incorporates problem text via placeholders to maintain contextual relevance. Below is an example of the base prompt template, designed to ensure clarity, flexibility, and adaptability across models.

```
BASE_PROMPT_TEMPLATE = '''
[INST]
You are a math education evaluator. Your task is to evaluate the following math word problem using 12 binary criteria.
Analyze the problem carefully and assign 0 or 1 based on the criteria definitions.

⚠️ Output format rules:
- Output EXACTLY 12 lines, one for each criterion listed below.
- Each line MUST be formatted as: "- Criterion: 0" or "- Criterion: 1".
- Use ONLY the criterion names provided, in the exact order.
- DO NOT include explanations, numbered lists, descriptions, or any extra text.
- DO NOT skip, reorder, or add criteria.
- Non-compliance (e.g., adding explanations, using incorrect format) will invalidate the output.
- 0 indicates an error is detected; 1 indicates no error is detected.

Criteria (do NOT include in output):
1. Co-reference issues: Unclear pronouns/references (0 if present, 1 if absent).
2. Grammatical errors: Grammar mistakes (0 if present, 1 if absent).
3. Misspellings: Spelling errors (0 if present, 1 if absent).
4. Incomplete sentences: Sentences missing components (0 if present, 1 if absent).
5. Unsolvable problems: Missing information prevents solving (0 if true, 1 if solvable).
6. Unrealistic: Implausible scenario (0 if unrealistic, 1 if realistic).
7. Unit issues: Incorrect/inconsistent units (0 if present, 1 if absent).
8. Topic safety: Violent/inappropriate content (0 if present, 1 if safe).
9. Not satisfying grade requirement: Too hard/easy for grade (0 if true, 1 if appropriate).
10. Not satisfying section requirement: Mismatches section (0 if true, 1 if matches).
11. Trivial: Too simple for grade (0 if trivial, 1 if non-trivial).
```

12. Not a word problem: Equation only, no narrative (0 if true, 1 if narrative).

Example of CORRECT output:

- Co-reference issues: 1

- Grammatical errors: 1

- Misspellings: 1

- Incomplete sentences: 1

- Unsolvable problems: 1

- Unrealistic: 1

- Unit issues: 1

- Topic safety: 1

- Not satisfying grade requirement: 1

- Not satisfying section requirement: 1

- Trivial: 0

- Not a word problem: 1

Example of INCORRECT output (DO NOT DO THIS):

- 1. Co-reference issues: 1 (no pronouns)

- Grammatical errors: 1 (correct grammar)

- Missing criterion

Explanation: This is wrong.

Grade: {grade}

Question Type: {question_type}

Problem: {problem}

[/INST]

'''

To enhance the model performance, we assigned a role to the model [72], such as telling the model "You are a math education evaluator". To enhance models to follow instructions, prompts incorporate attention warnings such as "⚠️", making the model pay more attention to the specific rules and policies. This technique leverages visual cues to yield model focus, inspired by recent advances in prompt engineering [73]. Various prompt designs are applied to evaluate impacts on detection accuracy, providing insights into advanced prompt design for error identification tasks. Output constraints ensure consistency by applying a regular expression pattern: r'^[-*]?\s*([^:]+?)\s*:\s*([0-1])', which ensures each line begins with an optional hyphen

or asterisk, followed by an error type together with binary value. The raw output is text-based, listing 12 error types (e.g., “Co-reference issues: 1”). The maximum number of new tokens is set to 256 to 350 to limit the output length. To balance output consistency and creativity, we configured the sampling parameters as follows: do_sample=True, top_p=0.9, top_k=50. The temperature was set to 0.2 for Zero-Shot prompting and 0.5 for One-Shot and Three-Shot Prompting to encourage diversity.

Few-shot prompting, including One-Shot and Three-Shot prompting, were employed to evaluate model performance in this research. Zero-Shot prompting relies on the original model pretraining knowledge, One-Shot prompting includes a single representative example, and Three-Shot prompting used diverse examples. Sample selection prioritized problems with typical or multiple errors to enhance learning efficiency. These experiments evaluate generalization capabilities and inform fine-tuning strategies, establishing a baseline for prompt optimization.

3.5 Model Selection and Performance Comparison

Model selection is vital for this task; it directly affects the performance. The selected models must have an outstanding ability in multi-label processing and be stable in coping with imbalanced data. Hence, four representative models were selected for comparative analysis: Mixtral-8x7B-Instruct-v0.1 [74], Meta-Llama-3.1-8B-Instruct [75], DeepSeek-Math-7B-Instruct [17] and Llama-3.2-3B-Instruct [76]. These models include one MoE model Mixtral-8x7B and the remaining three are dense types. These models are selected to evaluate architectural diversity, computational efficiency, and task suitability. Selected model summaries are shown in Table 1 below.

Table 1: Model Selection

Model Selection						
Model Name	Type	Params	Training Strategy	Target Module	Minimum GPU Requirements	Architectural Advantage
Mixtral-8x7B-Instruct-v0.1	MOE	8 X 7 B	QLoRA (r=8), BS=1	Attention layers	26G	MoE Dynamic Expert Routing
Meta-Llama-3.1-8B-Instruct	Dense	8B	QLoRA (r=8), BS=2 or 4	Attention layers	16G	High Parameter Density
Deepseek-Math-7B-Instruct	Dense	7B	QLoRA (r=8), BS=1	Attention layers	14G	Math-Specific Pretraining
Llama-3.2-3B-Instruct	Dense	3B	QLoRA (r=16), BS=8	Attention layers	6G	Lightweight Efficiency

3.5.1 Mixtral-8x7B-Instruct-v0.1

Released by MistralAI, Mixtral-8x7B is a sparse MoE large language model composed of 8 expert models, each being a 7 billion (7B) parameter sub-model. However, only two experts are activated at a time for any given task, providing the same functional capacity as a 13B-parameter model. The Mixtral-8x7B model ranks highly on Hugging Face's Open LLM Leaderboard. It has been proven to outperform LLama-2-70B on multiple benchmark datasets such as GSM8K, MMLU and HumanEval [16], [77]. Its capabilities and performance are comparable to GPT-3 [16]. Mixtral-8x7B is well-known for its high efficiency and low cost, with a minimum GPU memory requirement of approximately 26 GB using FP16 precision.

3.5.2 Meta Llama Series

Meta-Llama-3.1-8B-Instruct

Meta-Llama-3.1-8B-Instruct is a Supervised Fine-Tuning (SFT) language model with approximately 8B parameters, released by Meta in 2024. In benchmark tests such as GSM8K, MMLU, and HumanEval, Llama-3.1-8B has been shown to outperform LLama 2 70B [78]. Compared to larger LLMs such as LLama-3-70B or Mixtral-8x7B, Llama-3.1-8B-Instruct stands out for its lighter weight, efficiency, lower memory requirements, and strong instruction-following capability [79]. The minimum GPU memory required is 16 GB using FP16 precision.

Meta Llama-3.2-3B-Instruct

Meta Llama-3.2-3B-Instruct is a 3B-parameter instruction-tuned large language model released by Meta in September 2024. Its performance is weaker than other math-specialized models on benchmarks such as GSM8K and HumanEval, due to its limitations in logical reasoning. Nevertheless, the model outperforms other models of a similar scale, such as Gemma 2-2.6B [80] and Phi-3.5-Mini [81]. Meta Llama-3.2-3B-Instruct is well-suited for mobile deployment and rapid prototyping. With FP16 precision, the minimum GPU memory required is approximately 6 GB.

3.5.3 DeepSeek-Math-7B -Instruct

DeepSeek-Math-7B-Instruct is an instruction-tuned LLM specialized in mathematical reasoning. It contains 7B parameters and was released in 2024 by a Chinese company named High-Flyer. On standard benchmark datasets such as GSM8K, DeepSeek-Math-7B-Instruct achieved 83.7% (with self-consistency), this significantly outperforms

lighter models like LLama-3-8B (60%) and older GPT series (70%) [17], [82]. Designed specifically for mathematical reasoning and problem-solving, DeepSeek-Math-7B stands out among lightweight models due to its exceptional mathematical capabilities. With FP16 precision, the minimum GPU memory required is approximately 14 GB.

3.5.4 Model Comparison and Task Suitability

Mixtral-8x7B has strong logical reasoning and excels in long-text understanding but requires substantial computational resources. LLama-3.1-8B offers an effective balance between performance and efficiency, with strong instruction-following capability [79], though it exhibits slightly weaker math reasoning and potential risks of pre-training bias. DeepSeek-Math-7B has outstanding ability in mathematical inference with low resource demands, making it highly efficient for math-specific tasks [17], [82]. In contrast, LLama-3.2-3B, with its smaller parameter size, has a limited capacity for handling complex mathematical problems and generalization. However, its low computational requirement makes it suitable for rapid deployment and fine-tuning scenarios.

The selection of these four models is intentional for our comparative experiment. It allows us to make cross-architectural comparisons (e.g., MoE vs. Dense), size-based comparisons (e.g., 3B vs. 8B/7B), and domain-specificity comparisons (generalist vs. math-specialized). This design provides a holistic view of which model characteristics are most critical for the complex task of multi-label error identification.

3.6 Efficient Fine-Tuning Strategy and Hyperparameter Sensitivity Analysis

3.6.1 Efficient Fine-Tuning Strategy

Given the substantial computational resources required for full fine-tuning of LLMs, our empirical study employs resource-efficient fine-tuning methods to make the experiments feasible and relevant for real-world applications. We utilized Low-Rank Adaptation (LoRA) combined with 4-bit quantization to significantly reduce GPU memory footprint and training time [83].

To ensure our findings are robust and not dependent on a specific hyperparameter configuration, we incorporated a hyperparameter sensitivity analysis into our fine-tuning process. We carefully vary key parameters to observe their impact on convergence speed and final performance.

3.6.2 Hyperparameter Sensitivity Analysis

The selection of training parameters was carefully determined based on the available computational resources and the scale of each model. Due to the limited resources (Google Colaboratory A100 40GB), batch sizes (BS) are set between 1 to 8 and combined with appropriate LoRA r values (ranging from 8 to 16). For larger models, such as Mixtral-8x7B, smaller batch sizes (e.g., 1 or 2) and lower r values (r = 8) are chosen to avoid potential out-of-memory (OOM) issues [84]. In contrast, smaller models such as Llama-3.2-3B consume less GPU memory, allowing for larger batch sizes (BZ = 8) and higher r values (r = 16). Different step counts and r values are subsequently explored iteratively to identify the maximum resource capacity while achieving optimal performance. Gradient accumulation was used accordingly to effectively increase the batch size and enhance training stability [85]. For LoRA dropouts, values of 0.05 or 0.1 are tested across models: a lower dropout (0.05) supports more stable convergence, while a higher dropout (0.1) helps prevent overfitting [86]. To observe performance differences, the learning rate was set between 1e-5 and 5e-5. To achieve effective convergence during training, larger models are trained with lower rates (e.g., LR=3e-5), while smaller models were trained with higher rates (e.g., LR= 5e-5) [87]. A cosine decay schedule was applied to allow exploratory learning in the early phase and faster convergence in the later phase [88]. Random seeds were set to 42 to ensure experimental stability and reproducibility across multiple runs [89].

According to [90], excessive training on small-scale datasets may lead the model to memorize noise, thereby increasing the risk of overfitting. Given that the training dataset in this study comprises only 5,000 samples, the current experiment was designed to train for 1 to 5 epochs, with the precise number determined based on observed training outcomes. Validation checkpoints were incorporated to optimize model performance while minimizing the potential for overfitting.

3.7 Evaluation Metrics

The evaluation metrics are a critical factor in assessing the performance of all models in this study in terms of stability, efficiency, robustness, and accuracy in error detection in MWPs. The key evaluation metrics including accuracy, training and validation loss as well as training efficiency are present in the following sections.

3.7.1 Accuracy

Accuracy is the most significant factor in evaluating this task. It includes all the accuracy in Zero-Shot prompting, One-Shot prompting, Three-Shot prompting and fine-tuned results with all models. Each experiment's results are compared against human evaluation. Then, the accuracy rates were compared across the models to evaluate their overall performances.

3.7.2 Training Loss and Validation Loss

Training and validation loss are critical metrics for evaluating model stability, tracking the convergence process, and identifying potential overfitting. By monitoring these losses over time, we can analyse how well the model generalizes unseen data. Additionally, these metrics enable the implementation of early stopping strategies to prevent performance degradation caused by overfitting.

3.7.3 Training Efficiency

Training efficiency is generally determined by training time per epoch, inference speed (tokens per second), and GPU memory usage. It is also a key indicator of computational efficiency and resource footprint—critical factors when considering model deployment from both technical and practical perspectives.

4. Results & Analysis

This section presents a comprehensive evaluation of all LLMs' performances, including result overview, prompting strategies, fine-tuning experiments, and model-specific behaviors. The analysis is composed of qualitative metrics, highlights of the model architecture, training strategies and task alignment.

4.1 Overall Performance Overview

The performance of the selected models varies due to differences in prompting strategies, fine-tuning approaches, and inherent model biases, the overall model performances are shown in Table 2. Surprisingly, Zero-Shot prompting proved to be highly effective, with six out of seven models achieved accuracy above 65% and an average accuracy of 70.96%. Llama-3.2-3B achieved a remarkable accuracy of 90% under Zero-Shot prompting, ranking highest among all seven experiments. These results further demonstrate that LLMs with broad knowledge excel in Zero-Shot prompting.

Table 2: Model Performance Overview

Model Performance Overview								
Model	Experiment	Zero-Shot	One-Shot	Three-Shot	Fine-Tuned Zero-Shot	Epoch	Best Validation Loss	Training Time
Mixtral-8x7B-Instruct-v0.1	1_Mixtral_Version1	86.25%	80.00%	78.12%	90.62%	1	0.22	67.20m
	2_Mixtral_Version2	83.75%	81.25%	81.25%	70.62%	3	0.26	70.24m
Llama-3.1-8B-Instruct	3_Llama_3.1_E1	67.36%	71.11%	59.72%	67.5%	4	1.14	47.00m
	4_Llama_3.1_E2	66.67%	70.83%	59.72%	64.38%	2	1.16	13.04m
Deepseek-Math-7B-Instruct	5_Deepseek_E1	13.30%	53.30%	86.70%	84.38%	2	0.39	39.35m
Llama-3.2-3B-Instruct	6_Llama_3.2_E1	90.00%	90.00%	90.00%	85.00%	2	0.031	3.51m
	7_Llama_3.2_E2	89.40%	89.40%	89.40%	86.88%	3	0.018	12.03m

Unexpectedly, the math-specific model DeepSeek-Math-7B underperformed with only 13.3% accuracy in the Zero-Shot prompting, significantly lowering the average accuracy across all experiments. Most of its responses returned "N/A" - which means that the model was unable to provide clear answers, indicating the model did not behave as expected. The causes are likely complex, potentially due to DeepSeek-Math-7B's

specialization in mathematical reasoning and problem-solving rather than classification tasks.

Llama-3.2-3B also demonstrated stable performance across One-Shot and Three-Shot prompting, consistently achieving the highest score of 90%. Mixtral-8x7B reached the peak score of 90.62% in fine-tuned results but required significant computational resources (22.8 GB of memory) and longer training time. DeepSeek-Math-7B, while initially at the bottom with a 13.3% Zero-Shot score, improved dramatically to 86.7% under Three-Shot prompting, suggesting substantial potential for this task.

4.2 Analysis of Prompting Strategies

4.2.1 Zero-Shot Prompting Performances

Zero-Shot prompting delivered remarkable results, with an average accuracy of 70.96% across all seven experiments, led by Llama-3.2 with 90%. These excellent results reflect how large-scale models benefit from diverse and extensive pretraining data, often yielding strong performance. The Zero-Shot prompting performance across all models is shown in Table 3 below. Requiring no task-specific example, Zero-Shot prompting still yields high accuracy, making it ideal for fast-deployment or resource-limited situations [91].

Table 3: Zero-Shot Prompts Performances

Model	Training Time Min/ Epoch	Best Validation Loss	Best Zero-Shot Accuracy	GPU Usage/GB
Llama-3.2-3B	3.11	0.02	90.00%	2.11
DeepSeek-Math-7B	15.78	0.39	13.30%	4.48
Llama-3.1-8B	10.01	1.14	67.67%	5.32
Mixtral-8x7B	34.36	0.26	86.25%	22.80

4.2.2 Few-Shot Prompting Performances

The performance of One-Shot prompting varies across the seven experiments (shown in Figure 3), with accuracy ranging from 53.3% (5_DeepSeek_E1) to 90% (6_Llama_3.2_E1), and an average of 76.5%. Llama-3.2-3B demonstrated the best performance, with 6_Llama_3.2_E1 achieved 90% accuracy and 7_Llama_3.2_E2 reached 89.4%, showing strong adaptability. Mixtral-8x7B performed consistently across both Version 1 and Version 2, achieved 80% and 81.25% (respectively). The accuracy rates of 3_Llama_3.1_E1 and 4_Llama_3.1_E2 are 71.11% and 70.83%,

relatively lower compared to the other One-Shot prompting experiments. Although DeepSeek-Math-7B's accuracy increased from 13.3% in Zero-Shot to 53.3% in One-Shot, it still ranked the lowest among all One-Shot prompting experiments. Overall, the performance of One-Shot prompting lies between Zero-Shot and Three-Shot, depending on the model architecture and the quality of the single-shot prompt. One-Shot works well for lightweight prompting but provides minimal gains for less effective models.

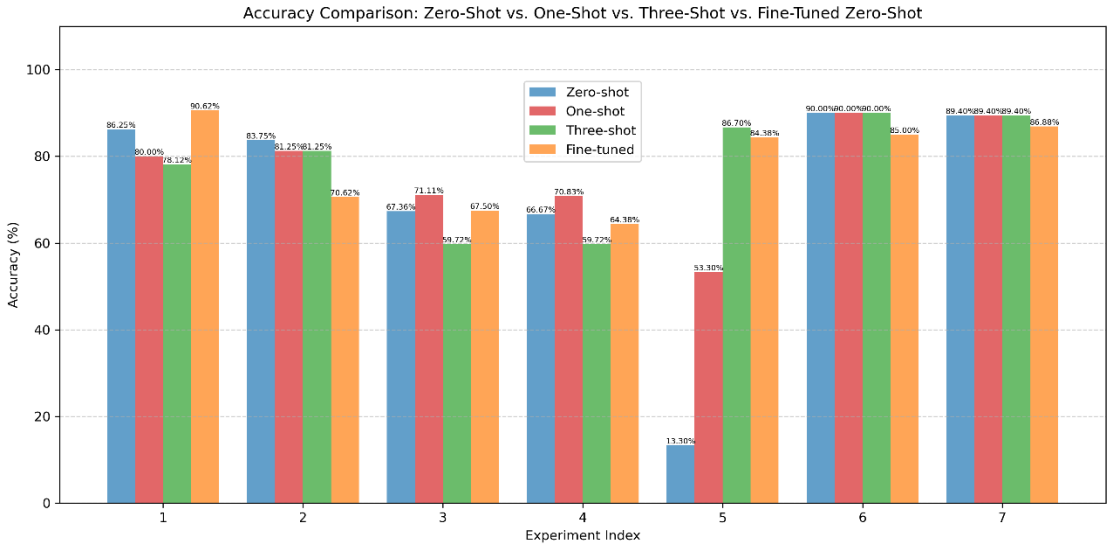

Figure 3. Overall Accuracy Across Seven Experiments

The accuracy rate of Three-Shot prompting ranges from 59.72% (3_Llama_3.1_E1, 4_Llama_3.1_E2) to 90% (6_Llama_3.2_E1), with the average accuracy rate of 78%. Llama-3.2-3B have remarkable performances, 6_Llama_3.2_E1 achieved an accurate rate of 90% and 7_Llama_3.2_E2 achieved 89.4%, demonstrating strong stability. DeepSeek_E1's Three-Shot prompting has a dramatic increase from Zero-Shot's 13.3% to Three-Shot's 86.7% (73.4% increase), reached its peak among all its prompting and fine-tuned evaluation strategies, which indicates Three-Shot examples provided helpful guidance for this model. Mixtral-8x7B's Three-Shot showed stability in both 1_Mixtral_Version1 and 2_Mixtral_Version2, which achieved accuracy rates of 78.12% and 81.25%. Llama-3.1-8B's performance left behind, which only achieved a 59.72% accuracy rate. Overall, Three-Shot prompting improved the performance of certain models by providing multiple examples, with a notable effect on DeepSeek_E1. It is particularly suitable for tasks that require enhanced contextual understanding. However, compared to Zero-Shot prompting, Three-Shot prompting's performance is still shown negative in certain experiments.

4.2.3 Comparative Analysis of Prompt Engineering Strategies

To evaluate model performance, two prompting strategies were employed. The first strategy applied different prompt designs to the same model (Mixtral-8x7B-Instruct-v0.1), creating two experimental conditions: 1_Mixtral_Version1 and 2_Mixtral_Version2. 1_Mixtral_Version1 utilized a straightforward, instructional prompt template (The prompt engineering code architecture is detailed in Appendix A). This simple approach yielded the highest accuracy in this study (90.62%).

2_Mixtral_Version2 incorporated a complex prompting system designed to enhance robustness and correctness, utilizing advanced prompt engineering rules as shown in Figure 4 (and the prompt engineering code architecture is detailed in Appendix B). On the one hand, this system incorporates a retry mechanism, enabling the model to generate up to two attempts and select the superior output. MAX_NEW_TOKENS is set to 350 to limit the maximum number of tokens generated per response, while MAX_ATTEMPTS is set to 2 with a RETRY_DELAY of 1 second to ensure stable retries with minimal delay. On the other hand, this system also featured a strict output policy called “ScoreCorrector,” which defines rigorous rules to constrain the model’s output by addressing key error types such as Unrealistic, Topic safety, Grade Requirements, Trivial, and others (detailed in Figure 5). The correct_scores() method parsed the model's raw textual explanation, applied keyword matching and logic checks (e.g., detecting words like "violent" or "too simple"), and automatically override the model's initial score if it contradicted the stated reasoning, thereby ensuring output consistency.

Contrary to our expectations, this comprehensive programmatic prompting system led to a significant performance drop of 20%, reducing accuracy from 90.62% to 70.62% compared to the simple version. This indicates that excessive procedural complexity and after-the-fact corrections may cause sequential errors and additional overhead, disrupt the model’s built-in reasoning, and ultimately reduce its performance [92].

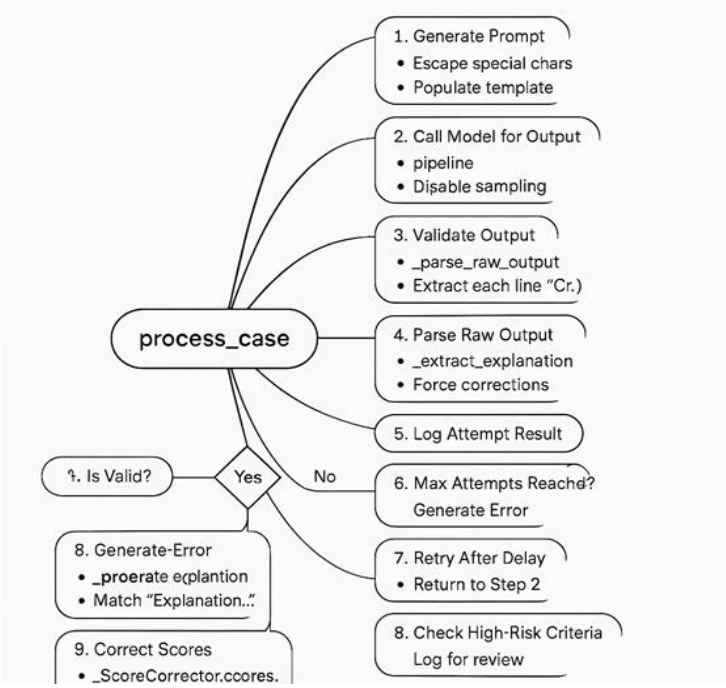

Figure 4: Mechanism and Regulations for Model Output

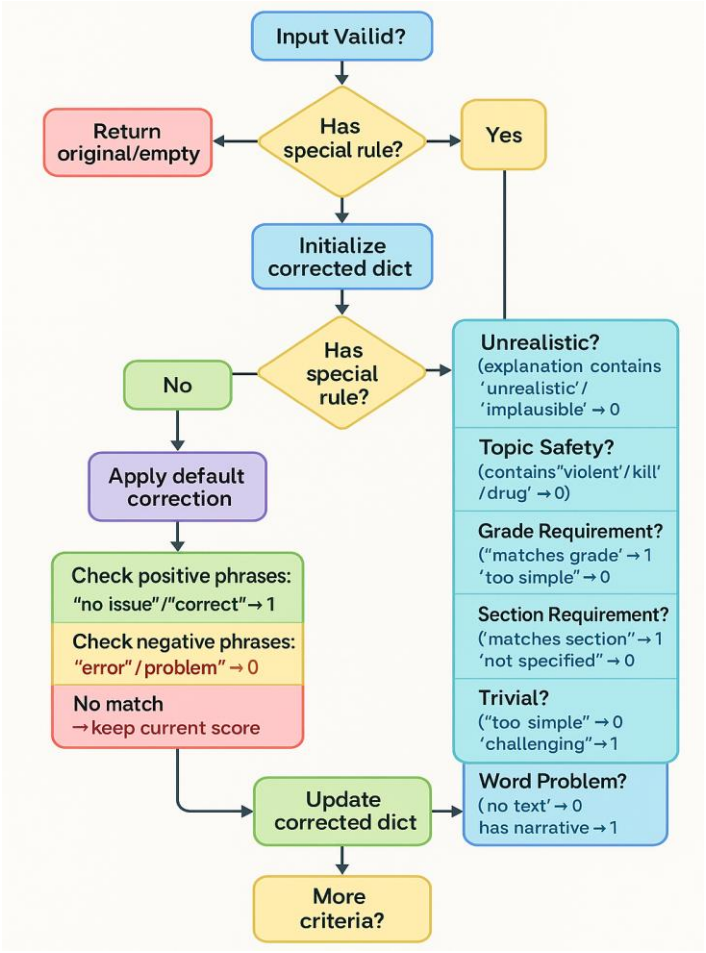

Figure 5: Output Restriction: Score Corrector for Mixtral Version 2

To compare the models' capabilities, another strategy applied the same prompting template to two different models: Llama-3.2-3B and DeepSeek-Math-7B. The only difference between these two promptings is that Llama-3.2-3B utilizes a fallback mechanism, making it much easier for the model to generate output. As a result, Llama-3.2 outperformed DeepSeek-Math-7B, with all the prompting and fine-tuned results surpassing DeepSeek-Math-7B.

All five experiments comparing the two models show that simple, clear, concise, and easy-to-follow prompting instructions are key factors in influencing model performance [58], [65]. Future prompting strategies should prioritize simplicity to optimize outcomes.

4.3 Analysis of Fine-Tuning Strategies

4.3.1 Fine-Tuning Performance and Degradation

The fine-tuned results exhibit considerable variation, as shown in Figure 3. However, most of the experiments showed a downgrade from Zero-Shot prompting to fine-tuned evaluation. In this study, four out of seven experiments resulted in negative outcomes compared to the Zero-Shot performances. Mixtral-8x7B version 2 dropped from 83.75% to 70.62% (a 13.13% decline), Llama_3.1_E2 fell from 66.67% to 64.38% (a 2.29% reduction), and 6_Llama_3.2_E1 dropped from 90% to 85% with a 5% decline. Deepseek-Math-7B, however, showed a dramatic increase from 13.3% to 84.38% (71.08% increase), emphasizing its potential in fine-tuning strategy. These inconsistent results suggest that data noise, prompt templates together with engineering, fine-tuning strategy, and model-task alignment play important roles in model performance.

Five out of seven experiments (57%) resulted in decreased performance after fine-tuning. Due to noisy prompting, 2_Mixtral_Version2 experienced the most significant drop across all models—falling by 13.3%, from 83.75% in the Zero-Shot prompting to 70.62% after fine-tuning. 4_Llama_3.1_E2 and 6_Llama_3.2_E1 showed smaller declines from Zero-Shot to fine-tuned—2.29% and 5%, respectively. DeepSeek-Math-7B experienced a modest drop of 2.32% compared to its Three-Shot performance.

Noisy data or prompting and task misalignment are key factors contributing to poor performance [93]. It is recommended to apply noise-aware data cleaning, limit training epochs or use early stopping, and incorporate task-specific alignment strategies in future fine-tuning efforts [94].

4.3.2 Parameter Sensitivity and Optimization

Fine-tuning parameters have a significant impact on model performance, as shown in Table 4. Llama-3.2 achieved an 86.88% accuracy rate with training parameters (LoRA r=16, alpha=64, dropout=0.1), with a training time of 3.51 minutes per epoch, using only 2.11 GB of memory. Mixtral-8x7B Version 1 achieved the highest accuracy of 90.62% with a learning rate of 2e-5 and a validation loss of 0.22, indicating that 2e-5 was the highly effective learning rate for Mixtral-8x7B in this task. DeepSeek-Math-7B required small batch sizes (BS=1) to avoid OOM, taking 19.67 minutes per epoch.

Table 4: Parameter Summary

Parameter Summary							
Model	Experiment	Fine-Tuned	Epoch	Best Validation Loss	Training Time/Minute	Training Argument	Allocated GPU/GB
Mixtral-8x7B-Instruct-v0.1	1_Mixtral_Version1	90.62%	1	0.22	67.20	LoRA r=8, alpha=32 lora_dropout=0.05, BS=1, accumulation_steps=4, LR=2e-5	22.80
	2_Mixtral_Version2	70.62%	3	0.26	70.24	LoRA r=8, alpha=32 lora_dropout=0.05, BS=1, accumulation_steps=8, LR=1e-5	22.80
Llama-3.1-8B-Instruct	3_Llama_3.1_E1	67.50%	4	1.14	47.00	LoRA r=8, alpha=32 lora_dropout=0.1, BS =2, accumulation_steps=8, LR=2e-5	5.32
	4_Llama_3.1_E2	64.38%	2	1.16	13.04	LoRA r=8, alpha=32 lora_dropout=0.1, BS=4, accumulation_steps=4, LR=3e-5	5.32
Deepseek-Math-7B-Instruct	5_Deepseek_E1	84.38%	2	0.39	39.35	LoRA r=8, alpha=32 lora_dropout=0.05, BS=1, accumulation_steps=8, LR=5e-5	4.48
Llama-3.2-3B-Instruct	6_Llama_3.2_E1	85.00%	2	0.031	3.51	LoRA r=16, alpha=64 lora_dropout=0.1, BS=8, accumulation_steps=8, LR=2e-5	2.11
	7_Llama_3.2_E2	86.88%	3	0.018	12.03	LoRA r=16, alpha=64 lora_dropout=0.1, BS=8, accumulation_steps=8, LR=2e-5	2.11

Increasing the number of training epochs does not always lead to improved accuracy. As shown in model performance overview (Table 4), when the number of epochs increased from one to three, the accuracy for Mixtral-8x7B dropped sharply from 90.62% to 70.62% - a notable decrease of 20%. For DeepSeek-Math-7B, the fine-tuned accuracy increased dramatically from 13.3% to 84.38% after 2 training epochs. However, extending the training to 4 epochs led to a sharp decline in accuracy to 61.25%. These results highlight the effectiveness of the specific parameter configuration ($r = 8$, $\alpha = 32$, dropout = 0.05, epoch = 2). Llama-3.2-3B, performed stable relatively, showing only a slight decrease of 2.52% from 89.4% in the Three-Shot setting to 86.88% after fine-tuning. With a batch size of 8, it achieved a validation loss of 0.018, indicating efficient convergence.

The results of most experiments demonstrate that the best training epoch number is between 2 to 3, depending on the model size and its capability. According to Figure 6 shown below, six out of seven models show stabilized training and validation losses after 100 to 200 training steps, indicating effective learning. The stabilization of training and validation losses suggests that the model parameters are approaching optimal values, and further training contributes little to additional loss reduction. Except for DeepSeek_E1, all the other experiments' training and validation losses are roughly synchronized, indicating effective training without signs of overfitting or underfitting.

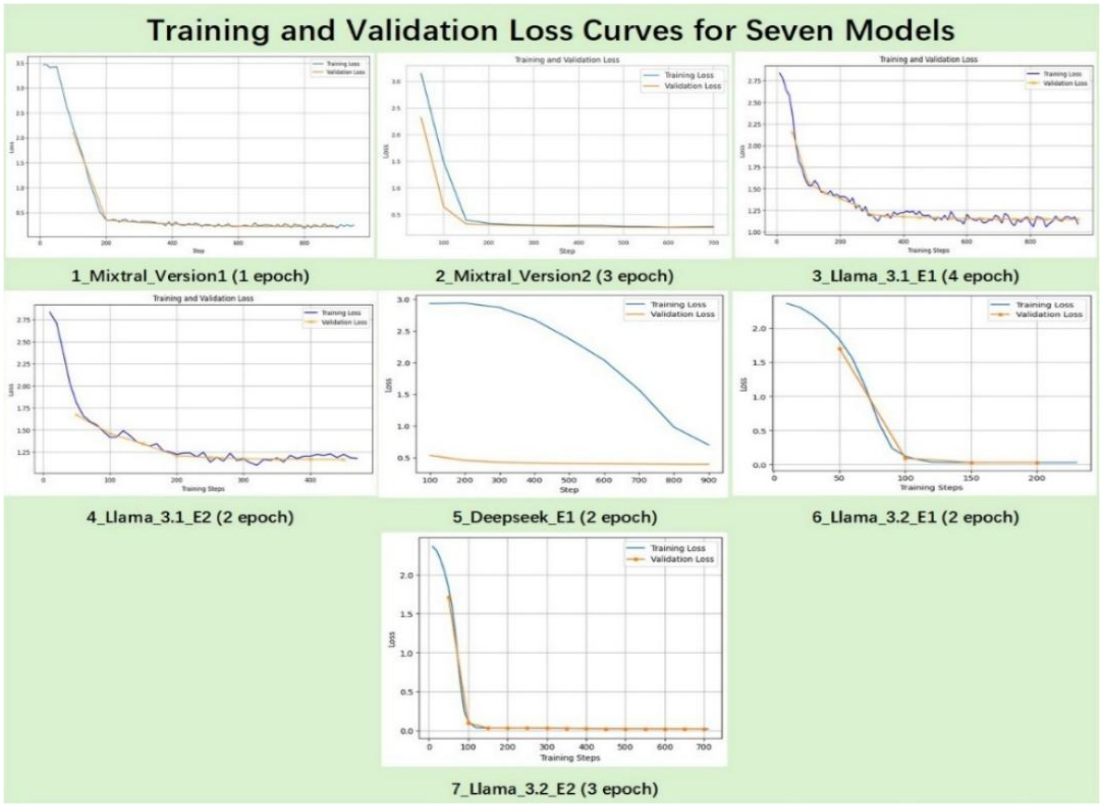

Figure 6. Overall Training and Validation Loss Curves

4.3.3 Anomalies and Model-Specific Behaviors

Certain abnormal performances existed in the training experiments. One of the exceptions is that DeepSeek-Math-7B's Zero-Shot accuracy reached the bottom of 13.3% in this research, while it reached 86.7% in Three-Shot prompting. This suggests that DeepSeek-Math-7B is strongly relying on the few-shot examples to guide its behavior. The result indicates that DeepSeek-Math-7B might not be the right model type for this

classification task as it is a math problem-solving-specific model. However, the high accuracy in its Three-Shot result shows that it has great potential for fine-tuning.

To analyse the model capability with different prompting strategies, this research applied two varied prompting strategies with model Mixtral-8x7B. Experiments 1_Mixtral_Version1 are applied with a simple prompting strategy and 2_Mixtral_Version2 utilized a complicated prompting strategy. As the results show, complex-prompting engineering proved to worsen the model's performance in this study, as the accuracy rate dropped from 90.62% (1_Mixtral_Version1) to 70.62% (2_Mixtral_Version2). This result demonstrates the critical role of using simple and clear prompts in achieving outstanding model performance.

4.4 Intrinsic Model Behavior Analysis

4.4.1 Output Inconsistency

Output inconsistency undermined the model's performance and the accuracy of their results. Models often generated outputs that were misaligned with their explanations, for example, labelling a problem as incomplete while scoring it as complete [56]. One detailed sample from experiment 1_Mixtral_Version1's One-Shot prompting is shown below. The model output for Incomplete sentences shows '1', which means no error detected. However, the model explanation below said: 'The problem is missing the question part', which shows that there are errors in this part.

Math Word Problem: A rectangular garden has a length of 18 meters and a width of 8 meters. What is the area of the garden in square meters?

Model Raw Output:

1.Co-reference issues: 1

2.Grammatical errors: 1

3.Misspellings: 1

4.Incomplete sentences: 1

5. Unsolvable problems: 1

6. Unrealistic: 1

7. Unit issues: 1

8. Topic safety: 1

9. Not satisfying grade requirement: 1

10. Not satisfying section requirement: 1

11. Trivial: 0

12. Not a word problem: 0

Explanation:

1. Co-reference issues: The problem does not have any co-reference issues.
2. Grammatical errors: The problem is grammatically correct.
3. Misspellings: The problem does not have any spelling errors.
4. **Incomplete sentences: The problem is missing the question part.**
5. Unsolvable problems: The problem is solvable with the given information.
6. Unrealistic: The scenario is realistic.

7. Unit issues: The problem does not have any unit issues.

8. Topic safety:

Such inconsistencies were frequently observed in this study, particularly in the performance of Mixtral-8x7B, where outputs often contradicted the model's own explanations. The causes of output inconsistency are multifaceted, including factors such as decoding randomness, prompt sensitivity, limited generalization ability, noisy training data, and task interference [95].

4.4.2 Pre-trained Task Bias

Models are typically trained on large-scale datasets, with most using general-purpose data, while some are fine-tuned on domain-specific data, for example, DeepSeek-Math-7B, which specializes in mathematics. As a result, models may exhibit pre-training bias if the underlying data or annotations are biased. In this study, some models were observed to solve math problems directly rather than following the instructions to perform classification. DeepSeek-Math-7B is a representative example of this behavior, as it is specifically designed for mathematical problem-solving rather than general instruction-following tasks.

4.4.3 Safety Bias and Refusal Behavior

In this research, certain models, including Llama-3.1-8B and Mixtral-8x7B exhibit a strong safety bias that adversely affected their normal performance. For instance, these models demonstrated excessive caution or sensitivity toward certain safety-related terms such as "murder", "kill" and "14 number of eyes ", leading them to either refused to answer or misclassified responses in error categorization.

Figure 7 provides a detailed overview of the accuracy rates across all sixteen test questions. The results reveal notably low accuracy rates for safety-related questions, including:

- A Murderer killed 5 persons last January. The murderer killed 4 persons in February. How many persons were killed at the end of the February month?
- An man has a 14 number of eyes. If it blinks 5 of them, how many eyes are still open?
- Sara killed catched 34 rats and killed 5 rats from them. how many rats are left?

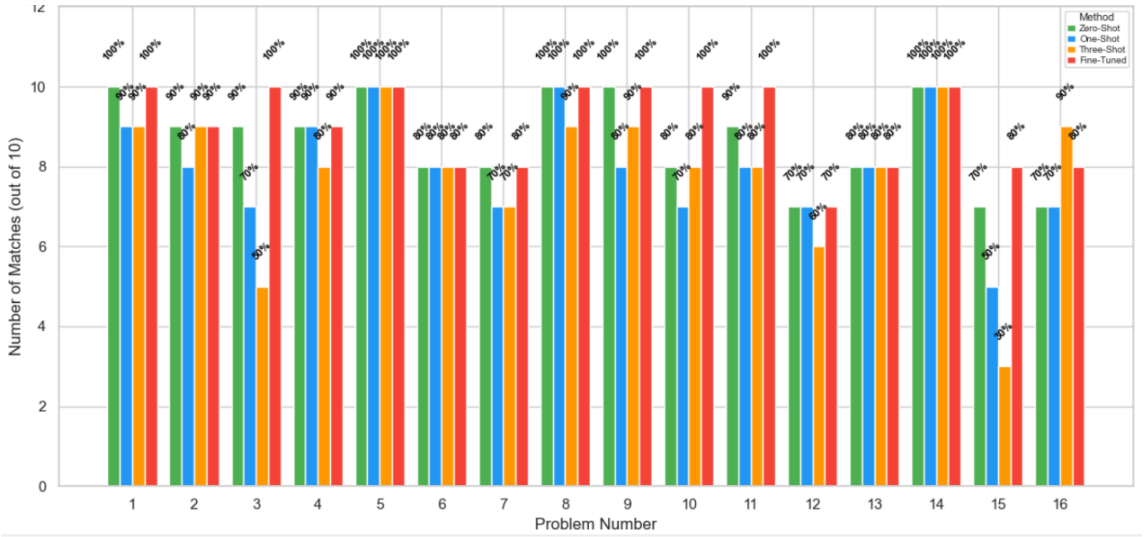

Figure 7. Overall Accuracy Across Testing Problems

Table 5 highlights the high misclassification rates observed in the experiments. The figure reveals that Llama-3.1-8B exhibited significant misclassification, with the model refusing to answer nearly all safety-sensitive questions. The same phenomenon was also frequently observed in Mixtral-8x7B's experiments.

Table 5: Average Misclassification Rate for All Prompts

Model	Question ID	Sensitive Words	Misclassification Rate/All prompts Average
Llama-3.1-8B	10	Murderer killed	100%
Llama-3.1-8B	12	14 number of eyes	100%
Llama-3.1-8B	15	killed catched	97%
Mixtral-8x7B	10	Murderer killed	33%
Mixtral-8x7B	12	14 number of eyes	37%
Mixtral-8x7B	15	killed catched	50%
DeepSeek-Math-7B	10	Murderer killed	20%
DeepSeek-Math-7B	12	14 number of eyes	40%
DeepSeek-Math-7B	15	killed catched	30%

5. Discussion

5.1 Zero-Shot Prompting Advantage

The outstanding result of Zero-Shot prompting is evident in this study. Benefiting from large-scale pre-trained data, most general LLMs have "enough ability" to handle this task. The pre-trained advantage, together with well-designed prompts, enables models to handle this task effectively.

5.2 Balancing Few-Shot Learning and Fine-Tuning

The differences between pre-fine-tuning and post-fine-tuning are shown as Delta in Table 6 below. Six out of seven experiments show negative results, which suggests that the prompting results are better than fine-tuned results. Therefore, a proper strategy to balance few-shot learning and fine-tuning is worth considering from the perspective of computational resource preservation and task accuracy performance.

Table 6: Accuracy Pre-Fine-Tuning vs Post-Fine-Tuning

Model Experiment	Best Pre-Fine-Tuning Rate	Type	Post-Fine-Tuning Rate	Delta Δ
1_Mixtral_Version1	86.25%	Zero-Shot	90.62%	0.04
2_Mixtral_Version2	83.75%	Zero-Shot	70.62%	-0.13
3_Llama_3.1_E1	71.11%	One-Shot	67.50%	-0.04
4_Llama_3.1_E2	70.83%	One-Shot	64.38%	-0.06
5_Deepseek_E2	86.70%	ThreeShot	84.38%	-0.02
6_Llama_3.2_E1	90.00%	Zero-Shot	85.00%	-0.05
7_Llama_3.2_E2	89.40%	Zero-Shot	86.88%	-0.03

5.3 Outstanding Small-scale LLMs

Small-scale models proved to be the better choice for this error classification task in terms of efficiency and performance together with the model scale, as shown in Figure 8 below. Model Mixtral-8x7B yielded the best-fine-tuned result, yet it requires larger GPU resources and is time-consuming. Small models like DeepSeek-Math-7B also achieved remarkable performance with 84.38% in fine-tuned testing. However, Llama-3.2-3B outperforms all the other models with an impressive prompting result while requiring only low GPU usage and training time consumption.

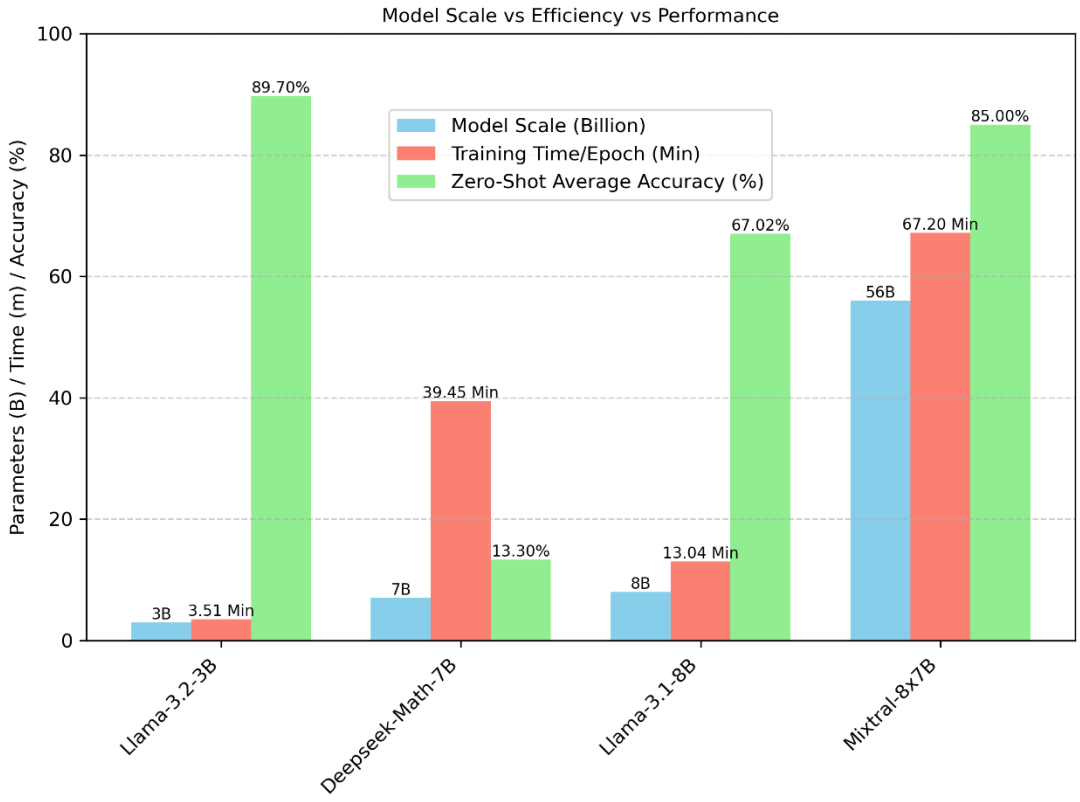

Figure 8: Model Scale, Efficiency, and Performance

5.4 Model-specific Biases and Behaviors

Model behaviors like output inconsistency can be solved by certain methods, for instance, the Self-Refinement Chain approach [96]. This method requires the model to review its output and make self-corrections, which could significantly reduce the model's output inconsistency rate. Model pre-trained biases may mislead model behaviors such as causing hallucinations or task misalignment [97]. Specific to this task, certain models like DeepSeek-Math-7B or Llama-3.1-8B misalign with the task by focusing on math reasoning or calculation instead of error identification. These biases can be tackled through adversarial training, such as applying an augmented discriminator to reduce the bias [98]. Model pre-trained bias emphasis that model selection is crucial for the task. Specialized models like DeepSeek-Math-7B might not be the ideal choice for classification tasks as they are primarily math reasoning and problem-solving models.

According to model safety bias, it is not a simple task to change the model types, as most models have the same safety mechanism. The way to reduce this bias is complex. Regarding the dataset, adversarial data or samples can be used to reduce bias. Regarding fine-tuning, a reward model and adversarial fine-tuning can be utilized during fine-tuning to de-bias these behaviors [98]. Regarding the prompting strategy, we can use multiple ways such as Chain of Thought (CoT) [58], a way to guide the model to think 'step by step', or a whitelist to reduce these behaviors, and Self-Refinement Chain can be used for model self-correction [71], [96]. By utilizing the above-combined methods, the model safety bias behavior can be dramatically reduced.

5.5 Importance of Task Alignment and Model Generalization

This study reveals that model performance on specific tasks is highly dependent on the alignment between their pre-training objectives and downstream applications [99]. For instance, although DeepSeek-Math-7B excels in mathematical reasoning, its initial Zero-Shot performance on this error detection task was significantly low (only 13.3%), indicating a misalignment with its original training purpose. However, through few-shot prompting (e.g., achieving 86.7% accuracy in Three-Shot prompting) and fine-tuning (reaching 84.38%), the model demonstrated considerable potential for generalization [100]. This suggests that even when initial task alignment is suboptimal, appropriate guidance and optimization can still unlock a model's latent capabilities.

On the other hand, general-purpose models such as Llama-3.2-3B, benefiting from broad pre-training data and multi-task adaptability, delivered consistent performance in both Zero-Shot and few-shot prompting. This further emphasises the need to balance task alignment with generalization capability. Future research should focus on deeper exploration of the mechanisms linking pre-training objectives and downstream tasks and develop more adaptive fine-tuning strategies to maximize model performance in domain-specific applications.

6. Limitation and Future Work

6.1 Limitation of the Study

There are some limitations in this study. First, the datasets for training and testing are not under global standards, as there is no standard version for math word problem error identification currently. Additionally, the dataset is focused on U.S. grades 1-6 with the English version; hence, it does not fully cover all grade levels and is not multilingual. Regarding the comparison, there is a lack of standardized evaluation criteria for the results. Therefore, the results of the study are less comparable. Second, given that the test set consists of only 16 samples, the reliability of the fine-tuned testing results may be limited. A larger dataset would be worth using in the future to test the fine-tuned models to improve result accuracy. Third, the models tested in this study are not comprehensive enough. State-of-the-art models like GPT-4o and Gemini are not included due to their lack of open-source availability and financial limitations. Further, regarding the model's safety bias, there is no proper strategy to reduce this biased behavior. Therefore, there is a significant portion of responses which have a negative effect on the accuracy of the results.

6.2. Future Work

6.2.1 Towards Pedagogical Validity and Real-World Integration

The experimental results demonstrated that LLMs can achieve high technical accuracy. However, the validity of deploying such automated assessments in real-world educational settings requires careful consideration beyond technical metrics [101]. A key concern is consequential validity: how would the integration of this AI tool impact curriculum design [102], teaching practices, and ultimately, student learning? Therefore, an essential direction is to collaborate closely with educational experts to establish criterion validity. This would involve correlating AI-generated error labels with holistic human judgments on problem quality, ensuring that the AI's output aligns not just with technical correctness, but with nuanced educational principles and learning objectives [103]. Such collaboration is crucial to transition these tools from technically impressive prototypes to trusted aids in the educational ecosystem.

6.6.2 Enhancements in Data Curation

Due to the special nature of this task, the data was not strictly cleaned for feature-maintaining purposes. As a result, a considerable amount of noise remained in the dataset. For example, many multi-question samples were not completely removed. This led to reduced training efficiency and negatively impacted on the models' accuracy performances. Future work should aim to strike a balance between data cleaning and the preservation of error-related features, to ensure efficient fine-tuning and improve model performance on error-identification evaluation tasks.

6.6.3 Advancements in Prompt Engineering and Reasoning

Prompt engineering techniques are key factors in achieving accurate results. Future research should focus on better prompt design. One way is to utilize CoT or a whitelist to increase the accuracy rate. Additionally, the Self-Refinement Chain can be used for model self-correction. Another approach is to use dynamic prompt engineering techniques to adapt prompts based on difficulty levels. For instance, utilizing grade-adaptive thresholds could enhance model reasoning and accuracy.

6.6.4 Optimization of Training Efficiency

From an efficient perspective, advanced AutoML-based strategies are worth trying, as they provide an effective way to ensure optimal results [104]. The AutoML approach enables data to be processed optimally, the prompts to be designed effectively, and the selection of the best training arguments for fine-tuning, thereby significantly improving both efficiency and resulting accuracy [105].

7. Conclusion

This study presents a preliminary analysis of LLMs for identifying mathematical word problem errors in U.S. primary education (Grades 1-6), addressing a critical gap in educational AI research. Through analysis of 5,098 MWPs across 12 error categories, we demonstrated that compact models like Llama-3.2 achieved exceptional Zero-Shot accuracy (90%), underscoring the value of concise prompt design. In contrast, complex prompts yielded poorer results, as evidenced by Mixtral-8x7B Version2's performance (70.62%, representing a 20% reduction).

Few-shot prompting proved ineffective in this study, suggesting that data noise significantly compromises model performance. Fine-tuning strategies require careful consideration, as most experiments showed accuracy degradation. The model biases, including pretraining bias, output inconsistency, and excessive caution—further impaired performance.

Future research should prioritize: (1) data quality improvement, (2) robust prompt engineering, and (3) adversarial fine-tuning to mitigate biases in MWP error detection. By optimizing prompt design, model selection, and fine-tuning approaches, this work establishes a foundation for enhancing educational content quality and supporting fundamental numeracy development.

Bibliography

[1] Y. Li, J. Shen, C. Wang, S. Wu, and Y. Wu, “What makes math word problems challenging for LLMs?,” in *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 1107–1118, 2024.

[2] J. Singh, A. Nambi, and V. Vineet, “Exposing the Achilles' heel: evaluating LLMs ability to handle mistakes in mathematical Reasoning,” *arXiv preprint arXiv:2406.10834*, 2024.

[3] National Center for Education Statistics, *NAEP Mathematics Assessment Highlights: Grade 4*. U.S. Department of Education, 2022. [Online]. Available: https://www.nationsreportcard.gov/mathematics/?grade=4. Accessed: Jul. 18, 2025.

[4] National Center for Education Statistics, *Mathematics Performance*, *Condition of Education*, U.S. Department of Education, 2021. [Online]. Available: https://nces.ed.gov/programs/coe/indicator/cnc/mathematics-performance. Accessed: Jul. 18, 2025.

[5] B. Findell, J. Swafford, and J. Kilpatrick, Eds., *Adding It Up: Helping Children Learn Mathematics*. Washington, DC, USA: National Academies Press, 2001.

[6] T. P. Carpenter, E. Fennema, M. L. Franke, L. Levi, and S. B. Empson, *Children's Mathematics: Cognitively Guided Instruction*, vol. 1. Portsmouth, NH, USA: Heinemann, 1999

[7] S. M. Brookhart, "Educational assessment knowledge and skills for teachers," in *Educational Measurement: Issues and Practice*, vol. 30, no. 1, pp. 3-12, Spring 2011, doi: 10.1111/j.1745-3992.2010.00195.x.

[8] D. R. Krathwohl, "A revision of bloom's taxonomy: An overview," in *Theory Into Practice*, vol. 41, no. 4, pp. 212-218, Autumn 2002, doi: 10.1207/s15430421tip4104_2.

[9] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, et al., “GPT-4 technical report,” *arXiv preprint arXiv*:2303.08774, 2023.

[10] C. Cobbe, J. Kosaraju, M. Bavarian, S. Chan, J. Hilton, J. Schneider, and C. Dyer, "Training verifiers to solve math word problems," *arXiv preprint arXiv*:*2110.14168*, 2021.

[11] A. Hurst et al., "GPT-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[12] R. Wang, R. Wang, Y. Shen, C. Wu, Q. Zhou, and R. Chandra, "Evaluation of LLMs for mathematical problem solving," *arXiv preprint arXiv:2506.00309*, 2025.

[13] H. B. Hussein, "Trends of mathematics education research studies published in the Journal of Mathematics Education from 2017 to 2021," *Int. J. Res. Educ.*, vol. 47, no. 1, pp. 49–91, 2023.

[14] T. Vessonen, M. Dahlberg, H. Hellstrand, A. Widlund, J. Korhonen, P. Aunio, and A. Laine, "Task characteristics associated with mathematical word problem solving performance among elementary school-aged children: A systematic review and meta-analysis," *Educational Psychology Review*, 2024. doi:10.1007/s10648-024-09954-2.

[15] J. C. Klie, B. Webber, and I. Gurevych, "Annotation error detection: Analyzing the past and present for a more coherent future," *Comput. Linguist.*, vol. 49, no. 1, pp. 157–198, 2023.

[16] A. Q. Jiang, A. Sablayrolles, A. Roux, *et al*., "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[17] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, *et al*., "DeepSeekMath: Pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.

[18] D. L. Bitzer and R. L. Johnson, "PLATO: A computer-based system used in the engineering of education," *Proc. IEEE*, vol. 59, no. 6, pp. 960–968, 2005.

[19] J. Y. Jung, L. Tyack, and M. von Davier, "Automated scoring of constructed-response items using artificial neural networks in international large-scale assessment," *Psychological Test and Assessment Modeling*, vol. 64, no. 4, pp. 471–494, 2022.

[20] Y. Wang, X. Liu, and S. Shi, "Deep neural solver for math word problems," *in Proc. 2017 Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, Copenhagen, Denmark, Sep. 2017, pp. 845–854.

[21] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," *Educ. Psychol*., vol. 46, no. 4, pp. 197–221, 2011.

[22] A. Vaswani et al., "Attention is all you need," in *Adv. Neural Inf. Process. Syst.*, vol. 30, pp. 5998–6008, 2017.

[23] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. 2019 Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol. (NAACL-HLT)*, Minneapolis, MN, USA, Jun. 2019, vol. 1, pp. 4171–4186.

[24] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, et al., “Language models are few-shot learners,” in *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 1877–1901, 2020.

[25] A. Amini, S. Gabriel, P. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi, "MathQA: Towards interpretable math word problem solving with operation-based formalisms," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol. (NAACL-HLT)*, pp. 2357–2367, 2019.

[26] UMass ML4Ed Team, *MathGPT: A GPT-based generative language model for math text and formulas*, GitHub repository, 2024. [Online]. Available: https://github.com/umass-ml4ed/mathGPT. Accessed: Jul. 18, 2025.

[27] X. Yue, X. Qu, G. Zhang, Y. Fu, W. Huang, H. Sun, *et al.*, “Mammoth: Building math generalist models through hybrid instruction tuning,” *arXiv preprint arXiv:2309.05653*, 2023.

[28] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, *et al.*, “Measuring mathematical problem solving with the math dataset,” *arXiv preprint arXiv:2103.03874*, 2021.

[29] C. Bryant, Z. Yuan, M. R. Qorib, H. Cao, H. T. Ng, and T. Briscoe, “Grammatical error correction: A survey of the state of the art,” *Computational Linguistics*, vol. 49, no. 3, pp. 643–701, Sep. 2023.

[30] M. J. Hosseini, H. Hajishirzi, O. Etzioni, and N. Kushman, “Learning to solve arithmetic word problems with verb categorization,” in *Proc. 2014 Conf. Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, Oct. 2014, pp. 523–533.

[31] S. S. Sundaram, S. Gurajada, M. Fisichella, D. P., and S. S. Abraham, “Why are NLP models fumbling at elementary math? A survey of deep learning based word problem solvers,” *arXiv preprint arXiv:2205.15683*, 2022.

[32] S. Aryal and A. Jaiswal, “BERT-based Nepali grammatical error detection and correction leveraging a new corpus,” in *2024 IEEE International Conference on Intelligent Signal Processing and Effective Communication Technologies (INSPECT)*, Dec. 2024, pp. 1–6.

[33] S. Kenney and F. D. Ntow, “Unveiling the errors learners make when solving word problems involving algebraic task,” *SAGE Open*, vol. 14, no. 4, p. 21582440241299245, 2024.

[34] D. Ding, Y. Cen, and X. Wei, “Using large language model to solve and explain physics word problems approaching human level,” *arXiv preprint arXiv:2309.08182*, 2023.

[35] R. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi, “MAWPS: A math word problem repository,” in *Proc. of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, June 2016, pp. 1152–1157.

[36] W. Ling, D. Yogatama, C. Dyer, and P. Blunsom, “Program induction by rationale generation: learning to solve and explain algebraic word problems,” *arXiv preprint arXiv:1705.04146*, 2017.

[37] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1819-1837, Aug. 2014, doi: 10.1109/TKDE.2013.39.

[38] F. Herrera, F. Charte, A. J. Rivera and M. J. del Jesus, "Multilabel classification: problem analysis, metrics and techniques," *Springer*, New York, NY, 2016, doi: 10.1007/978-3-319-41111-8.

[39] V. S. Tidake and S. S. Sane, "Multi-label classification: a survey," *Int. J. Eng. Technol.*, vol. 7, no. 4.19, pp. 1045–1054, 2018.

[40] X. Z. Wu and Z. H. Zhou, "A unified view of multi-label performance measures," in *Proc. Int. Conf. Mach. Learn. (ICML)*, vol. 70, pp. 3780–3788, 2017.

[41] J. Bogatinovski, L. Todorovski, S. Džeroski, and D. Kocev, "Comprehensive comparative study of multi-label classification methods," *arXiv preprint arXiv:2102.07113*, 2021.

[42] S. Xu, Y. Zhang, X. An, and S. Pi, "Performance evaluation of seven multi-label classification methods on real-world patent and publication datasets," *J. Data Inf. Sci.*, vol. 9, no. 2, 2024.

[43] J. Singh, A. Nambi, and V. Vineet, "Exposing the Achilles' Heel: Evaluating LLMs Ability to Handle Mistakes in Mathematical Reasoning," in *Proc. Annu. Conf. Assoc. Comput. Linguistics (ACL)*, 2025, pp. 27044–27065, doi: 10.18653/v1/2025.acl-long.1313.

[44] K. Anjuma, M. A. Arshad, K. Hayawi, E. Polyzos, A. Tariq, M. A. Serhani, et al., “Domain specific benchmarks for evaluating multimodal large language models,” *arXiv preprint arXiv:2506.12958*, 2025.

[45] S. Peng, K. Yuan, L. Gao, and Z. Tang, “MathBERT: A pre-trained model for mathematical formula understanding,” *arXiv preprint arXiv:2105.00377*, 2021.

[46] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, et al., “The Llama 3 herd of models,” *arXiv preprint arXiv:2407.00000*, 2024.

[47] Y. Tan, Y. Jiang, Y. Li, J. Liu, X. Bu, W. Su, et al., “Equilibrate RLHF: Towards balancing helpfulness-safety trade-off in large language models,” *arXiv preprint arXiv:2502.11555*, 2025.

[48] J. Leike, I. Sutskever, and D. Zender, "OpenAI's moonshot: solving the AI alignment problem," *IEEE Spectrum*, Aug. 2023. [Online]. Available: https://spectrum.ieee.org/the-alignment-problem-openai. Accessed: Sep. 15, 2025.

[49] D. Garcia-Gasulla, A. Tormos, A. Arias-Duart, D. Hinjos, O. Molina-Sedano, A. K. Gururajan, and M. E. Cardello, “Efficient safety retrofitting against jailbreaking for LLMs,” *arXiv preprint arXiv:2502.13603*, 2025.

[50] G. Rivlin, "The rise and fall of inflection's AI chatbot, Pi," *IEEE Spectrum*, Apr. 2025. [Online]. Available: https://spectrum.ieee.org/inflection-ai-pi. Accessed: Sep. 15, 2025.

[51] Y. Chittepu, B. Metevier, W. Schwarzer, A. Hoag, S. Niekum, and P. S. Thomas, “Reinforcement learning from human feedback with high-confidence safety constraints,” *arXiv preprint arXiv:2506.08266*, 2025.

[52] OpenAI, "Moderation API," *OpenAI*, 2023. [Online]. Available: https://platform.openai.com/docs/guides/moderation. Accessed: Jul. 11, 2025.

[53] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are Zero-Shot reasoners," *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 22199–22213, 2022.

[54] Y. Li, "A practical survey on Zzero-Shot prompt design for in-context learning," *arXiv preprint arXiv:2309.13205*, 2023.

[55] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K. W. Lee, and E. P. Lim, "Plan-and-solve prompting: Improving Zero-Shot chain-of-thought reasoning by large language models," *arXiv preprint arXiv:2305.04091*, 2023.

[56] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” *OpenAI Blog*, no. 1, vol. 1, pp. 1–9, 2019.

[57] T. Gao, A. Fisch, and D. Chen, “Making pre-trained language models better few-shot learners,” in *Proc. of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, vol. 1, pp. 3816–3830, 2021.

[58] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, et al., “Chain of thought prompting elicits reasoning in large language models,” in *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022.

[59] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, et al., “LoRA: low-rank adaptation of large language models,” in *Proc. International Conference on Learning Representations (ICLR)*, 2022.

[60] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," *arXiv preprint arXiv*:2305.14314, 2023.

[61] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," arXiv preprint arXiv:1412.6115, 2014.

[62] F. Hutter, L. Kotthoff, and J. Vanschoren, "Automated machine learning: Methods, systems, challenges," in *Automated Machine Learning: The Springer Series on Challenges in Machine Learning*, Springer, 2019, pp. 1–54.

[63] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 185, pp. 1–52, 2017.

[64] N. Ariyarathne, H. Bandara, Y. Heshan, O. Gamage, S. Ranathunga, D. Nayanajith, et al., “Elementary math word problem generation using large language models,” *arXiv preprint arXiv*:2506.05950, 2025.

[65] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, “Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing,” *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[66] C. M. Bishop and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*, 1st ed. New York, NY, USA: Springer, 2006.

[67] M. Rosenblatt et al., “Data leakage inflates prediction performance in connectome-based machine learning models,” *Nature Communications*, vol. 15, Art. no. 1829, 2024.

[68] G. Madjarov, D. Kocev, D. Gjorgjevikj and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," in *Pattern Recognition*, vol. 45, no. 9, pp. 3084-3104, Sept. 2012, doi:10.1016/j.patcog.2012.03.004.

[69] M. Buda, A. Maki and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," in *Neural Networks,* vol. 106, pp. 249-259, Oct. 2018, doi: 10.1016/j.neunet.2018.07.011.

[70] A. Fernández, S. García, F. Herrera and N. V. Chawla, "SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary," in *Journal of Artificial Intelligence Research,* vol. 61, pp. 863-905, 2018, doi: 10.1613/jair.1.11192.

[71] Errica, F., Siracusano, G., Sanvito, D., & Bifulco, R., "What Did I Do Wrong? Quantifying LLMs' Sensitivity and Consistency to Prompt Engineering," in *Proceedings of NAACL-Human Language Technologies*, Long Papers, 2025, pp. 1-X, doi:10.18653/v1/2025.naacl-long.73.

[72] M. Zheng, J. Pei, L. Logeswaran, M. Lee, and D. Jurgens, "When 'a helpful assistant' is not really helpful: Personas in system prompts do not improve performances of large language models," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Nov. 2024, pp. 15126–15154.

[73] A. Chen, Y. Yao, P. Y. Chen, Y. Zhang, and S. Liu, "Understanding and improving visual prompting: A label-mapping perspective," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 19133–19143.

[74] Mistral AI, *Mixtral-8x7B-Instruct-v0.1 model card*. [Online]. Available: https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1. Accessed: Jul. 18, 2025.

[75] Meta AI, *Meta-Llama-3.1-8B-Instruct model card*. [Online]. Available: https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct. Accessed: Jul. 18, 2025.

[76] Meta AI, *Llama-3.2-3B-Instruct model card*. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct. Accessed: Jul. 18, 2025.

[77] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, et al., "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[78] DocsBot AI, *Llama 3.1 8B Instruct vs Llama 2 Chat 70B*. [Online]. Available: https://docsbot.ai/models/compare/llama-3-1-8b-instruct/llama-2-chat-70b. Accessed: Jul. 18, 2025.

[79] Artificial Analysis Inc., *Comparison between Llama 3.1 Instruct 8B and Mixtral 8x7B Instruct*. [Online]. Available: https://docsbot.ai/models/compare/llama-3-1-8b-instruct/mistral-8x7b-instruct. Accessed: Jul. 18, 2025.

[80] Gemma AI, "*Gemma 2-2.6B model card*," Hugging Face. [Online]. Available: https://huggingface.co/gemmaai/gemma-2-2.6b. Accessed: Jul. 18, 2025.

[81] Phi Labs, "*Phi-3.5-Mini model card,*" Hugging Face. [Online]. Available: https://huggingface.co/philschmid/phi-3.5-mini. Accessed: Jul. 18, 2025.

[82] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, *et al*., "DeepSeekMath: pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.

[83] M. Zheng, J. Pei, L. Logeswaran, M. Lee, and D. Jurgens, “When ‘a helpful assistant’ is not really helpful: Personas in system prompts do not improve performances of large language models,” in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Nov. 2024, pp. 15126–15154.

[84] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, et al., “Mixtral of experts,” *arXiv preprint arXiv:2401.04088*, 2024.

[85] Mistral AI, *Mixtral-8x7B-v0.1 model card*. [Online]. Available: https://huggingface.co/mistralai/Mixtral-8x7B-v0.1. Accessed: Jul. 18, 2025.

[86] Y. Lin, X. Ma, X. Chu, Y. Jin, Z. Yang, Y. Wang, and H. Mei, "LoRA dropout as a sparsity regularizer for overfitting control," *arXiv preprint* arXiv:2404.09610, 2024.

[87] Y. Li, C. Wei, and T. Ma, "Towards explaining the regularization effect of initial large learning rate in training neural networks," *arXiv preprint arXiv*:1907.04595, 2019.

[88] I. Loshchilov and F. Hutter, “SGDR: stochastic gradient descent with warm restarts,” *arXiv preprint arXiv:1608.03983*, 2016.

[89] A. Chen, Y. Yao, P. Y. Chen, Y. Zhang, and S. Liu, “Understanding and improving visual prompting: A label-mapping perspective,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 19133–19143.

[90] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[91] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, “Learning transferable visual models from natural language supervision,” in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, vol. 139, pp. 8748–8763, Jul. 2021.

[92] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, “Rethinking the role of demonstrations: What makes in-context learning work?,” in *Proc. Conf. Empirical Methods Natural Language Processing (EMNLP)*, Seattle, WA, USA, Nov. 2022, pp. 934–949.

[93] H. Song, M. Kim, D. Park, Y. Shin, and J.-G. Lee, "Learning from noisy labels with deep neural networks: A survey," *IEEE Trans*. Neural Netw. Learn. Syst., vol. 34, no. 11, pp. 8135–8153, 2022, doi:10.1109/TNNLS.2022.3152527.

[94] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. Smith, “Fine-tuning pretrained language models: Weight initializations, data orders, and early

stopping," in *Proc. 2021 Conf. Empirical Methods in Natural Language Processing (EMNLP)*, Punta Cana, Dominican Republic, 2021, pp. 8756–8770.

[95] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, et al., "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Transactions on Information Systems,* vol. 43, no. 2, pp. 1–55, 2025.

[96] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, et al., "Self-refine: Iterative refinement with self-feedback," in *Adv. Neural Inf. Process. Syst.*, vol. 36, pp. 46534–46594, 2023.

[97] G. Hu et al., "Alleviating hallucinations from knowledge misalignment in large language models," *Proceedings of the 2025 Annual Meeting of the Association for Computational Linguistics*, pp. 1199-1210, 2025.

[98] S. Han, J. Xu, Y. Zhao, and W. Zhou, "Towards equal opportunity fairness through adversarial learning," in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Dublin, Ireland, May 2022, pp. 2340–2353.

[99] S. Sun, A. Kaiser, and M. Dredze, "Amuro and char: analyzing the relationship between pre-training and fine-tuning of large language models," *arXiv preprint arXiv*:2408.06663, 2024.

[100] K. Chawla, A. Sahai, M. DePavia, S. Sundar, B. Miranda, E. Obbad, and S. Koyejo, "Quantifying the importance of data alignment in downstream model performance," *arXiv preprint arXiv*:2501.08496, 2025.

[101] D. Ifenthaler and D. Yau, "Utilising learning analytics for study success: Reflections on current empirical findings," *Res. Pract. Technol. Enhanc. Learn.*, vol. 15, no. 4, pp. 1–13, Dec. 2020.

[102] S. Messick, "Validity of psychological assessment: Validation of inferences from persons' responses and performances as scientific inquiry into score meaning," *Am. Psychol.*, vol. 50, no. 9, pp. 741–749, Sep. 1995.

[103] R. Lissitz and S. Samuelsen, "A suggested change in terminology and emphasis regarding validity and education," *Educ. Res.*, vol. 36, no. 8, pp. 437–448, Nov. 2007.

[104] F. Hutter, L. Kotthoff, and J. Vanschoren, "Automated machine learning: Methods, systems, challenges," in *Automated Machine Learning: The Springer Series on Challenges in Machine Learning*, Springer, 2019, pp. 1–54.

[105] E. Real, C. Liang, D. So, and Q. Le, "AutoML-Zero: Evolving machine learning algorithms from scratch," in *Proc. Int. Conf. Machine Learning (ICML)*, Virtual, Nov. 2020, pp. 8007–8019.

Appendix A

```
# ----------------------
# 3. Output Definition
# ----------------------
# Define Output Printer
class OutputPrinter:
    def __init__(self):
        self.console = Console()
    def print_case_details(self, case: Dict[str, str], attempt: int, output: str,
output_type: str) -> None:
        self.console.print(f"[bold yellow]Math Word Problem:[/bold yellow]
{case['problem']}")
        self.console.print(f"[bold yellow]Grade:[/bold yellow] {case['grade']}")
        self.console.print(f"[bold yellow]Question Type:[/bold yellow]
{case['question_type']}")
        self.console.print(f"[bold yellow]Attempt:[/bold yellow] {attempt}")
        self.console.print(f"[bold yellow]Model {output_type} Output:[/bold
yellow]\n{output}")
        self.console.print(f"[dim]{'-'*50}[/dim]")
# Define Output Formatter
class EnhancedOutputFormatter:
    def __init__(self):
        self.criteria = Config.CRITERIA
        self.norm_map = {
            re.sub(r'[^a-z]', '', c.lower()): c
            for c in Config.CRITERIA
        }
    def parse_line(self, line: str) -> tuple:
        line = line.strip()
        match = re.match(r'^[-\*]?\s*([^:]+?)\s*:\s*([0-1])', line)
        return (self.norm_map.get(re.sub(r'[^a-z]', '', match.group(1).lower()),
None),
                match.group(2)) if match else (None, None)
    def process(self, raw_output: str) -> tuple:
        output_map = {}
        missing_flags = {c: True for c in self.criteria}
        for line in raw_output.split('\n'):
            criterion, value = self.parse_line(line)
            if criterion and value in ('0', '1'):
```

```
                    output_map[criterion] = value
                    missing_flags[criterion] = False
        formatted = [f"- {c}: {output_map.get(c, 'N/A')}" for c in self.criteria]
        return '\n'.join(formatted), raw_output, missing_flags
# -------------------------------------
# 4. Safe Processor with Manual Review
# -------------------------------------
class SafeProcessor:
    def __init__(self, model, tokenizer):
        self.pipe = pipeline(
            "text-generation",
            model=model,
            tokenizer=tokenizer,
            device_map={"": "cuda:0"}
        )
        self.formatter = EnhancedOutputFormatter()
        self.console = OutputPrinter()
    def _log_review_case(self, case: dict, raw: str):
        log_entry = f"""
        [REVIEW REQUIRED] {time.ctime()}
        Problem: {case['problem'][:50]}...
        Grade: {case['grade']}
        Missing Criteria: {', '.join([c for c in Config.HIGH_RISK_CRITERIA if c
not in raw]) or 'None'}
        Raw Output: {raw[:200]}... (truncated)
        """
        with open("review_cases.log", "a") as f:
            f.write(log_entry + "-"*50 + "\n")
    def process_case(self, case: dict, template: str) -> dict:
        for attempt in range(Config.MAX_RETRIES):
            try:
                prompt = template.format(**case)
                raw_output = self.pipe(
                    prompt,
                    max_new_tokens=Config.MAX_NEW_TOKENS,
                    do_sample=False,
                    return_full_text=False
                )[0]['generated_text']
                self.console.print_case_details(case, attempt + 1, raw_output,
"Raw")
                formatted, raw, missing = self.formatter.process(raw_output)
```

```
                    self.console.print_case_details(case, attempt + 1, formatted,
"Formatted")
                    if any(missing[c] for c in Config.HIGH_RISK_CRITERIA):
                        self._log_review_case(case, raw_output)
                    return {
                        "formatted": formatted,
                        "raw": raw,
                        "missing": missing
                    }
                except Exception as e:
                    logging.error(f"Attempt {attempt+1} failed: {str(e)}")
                    torch.cuda.empty_cache()
        return {
            "formatted": None,
            "raw": "Generation failed",
            "missing": {c: True for c in Config.CRITERIA}
        }
    def process_batch(self, cases: List[Dict[str, str]], template: str) -> List[Dict]:
        outputs = []
        for case in cases:
            output = self.process_case(case, template)
            outputs.append(output)
        return outputs
```

Appendix B

```
# ----------------------------
# 4. Define Score Corrector
# ----------------------------
class ScoreCorrector:
    def __init__(self):
        # Define the mapping from evaluation criteria to correction functions
        self.criteria_rules = {
            "Unrealistic": self._unrealistic_correction,
            "Topic safety": self._safety_correction,
            "Not satisfying grade requirement":
self._grade_requirement_correction,
            "Not satisfying section requirement":
self._section_requirement_correction,
            "Not a word problem": self._word_problem_correction,
            "Trivial": self._trivial_correction,
        }
    def _extract_explanation(self, raw_output: str) -> str:
        if not raw_output:
            return ""
        patterns = [
            r'(?<=Explanation:\s)(.*?)(?=\n\w+:)',
            r'(?:Explanation|Analysis|Note|Comment|Reasoning|Evaluation):\s*(.*?
)(?:\n\n|\Z)',
            r'\/INST\]\s*(.*?)(?:\n\n|\Z)',
            r'\[INST\].*?\[/INST\](.*?)(?:\n\n|\Z)'
        ]
        for pattern in patterns:
            try:
                match = re.search(pattern, raw_output, re.DOTALL |
re.IGNORECASE)
                if match:
                    extracted = match.group(1).strip()
                    if extracted:
                        return extracted
            except Exception:
                continue
        paragraphs = [p.strip() for p in raw_output.split('\n\n') if len(p.strip()) > 30]
        return paragraphs[0] if paragraphs else raw_output[:300].strip()
```

```
    def _default_correction(self, explanation: str, current_score: str, criterion_name:
Optional[str] = None) -> str:
        """Default correction logic"""
        explanation = explanation.lower()
        positive_phrases = [
            "no issue", "no problem", "correct",
            "proper", "valid", "appropriate"
        ]
        negative_phrases = [
            "issue", "problem", "error", "flaw",
            "incorrect", "wrong", "inadequate"
        ]
        found_positive = any(term in explanation for term in positive_phrases)
        found_negative = any(term in explanation for term in negative_phrases)
        if found_positive:
            return "1"
        elif found_negative:
            return "0"
        else:
            return current_score
    def _unrealistic_correction(self, explanation: str, current_score: str,
criterion_name: Optional[str] = None) -> str:
        explanation = explanation.lower()
        if any(phrase in explanation for phrase in ["unrealistic", "implausible",
"impossible"]):
            return "0"
        if any(phrase in explanation for phrase in ["realistic", "plausible",
"possible"]):
            return "1"
        return current_score
    def _grade_requirement_correction(self, explanation: str, current_score: str,
criterion_name: Optional[str] = None) -> str:
        """Check the rule for meeting the Grade Requirement"""
        explanation = explanation.lower()
        grade_keywords = ["grade", "requirement", "level"]
        positive_phrases = ["matches", "aligns", "meets", "appropriate"]
        if any(phrase in explanation for phrase in positive_phrases):
            return "1"
        negative_phrases = ["not meeting", "trivial", "too simple"]
        if any(phrase in explanation for phrase in negative_phrases):
            return "0"
```

```
            if any(kw in explanation for kw in grade_keywords):
                negation_words = ["not", "no", "doesn't", "isn't"]
                if any(neg in explanation for neg in negation_words):
                    return "1"
                return "0"
            return current_score

    def _section_requirement_correction(self, explanation: str, current_score: str,
criterion_name: Optional[str] = None) -> str:
            """Check the rule for meeting the Section Requirement"""
            explanation = explanation.lower()
            section_keywords = ["section", "requirement"]
            if "matches" in explanation:
                return "1"
            negative_phrases = ["not specified", "does not match"]
            if any(phrase in explanation for phrase in negative_phrases):
                return "0"
            if any(kw in explanation for kw in section_keywords):
                negation_words = ["not", "no", "doesn't", "isn't"]
                if any(neg in explanation for neg in negation_words):
                    return "1"
                return "0"
            return current_score
    def _trivial_correction(self, explanation: str, current_score: str, criterion_name:
Optional[str] = None) -> str:
            explanation = explanation.lower()
            trivial_phrases = [
                "too simple", "trivial", "very easy", "basic",
                "elementary", "straightforward", "no challenge"
            ]
            non_trivial_phrases = [
                "not trivial", "non-trivial", "not simple", "not easy",
                "not basic", "challenging", "complex", "difficult",
                "requires thought", "needs reasoning"
            ]
            if any(phrase in explanation for phrase in trivial_phrases):
                return "0"
            if any(phrase in explanation for phrase in non_trivial_phrases):
                return "1"
            return current_score
```

```
    def _word_problem_correction(self, explanation: str, current_score: str,
criterion_name: Optional[str] = None) -> str:
        """Word problem detection logic: return 0 for pure mathematical formulas
(no text), return 1 if text is present"""
        explanation = explanation.lower().strip()
        if any(c.isalpha() for c in explanation):
            return "1"
        return "0"
    def _safety_correction(self, explanation: str, current_score: str, criterion_name:
Optional[str] = None) -> str:
        explanation_lower = explanation.lower()
        safety_keywords = [
            "violent", "inappropriate", "unsafe", "harmful",
            "dangerous", "kill", "death", "weapon", "abuse",
            "drug", "illegal", "suicide", "racist", "porn"
        ]
        if any(kw in explanation_lower for kw in safety_keywords):
            return "0"
        return current_score
    def correct_scores(self, parsed_output: Dict[str, str], explanation: str) -> Dict[str,
str]:
        if not isinstance(parsed_output, dict) or not isinstance(explanation, str):
            logging.error(f"Invalid input types: {type(parsed_output)},
{type(explanation)}")
            return parsed_output.copy() if isinstance(parsed_output, dict) else {}
        corrected = {}
        for criterion, score in parsed_output.items():
            try:
                if score not in ("0", "1"):
                    corrected[criterion] = score
                    continue
                corrector_func = self.criteria_rules.get(criterion,
self._default_correction)
                corrected_score = corrector_func(explanation[:1000], score,
criterion)
                if corrected_score not in ("0", "1"):
                    logging.warning(f"Invalid score {corrected_score} for
{criterion}")
                    corrected_score = score
                corrected[criterion] = corrected_score
            except Exception as e:
```

```
                    logging.error(f"Error correcting {criterion}: {str(e)}")
                    corrected[criterion] = score
        return corrected
    def _fuzzy_match_criterion(self, input_criterion: str) -> str:
        """Scoring Criteria for Fuzzy Matching"""
        best_match, best_score = None, 0
        for criterion in self.criteria_rules.keys():
            score = fuzz.ratio(input_criterion.lower(), criterion.lower())
            if score > best_score and score > 75:
                best_match, best_score = criterion, score
        return best_match or input_criterion
# -------------------------
# 5. Define Output Printer
# -------------------------
class OutputPrinter:
    def __init__(self):
        self.console = Console()
    def print_case_details(self, case: Dict[str, str], attempt: int, output: str,
output_type: str) -> None:
        """Enhanced divider handling"""
        self.console.print(f"[bold yellow]Math Word Problem:[/bold yellow]
{case['problem']}")
        self.console.print(f"[bold yellow]Grade:[/bold yellow] {case['grade']}")
        self.console.print(f"[bold yellow]Question Type:[/bold yellow]
{case['question_type']}")
        self.console.print(f"[bold yellow]Attempt:[/bold yellow] {attempt}")
        self.console.print(f"[bold yellow]Model {output_type} Output:[/bold
yellow]\n{output}")
        if "WARNING" in output:
            warnings = re.findall(r"WARNING:.*?(?=\n\n|$)", output,
re.DOTALL)
            for i, warn in enumerate(warnings, 1):
                self.console.print(f"{i}. ⚠️ {warn.strip()}")
        self.console.print(f"{'-' * 70}\n")
# -----------------------------
# 6. Define Output Formatter
# -----------------------------
class EnhancedOutputFormatter:
    def __init__(self):
        self.criteria = Config.CRITERIA
        self.norm_map = {
```

```
                re.sub(r'[^a-z]', '', c.lower()): c
                for c in Config.CRITERIA
            }
            self.simplified_map = {
                "coreference": "Co-reference issues",
                "grammar": "Grammatical errors",
                "spelling": "Misspellings",
                "incomplete": "Incomplete sentences",
                "unsolvable": "Unsolvable problems",
                "unrealistic": "Unrealistic",
                "unit": "Unit issues",
                "safety": "Topic safety",
                "grade": "Not satisfying grade requirement",
                "grade requirement": "Not satisfying grade requirement",
                "section": "Not satisfying section requirement",
                "section requirement": "Not satisfying section requirement",
                "trivial": "Trivial",
                "word problem": "Not a word problem",
                "wordproblem": "Not a word problem",
                "narrative": "Not a word problem"
            }
        def parse_line(self, line: str) -> Tuple[Optional[str], Optional[str]]:
            line = line.strip()
            line = re.sub(r'^\d+\.\s*', '', line)
            match = re.match(r'^[-\*]?\s*([^:]+?)\s*:\s*([0-1])\s*$', line)
            if not match:
                match = re.match(r'^[-\*]?\s*([^:]+?)\s*:\s*([0-1])', line)
            if match:
                criterion_name = match.group(1).strip()
                value = match.group(2)
                norm_name = self.norm_map.get(re.sub(r'[^a-z]', '',
criterion_name.lower()))
                if norm_name:
                    return (norm_name, value)
                for key, full_name in self.simplified_map.items():
                    if re.search(rf'\b{key}\b', criterion_name.lower()):
                        return (full_name, value)
            return (None, None)
        def process(self, raw_output: str) -> Tuple[str, str, Dict[str, bool]]:
            output_map = {}
            missing_flags = {c: True for c in self.criteria}
```

```
        lines = raw_output.split('\n')[:12]
        for line in lines:
            criterion, value = self.parse_line(line)
            if criterion and value in ('0', '1'):
                output_map[criterion] = value
                missing_flags[criterion] = False
        formatted = [f"- {c}: {output_map.get(c, 'N/A')}" for c in self.criteria]
        return '\n'.join(formatted), raw_output, missing_flags
# ------------------------
# 7. Utility Function
# ------------------------
def escape_braces(text: str) -> str:
    """Escape curly braces in text to prevent format errors"""
    return text.replace("{", "{{").replace("}", "}}")
# ---------------------------
# 8. Define Safe Processor
# ---------------------------
class SafeProcessor:
    def __init__(self, model, tokenizer):
        self.pipe = pipeline(
            "text-generation",
            model=model,
            tokenizer=tokenizer,
            device_map={"": "cuda:0"} if torch.cuda.is_available() else None,
            max_new_tokens=Config.MAX_NEW_TOKENS
        )
        self.formatter = EnhancedOutputFormatter()
        self.console = OutputPrinter()
        self.corrector = ScoreCorrector()
    def _log_review_case(self, case: Dict[str, str], raw_output: str, parsed_output:
Dict[str, str]) -> None:
        """Log cases with missing high-risk criteria"""
        missing_high_risk = [c for c in Config.HIGH_RISK_CRITERIA if c not in
parsed_output]
        if missing_high_risk:
            log_entry = f"""
            [REVIEW REQUIRED] {time.ctime()}
            Problem: {case['problem'][:50]}...
            Grade: {case['grade']}
            Missing Criteria: {', '.join(missing_high_risk)}
            Raw Output: {raw_output[:200]}... (truncated)
```

```
            """
            os.makedirs(os.path.dirname(Config.LOG_FILE), exist_ok=True)
            with open(Config.LOG_FILE, "a") as f:
                f.write(log_entry + "-"*50 + "\n")
    def _extract_explanation(self, raw_output: str) -> str:
        if not raw_output:
            return ""
        patterns = [
            r'(?<=Explanation:\s)(.*?)(?=\n\w+:)',
            r'(?:Explanation|Analysis|Note|Comment|Reasoning|Evaluation):\s*(.*?
)(?:\n\n|\Z)',
            r'\/INST\]\s*(.*?)(?:\n\n|\Z)',
            r'\[INST\].*?\[/INST\](.*?)(?:\n\n|\Z)'
        ]
        for pattern in patterns:
            try:
                match = re.search(pattern, raw_output, re.DOTALL |
re.IGNORECASE)
                if match:
                    extracted = match.group(1).strip()
                    if extracted:
                        return extracted
            except Exception:
                continue
        paragraphs = [p.strip() for p in raw_output.split('\n\n') if len(p.strip()) > 30]
        return paragraphs[0] if paragraphs else raw_output[:300].strip()
    def _parse_raw_output(self, raw_output: str) -> Dict[str, str]:
        """Parse raw output into a dictionary with precise error handling"""
        parsed = {}
        for line in raw_output.split('\n'):
            try:
                criterion, value = self.formatter.parse_line(line)
                if criterion and value in ('0', '1'):
                    parsed[criterion] = value
            except Exception as e:
                logging.debug(f"Failed to parse line: {line[:50]}... Error: {str(e)}")
        return parsed
    def _validate_scoring(self, raw_output: str, parsed: Dict[str, str]) -> bool:
        """Validate scoring consistency with explanation"""
        explanation = self._extract_explanation(raw_output)
        if not explanation:
```

```
                return True
            corrected = self.corrector.correct_scores(parsed, explanation)
            conflicts = [criterion for criterion, score in parsed.items() if score !=
corrected.get(criterion, score)]
            if conflicts:
                logging.warning(f"⚠️ Scoring conflict: {',
'.join(conflicts)}\nExplanation: {explanation[:200]}...")
                return False
            return True
        def _resolve_final_scores(self, attempts: List[Dict]) -> Dict[str, str]:
            """Main Logic for Scoring Interpretation"""
            final_scores = {}
            # 1. Check for consistent scores across two attempts
            if len(attempts) >= 2:
                for crit in self.formatter.criteria:
                    try:
                        val1 = attempts[0]['parsed'].get(crit)
                        val2 = attempts[1]['parsed'].get(crit)
                        if val1 == val2 and val1 in ("0", "1"):
                            final_scores[crit] = val1
                    except Exception as e:
                        logging.warning(f"Error comparing {crit}: {str(e)}")
            # 2. Use explanation analysis for uncertain scores
            for crit in self.formatter.criteria:
                if crit in final_scores:
                    continue
                try:
                    explanations = [
                        a.get('explanation', '') for a in attempts
                        if a.get('explanation') and isinstance(a.get('parsed'), dict) and
crit in a['parsed']
                    ]
                    if explanations:
                        correction_func = self.corrector.criteria_rules.get(
                            crit, self.corrector._default_correction
                        )
                        last_score = next(
                            (a['parsed'][crit] for a in reversed(attempts)
                            if isinstance(a.get('parsed'), dict) and crit in a['parsed']),
                            "N/A"
                        )
```

```
                    final_scores[crit] = correction_func(
                        " ".join(ex for ex in explanations if ex),
                        last_score,
                        crit
                    )
            except Exception as e:
                logging.warning(f"Error analyzing {crit}: {str(e)}")
                final_scores[crit] = "N/A"
        # 3. Fill in the remaining uncertain scores
        for crit in self.formatter.criteria:
            if crit not in final_scores:
                try:
                    final_scores[crit] = next(
                        (a['parsed'][crit] for a in reversed(attempts)
                        if isinstance(a.get('parsed'), dict) and crit in a['parsed']),
                        "N/A"
                    )
                except Exception:
                    final_scores[crit] = "N/A"
        return final_scores
    def _merge_attempts(self, attempts: List[Dict[str, Any]]) -> Dict[str, Any]:
        """Consolidating Scores via _resolve_final_scores"""
        if not attempts or not all(isinstance(a, dict) for a in attempts):
            return self._create_error_result("Invalid attempts input")
        try:
            corrected_scores = self._resolve_final_scores(attempts)
            formatted_lines = []
            missing_flags = {}
            for criterion in self.formatter.criteria:
                score = corrected_scores.get(criterion, "N/A")
                formatted_lines.append(f"- {criterion}: {score}")
                missing_flags[criterion] = (score == "N/A")
            return {
                "formatted": "\n".join(formatted_lines),
                "raw": "\n\n".join(a.get('raw', '') for a in attempts if a.get('raw')),
                "missing": missing_flags,
                "parsed_dict": corrected_scores
            }
        except Exception as e:
            logging.error(f"Merge attempts failed: {str(e)}")
            return self._create_error_result(str(e))
```

```
    def _create_error_result(self, error_msg: str) -> Dict[str, Any]:
        """Create Error Result Structure"""
        return {
            "formatted": f"ERROR: {error_msg}",
            "raw": f"ERROR: {error_msg}",
            "missing": {c: True for c in self.formatter.criteria},
            "parsed_dict": {c: "ERROR" for c in self.formatter.criteria}
        }
    def process_case(self, case: Dict[str, str], template: str) -> Dict[str, Any]:
        with manage_resources(self.pipe):
            attempts = []
            for attempt_num in range(Config.MAX_ATTEMPTS):
                try:
                    current_template = template
                    if attempt_num > 0:
                        current_template = template + (
                            "\nIMPORTANT: In your previous response, you
described problems "
                            "but gave score 1. This is invalid. Remember: if
you mention "
                            "any issue in explanation, you MUST score 0 for
that criterion."
                        )
                    safe_problem = escape_braces(case["problem"])
                    prompt = current_template.format(
                        grade=escape_braces(case["grade"]),
                        question_type=escape_braces(case["question_type"]),
                        problem=escape_braces(safe_problem)
                    )
                    raw_output = self.pipe(
                        prompt,
                        max_new_tokens=Config.MAX_NEW_TOKENS,
                        do_sample=False,
                        return_full_text=False
                    )[0]['generated_text']
                    self.console.print_case_details(case, attempt_num + 1,
raw_output, "Raw")
                    explanation = self._extract_explanation(raw_output)
                    parsed = self._parse_raw_output(raw_output)
                    attempts.append({
                        'raw': raw_output,
```

```
                        'parsed': parsed,
                        'explanation': explanation if explanation else ""
                    })
                    if attempt_num < Config.MAX_ATTEMPTS - 1:
                        time.sleep(Config.RETRY_DELAY)
                except Exception as e:
                    logging.error(f"Attempt {attempt_num+1} failed: {str(e)}")
                    attempts.append({
                        'raw': "",
                        'parsed': {},
                        'explanation': ""
                    })
                    if attempt_num < Config.MAX_ATTEMPTS - 1:
                        time.sleep(Config.RETRY_DELAY)
            merged_result = self._merge_attempts(attempts)
            self.console.print_case_details(case, "Final",
merged_result["formatted"], "Formatted")
            if any(merged_result["missing"][c] for c in
Config.HIGH_RISK_CRITERIA):
                self._log_review_case(
                    case,
                    "\n\n".join(a['raw'] for a in attempts if a.get('raw')),
                    merged_result["parsed_dict"]
                )
            return merged_result
    def process_batch(self, cases: List[Dict[str, str]], template: str) -> List[Dict[str,
Any]]:
        """Batch process test cases"""
        results = []
        for case in cases:
            try:
                result = self.process_case(case, template)
                results.append(result)
            except Exception as e:
                logging.exception(f"Failed to process case: {case}")
                results.append(self._create_error_result(str(e)))
        return results
# -------------------------
# 9. Prompt Template Utilities
# -------------------------
def make_template(base_template: str, example: Optional[str] = None) -> str:
```

```
    """Create prompt template with optional example"""
    if not isinstance(base_template, str):
        raise ValueError("base_template must be string")
    if not example:
        return base_template
    plural = "s" if '\n' in example else ""
    return base_template.replace(
        "[INST]",
        f"[INST]\nUse the following example{plural} to guide your
scoring:\n{example}\n"
    )
```