Ask-Before-Detection: Identifying and Mitigating Conformity Bias in LLM-Powered Error Detector for Math Word Problem Solutions

Hang Li[1], Tianlong Xu[2], Kaiqi Yang[1], Yucheng Chu[1],
Yanling Chen[1], Yichi Song[3], Qingsong Wen[2], Hui Liu[1]*
[1]Michigan State University, [2]Squirrel Ai Learning, [3]Carleton College,
{lihang4,kqyang,chuyuch2,chenya67,liuhui7}@msu.edu
{tianlongxu,qingsongwen}@squirrelai.com, {songc2}@carleton.edu

Abstract

The rise of large language models (LLMs) offers new opportunities for automatic error detection in education, particularly for math word problems (MWPs). While prior studies demonstrate the promise of LLMs as error detectors, they overlook the presence of multiple valid solutions for a single MWP. Our preliminary analysis reveals a significant performance gap between conventional and alternative solutions in MWPs, a phenomenon we term conformity bias in this work. To mitigate this bias, we introduce the Ask-Before-Detect (AskBD) framework, which generates adaptive reference solutions using LLMs to enhance error detection. Experiments on 200 examples of GSM8K show that AskBD effectively mitigates bias and improves performance, especially when combined with reasoning-enhancing techniques like chain-of-thought prompting. The code and data are available at https://github.com/dse-ai-edu/AskBD.

1 Introduction

Automatic Error Detection (AED) has been a prominent research topic in education over the past few decades (Leacock et al., 2014). Supported by rapid advancements in natural language processing (NLP) technologies, particularly in language modeling (Min et al., 2023), AED research has achieved notable success in language education (Huang et al., 2023). The recent emergence of large language models (LLMs) presents new opportunities for AED studies. Leveraging their exceptional capabilities in logical reasoning (Pan et al., 2023), LLMs have become promising tools helping the quick development of AED in more challenging scenarios including programming (Messer et al., 2024) and mathematics learning (Jiang et al., 2024; Yen and Hsu, 2023). Recent studies have introduced benchmark datasets to demonstrate the potential of LLMs in AED across diverse domains (Yan et al., 2024). Moreover, due to the reasoning-intensive nature of error detection in mathematical problems, recent research has employed AED tasks on math word problems (MWPs) to evaluate the comparative reasoning capabilities of LLMs (Li et al., 2024). Studies (Zhou et al., 2024) have indicated that identifying errors in MWPs, rather than generating correct solutions, serves as a more effective metric for assessing differences in the reasoning capabilities of LLMs. In this paper, we explore AED for MWPs. Specifically, building on prior studies, we define the AED task as identifying both the erroneous step and its error category from a given input pair consisting of a question and its solution. It is important to note that a correct result requires accurate identification of both the error step and the error category.

While previous studies (Li et al., 2024; Zhou et al., 2024) have explored various methods for evaluating the error detection capabilities of LLMs on MWPs, these approaches predominantly focus on generating erroneous solutions based solely on the conventional solutions provided in the dataset. In practice, however, a single MWP can have multiple valid solutions, leaving the performance of LLMs on alternative solutions largely unexplored. In Figure 1, we present an illustrative example where both the conventional and alternative solutions are submitted to an LLM-powered error detector, yet only the conventional solution receives the correct detection result. Motivated by this observation, we propose an automatic method for generating alternative solutions and evaluate the behavior of LLM-powered error detectors on 200 pairs of conventional and alternative solutions. Our preliminary results in section 2.3.1 reveal an average 7% detecting accuracy performance gap between conventional and alternative solutions, with even advanced closed-source models exhibiting the same limitation. These findings suggest that current

*Corresponding author.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1597–1609

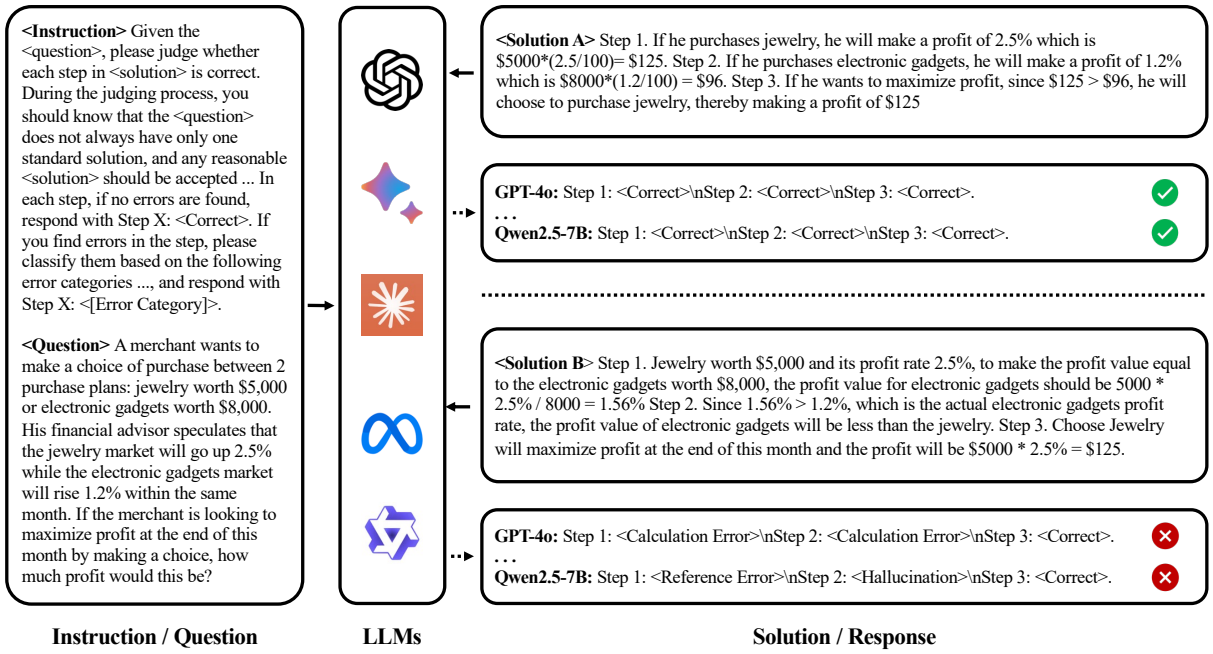


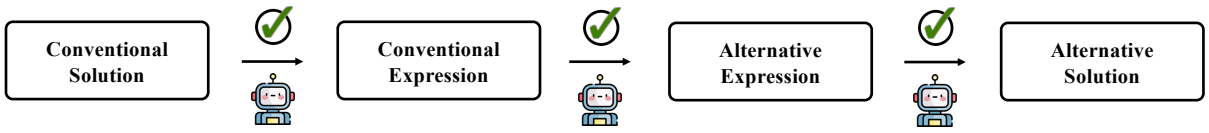
Figure 1: An illustration of error detection in MWP solutions: <Solution A> represents the conventional solution, which achieves accurate error detection with LLM-powered error detectors. In contrast, <Solution B>, while also correct, encounters erroneous error detection results across all LLM-powered error detectors.

LLM-based error detectors display a pronounced **conformity bias**, favoring a specific solution format while rejecting others. This bias is particularly concerning in educational contexts, as it discourages students from exploring diverse problem-solving approaches and stifles creativity.

To investigate the underlying causes of conformity bias and develop effective strategies to mitigate it, we conduct further preliminary studies in Section 2.3, which examines the common patterns in the behavior of LLM-powered error detectors when evaluating diverse solutions. Our findings reveal that error detection accuracy is closely correlated with the likelihood scores assigned by LLMs to solutions, with higher likelihood scores corresponding to improved detection performance. Since alternative solutions typically receive lower likelihood scores compared to conventional ones, conformity bias naturally emerges. This observation points to a potential remedy: adjusting the likelihood scores of solutions. However, this approach faces two significant challenges. First, fine-tuning advanced models requires high-quality datasets and incurs substantial costs. Second, while fine-tuning may improve likelihood scores for samples within the training dataset, its generalizability to novel solutions remains uncertain. During our investigation into the impact of introducing a reference answer during error detection, we observed that conformity bias is significantly reduced across all LLMs. This finding inspires us to leverage reference answers as a viable strategy for mitigating bias. However, uniformly providing a standard reference answer for every solution is suboptimal in practice. Misalignment between the reasoning behind different solutions and the reference answer can sometimes degrade final detection performance. To address this, we propose the Ask-Before-Detection (AskBD) framework, which generates adaptive reference answers through step-by-step question-answering techniques. By leveraging the strong problem-solving capabilities of LLMs and employing a decomposed, question-guided approach, the reference answers can be generated with high accuracy, even using less capable models. Incorporating these generated reference solutions significantly mitigates conformity bias in LLM-powered error detectors for MWP solutions. Furthermore, the adaptability of the generated references enhances the overall performance of LLM-powered error detector across both conventional and alternative solutions.

2 Preliminary Study

In this section, we present our preliminary studies aimed at identifying and understanding the presence of **conformity bias** in LLM-powered error detectors. Specifically, we begin by describing our



<Conventional Solution>: Step 1. Candidate A received 20% of 100 voters, so **100 x 20/100 = 20** votes. Step 2. Candidate B received 50% more than candidate A, which is **20 x 50/100 = 10** more votes. Step 3. Therefore, candidate B received **20 (votes from A) + 10 (additional votes) = 30 votes**. Step 4. Together, candidate A and B got **20 + 30 = 50** votes. Step 5. Candidate C received the remaining votes, which is **100 (total voters) - 50 (votes from A and B) = 50 votes.**

<Conventional Expression>: 100 - **((**100 × 0.2**)** + **((**100 × 0.2**)** + **((**100 × 0.2**)** × 0.5**))**

<Alternative Expression>: 100 × **(**1 - **(**0.2 + **(**0.2 × **(**1+ 0.5**))))**

<Alternative Solution>: Step 1. Calculate the ratio of the votes Candidate B received to those Candidate A received: **1 + 0.5 = 1.5**. Step 2. Calculate the fraction of total votes Candidate B received: **0.2 × 1.5 = 0.3**. Step 3. Calculate the combined fraction of votes received by Candidates A and B: **0.2 + 0.3 = 0.5**. Step 4. Calculate the fraction of total votes Candidate C received: **1 - 0.5 = 0.5**. Step 5. Calculate the number of votes Candidate C received: **100 × 0.5 = 50 votes**.

Figure 2: The ASP pipeline to generate permuted solution. The corresponding relationships between the calculations in each step and the parts enclosed by different parentheses in expression are highlighted using matching colors.

automated method for preparing an error detection dataset featured with paired conventional and alternative solutions. Then, we analyze the relationship between likelihood scores and the behavior patterns of LLM-powered error detectors. Finally, we present our observations on how incorporating reference solution text influences the performance and behavior of error detectors.

2.1 Automatic Solution Permutation

Building a high-quality alternative solution dataset is critical to our preliminary study, as low-quality alternatives, such as simple semantic paraphrases of conventional solutions, fail to effectively expose the "conformity bias" in LLM-powered error detectors. During our initial exploration, we observed that directly using simple prompts to query LLMs for automatically generating alternative solutions presents significant challenges. Specifically, LLMs often produce paraphrased versions of conventional solutions unless detailed and specific instructions about the solving strategy are provided during the generation process. To address this, we propose the Automatic Solution Permutation (ASP) method, which leverages the correspondence between conventional solutions and their solving expressions. Using these expressions as specific instructions helps LLMs move beyond paraphrasing behavior, enabling the generation of high-quality alternative solutions.

The ASP method operates in three stages: Extract, Permute, and Explain. At each stage, LLMs are prompted to execute specific tasks independently. In the Extract stage, ASP encapsulates the steps of a conventional solution into a single mathematical expression. To ensure accuracy, these expressions are executed, and any that fail to produce correct answer values are discarded. In the Permute stage, ASP generates new expressions by applying operations such as factorization, distribution, and order rearrangement, which transform the expressions while preserving their mathematical equivalence. A similar filtering process is applied to these permuted expressions to ensure correctness. Finally, in the Explain stage, the permuted expressions are provided to LLMs to guide the generation of high-quality alternative solutions. By instructing the LLMs to interpret the brackets within each expression as distinct steps, the ASP method produces detailed, step-by-step alternative solutions. Figure 2 illustrates the ASP pipeline and provides an example of paired conventional and alternative solutions alongside their corresponding solving expressions.

In our study, we use GPT-4o as the backbone LLM for each stage of the ASP method. To begin, we randomly sample 200 question-and-answer pairs from the test split of GSM8K (Cobbe et al., 2021) and use them to construct our conventional dataset, $\mathcal{D}$. For each sample in $\mathcal{D}$, we apply ASP three times to generate three candidate permuted solutions for each conventional solution. Subsequently, a graduate student from the education department reviews the quality of all the generated alternative solutions and selects the highest-quality alternative solution for each convention solution.

These selected solutions are then compiled to form the alternative dataset, $\mathcal{D}'$.

2.2 Erroneous Solution Generation

After preparing the alternative solution dataset, the next step is to generate erroneous solutions. Building on prior work (Li et al., 2024), which categorize common errors in solutions to MWPs into categories, we choose four representative error types that commonly encountered in real-world error grading scenarios: calculation errors ($\mathcal{E}_C$), reference errors ($\mathcal{E}_R$), missing steps ($\mathcal{E}_M$), and hallucinations ($\mathcal{E}_H$), for our study. It is worth noting that this study specifically aims to explore conformity bias, and therefore, we do not include all possible error types. To minimize the risk of experimental noise caused by ambiguous definitions, we defined these error types in a straightforward and easily distinguishable manner. Detailed descriptions of each error type are provided in Table 8 in Appendix C. To simulate erroneous solutions, we injected these errors into correct solutions using a generation strategy inspired by prior work (Li et al., 2024). During the injection process, the error type was controlled through a hyper-parameter, while the specific error location (error step number) was determined randomly. This approach enables controlled testing of the AED's ability to handle and identify various error scenarios effectively. For each example in $\mathcal{D}$ and $\mathcal{D}'$, we generated four corresponding erroneous solutions, each associated with one of the four error types. This process yielded a total of 2,000 examples, which were prepared for subsequent analysis.

2.3 Analysis and Findings

Before delving into the details about our analysis and findings, we first introduce the evaluation metric used for our following analyses. Specifically, since the locations and categories of injected errors are automatically labeled during the error injection process for each solution, we task the LLM-powered error detector with identifying both the error locations and their types. The evaluation metric is the identification accuracy across both correct and erroneous solutions.

2.3.1 Conformity Bias Identification

To identify conformity bias, we employ a widely-used LLM-powered error detection approach, leveraging prompt engineering techniques outlined in previous studies (Li et al., 2024). In addition, the instruction text informs the LLMs that alternative solutions to the given question exist and emphasizes that all reasonable solutions should be accepted. To minimize variability in performance due to ambiguity in error categories, we provide explicit definitions for each error category within the prompt text, ensuring clarity for the LLMs. The prompt used for the error detection task is illustrated in Figure 5 in Appendix D.

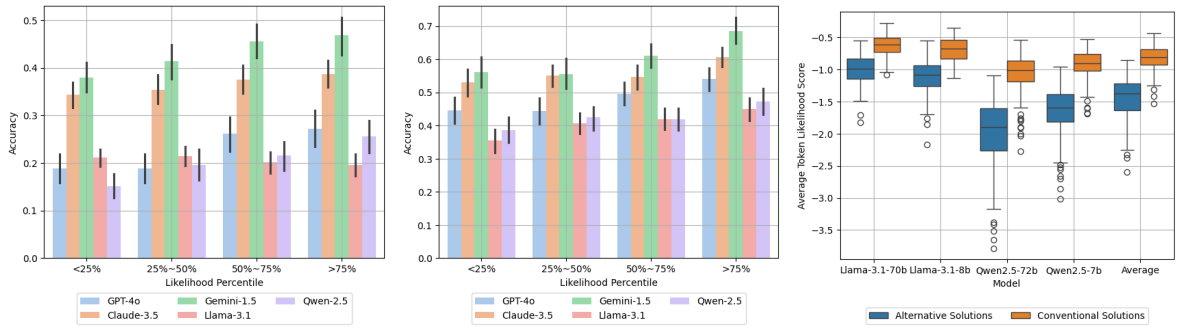
Table 1: Error detection performance on ordinary ($\mathcal{D}$) and alternative ($\mathcal{D}'$) solutions. The performance gap is calculated as $\Delta = |\mathcal{D}' - \mathcal{D}|$. Results marked with * indicate statistical significance based on student's t-test.

Model	Small			Large		
	$\mathcal{D}$	$\mathcal{D}'$	Δ	$\mathcal{D}$	$\mathcal{D}'$	Δ
GPT-4o	27.2	18.4	8.8*	52.9	43.4	9.5*
Claude-3.5	38.2	34.7	3.5*	59.9	52.7	7.2*
Gemini-1.5	46.4	39.5	6.9*	65.2	55.6	9.6*
Llama-3.1	20.2	20.9	0.7	44.3	37.2	7.2*
Qwen-2.5	24.7	16.3	8.4*	46.3	38.8	7.5*

To comprehensively analyze the conformity bias exhibited by various LLMs, we conducted experiments with 10 representative models. These include three closed-source series with their large (small) versions (e.g., GPT-4o (Mini) (Bubeck et al., 2023), Gemini-1.5-Pro (Flash) (Team et al., 2023), Claude-3.5-Sonnet (Haiku) (Anthropic, 2024)) and two open-source series with their large (small) counterparts (e.g., Llama-3.1-70B (8B) (Touvron et al., 2023) and Qwen-2.5-72B (7B) (Yang et al., 2024)). Table 1 presents a comparison of the average error detection accuracy across both the conventional solution $\mathcal{D}$ and the alternative solution $\mathcal{D}'$ dataset. The results clearly demonstrate a consistent performance gap between the two datasets, confirming the presence of conformity bias in LLM-based error detection tasks.

2.3.2 Solution Likelihood Score Analysis

To investigate the underlying causes of conformity bias, we first chose to use the log-likelihood score, denoted as $\log L_\theta(s|q)$, returned by the LLM for a given solution text s to the question text q, as an indicator. This likelihood score is utilized as it reflects the LLM's confidence in the solution text relative to the question text. If a solution known to be correct receives a low confidence score from the LLM, it suggests that the LLM does not fully understand the solution. Conversely, if the correct solution receives a high score, it indicates that the



(a) Small models (b) Large models (c) Likelihood distributions

Figure 3: Average error detection accuracy across samples grouped by the 25th, 50th, and 75th percentiles of I_u.

LLM is proficient with the solution. The detailed calculation method is presented below:

$$\log L_\theta(s|q) = \sum_{i=1}^{|s|} \log L_\theta(s_i|[q, s_{1:(i-1)}]) \quad (1)$$

where θ represents the parameters of the LLM, $[\cdot,\cdot]$ is text concatenation, s_i is the i-th token of the solution text. However, directly comparing the likelihood scores calculated by Equation 1 for solutions of varying lengths is still problematic, as the likelihood score is inversely proportional to the length of s. In other words, shorter solutions with fewer tokens tend to have higher scores than longer ones. To address this issue, we finally adopt the average token log-likelihood score for our analysis in subsequent studies.

$$\log \bar{L}_\theta(s|q) = \frac{\log L_\theta(s|q)}{|s|} \quad (2)$$

In Figure 3b and Figure 3a, we present the average error detection accuracy across different likelihood score groups. Specifically, given the likelihood score to both convention and alternative solutions, we group them based on their likelihood score percentiles. For simplicity, we use the four quarters in our experiment. It is important to note that, since the likelihood scores of closed-source models are unavailable, we use the average scores of all open-source LLMs as a pseudo-indicator for this analysis. From the figure, we observe that the larger quarter groups with higher indicator values exhibit a clear advantage over those with the smaller quarter ones. In addition, we plot the likelihood score distribution comparisons between the solution from $\mathcal{D}$ and $\mathcal{D}'$ in Figure 3c. From these plot, we can draw a clear conclusion that the conformity bias in current LLM to error detection tasks is caused by its decreased understanding to those alternative solution.

2.3.3 Reference-based Detection Findings

Directly improving the likelihood scores of alternative solutions poses inherent challenges. Strategies like fine-tuning large language models (LLMs) primarily improve likelihood scores for training samples, but their effectiveness on unseen alternative solutions remains unpredictable. Building on prior work (Daheim et al., 2024), which demonstrated that introducing reference answers during error detection enhances performance on conventional solutions, we extend this approach to alternative solutions. It is important to note that, in real-world error detection scenarios, reference answers are not always available. Even when they are, conventional solutions are more commonly provided. Take this into consideration, we conducted experiments comparing two reference-based detection setups: (1) uniformly using conventional solutions as references and (2) adaptively using corresponding solutions as references. The detailed results are presented in Table 3 and Table 2, respectively.

Table 2: Error detection performance w/ using corresponding solution as reference solution for both ordinary ($\mathcal{D}$) and alternative ($\mathcal{D}'$) solutions. The performance gap is calculated by $\Delta = |\mathcal{D}' - \mathcal{D}|$. Results marked with * indicate statistical significance based on student's t-test.

Model	Small			Large		
	$\mathcal{D}$	$\mathcal{D}'$	Δ	$\mathcal{D}$	$\mathcal{D}'$	Δ
GPT-4o	60.0	56.5	3.5*	75.5	73.8	1.7*
Claude-3.5	60.4	57.9	2.5*	84.0	81.6	2.4*
Gemini-1.5	69.7	66.7	3.0*	85.3	83.7	1.6*
Llama-3.1	34.6	33.7	0.9	77.5	79.4	1.9*
Qwen-2.5	54.4	51.2	3.2*	59.3	60.8	1.5*

Table 3: Error detection performance w/ using convention solution as reference for both ordinary ($\mathcal{D}$) and alternative ($\mathcal{D}'$) solutions. The performance gap is calculated by $\Delta = |\mathcal{D}' - \mathcal{D}|$. Results marked with * indicate statistical significance based on student's t-test.

Model	Small			Large		
	$\mathcal{D}$	$\mathcal{D}'$	Δ	$\mathcal{D}$	$\mathcal{D}'$	Δ
GPT-4o	60.0	32.0	28.0*	75.5	53.3	22.2*
Claude-3.5	60.4	38.9	21.5*	84.0	59.4	24.6*
Gemini-1.5	69.7	50.8	18.9*	85.3	67.7	17.6*
Llama-3.1	34.6	20.5	14.1*	77.5	48.8	28.7*
Qwen-2.5	54.4	22.2	32.2*	59.3	43.5	15.8*

By analyzing the results across Table 1, Table 3, and Table 2, it is evident that introducing reference solutions improves error detection accuracy for both datasets, $\mathcal{D}$ and $\mathcal{D}'$. However, the choice of reference solution significantly impacts performance. While introducing corresponding reference solutions effectively mitigates bias, uniformly using conventional solutions tends to amplify it. This contrast highlights the critical importance of selecting appropriate reference solutions to enhance error detection in alternative scenarios.

3 Method

The findings in Section 2.3.3 suggest that incorporating a reference solution during the detection process is an effective approach to addressing conformity bias. However, the choice of the reference solution plays a critical role. Building on this insight, we propose the Ask-Before-Detection (AskBD) framework, which leverages the generative capabilities of large language models (LLMs) to create adaptive reference solutions tailored to each provided solution during the grading process. The AskBD offers several advantages. First, it utilizes the inherent problem-solving capabilities of LLMs rather than relying on fine-tuning, which makes AskBD easily extendable to various solutions. Second, by adaptively generating reference solutions, the framework ensures that these references are well-aligned with the given answers, significantly reducing the risk of mismatches that could amplify bias. Furthermore, the AskBD is orthogonal to other reasoning techniques, such as chain-of-thought (CoT) (Wei et al., 2022a), meaning that it can complement and enhance their performance. By integrating AskBD with these algorithms, the error detection capabilities of LLMs can be further improved. The overall structure of the AskBD is illustrated in Figure 4.

Algorithm 1: Ask-Before-Detection

Input: Question text q, solution text s, large lanauge model f_θ, prompt text for each component $\mathcal{P}_{(\cdot)}$

1 Condition and question extractor (CQE): Extract condition information q_c and inquiry text q_i from the question text q. $(q_c, q_i) = f_\theta([\mathcal{P}_{cqe}, q])$;

2 Solution Step Inquirer (SSI): Convert solution text s into step-wise question list text Q and append inquiry text q_i at the end. $Q = [f_\theta([\mathcal{P}_{ssi}, s]), q_i]$;

3 Step Question Responder (SQR): Generate reference solution r by summarizing the answers to each question in Q using condition text q_c. $r = f_\theta([\mathcal{P}_{sqr}, q_c, Q])$;

4 Reference-Enhanced Grader (REG): Generate the error detection result (error location y_s, error type y_e) based on the input (q, s, r). $y = f_\theta([\mathcal{P}_{reg}, q, s, r])$;

5 **return** y_s, y_e

AskBD consists of four components, executed sequentially to generate an adaptive reference answer tailored to the input solution. First, the Condition and Question Extractor (CQE) processes the input question text, q, by extracting two key elements from the original question stem: condition information and inquiry text. The condition information, q_c, represents the known facts or context provided in the question, while the inquiry text, q_i, specifies the task or problem posed by the question. Then, the Solution Step Inquirer (SSI) focuses on generating step-specific questions based on the conclusions of each step in the provided solution, s. To improve the stability of the generated results, the SSI first summarizes the conclusion of each step before formulating corresponding questions. These step-specific questions are compiled into a question list, Q, with the inquiry text q_i always appended to the end of the list to ensure that the original question's task is addressed in the generated reference answer. Next, after both the condition text q_c and the question list Q are prepared, the Step Question Responder (SQR) generates responses to each question in Q and reorganizes them into a referenced solution, r. Finally, the Reference-Enhanced Grader (REG) uses the referenced solution r along with the inputs q and s to produce the final grading

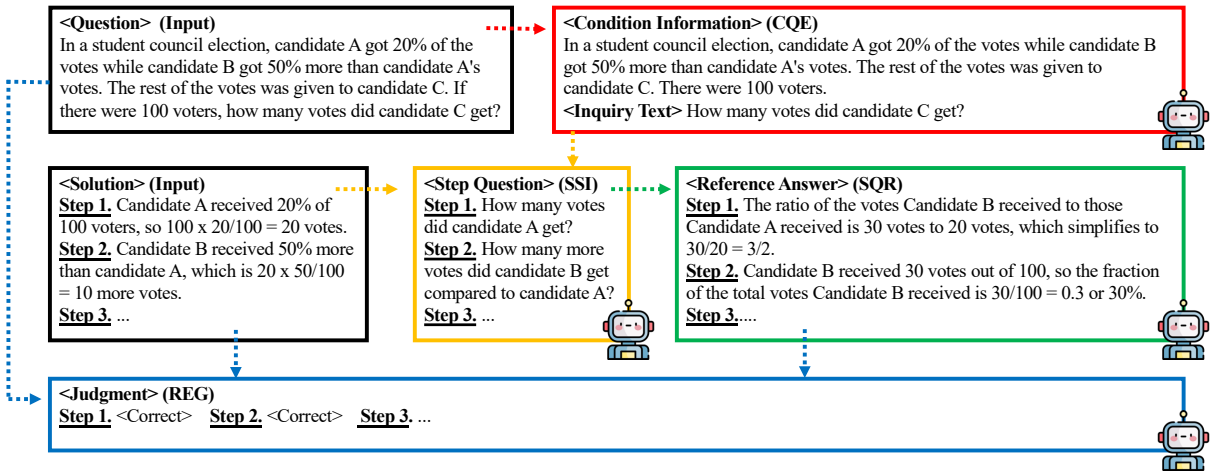


Figure 4: An overview of AskBG framework where steps are marked with colors. <Question> and <Solution> are the inputs and <Reference Answer> is generated by the framework, which used to generate the final response.

results. More details can be found in Algorithm 1.

4 Experiment

In this section, we present experiments to validate the effectiveness of AskBD. Experiments are designed to address the following research questions:

- **RQ1:** Does AskBD help mitigate conformity bias in error detection?
- **RQ2:** What additional performance advantages does AskBD provide?
- **RQ3:** How compatible is AskBD with other reasoning techniques, such as chain-of-thought?

4.1 Experimental Settings

To answer above research questions, we use dataset generated during the preliminary study introduced in Section 2. The detailed statistics of the dataset are shown as Table 6 in Appendix A. To evaluate the generalizability of AskBD, we implement it using the same 10 LLMs described in Section 2, along with two math-specific models, including Qwen2.5-Math. Detailed information about each model is provided in Table 7 in Appendix B. Additionally, we incorporate the CoT reasoning approach into the prompts to assess its compatibility with AskBD, the prompt is shown as Figure 6 in Appendix E. Each experiment is conducted using three different random seeds, and we report the mean error detection accuracy in the results. As this is the first study to systematically examine the occurrence of conformity bias in LLM-powered error detection for MWP solutions, the results from the preliminary study sections serve as one baseline. Furthermore, CoT, being an orthogonal method to AskBD, is treated as another baseline for comparison.

4.2 Result and Discussion

In Table 4, we present the comparison between baseline methods and AskDB over both the conventional solutions $\mathcal{D}$ and alternative solution $\mathcal{D}'$. To address RQ1, we analyze the values in the Δ columns between $\mathcal{M}_2$ and $\mathcal{M}_0$. The table clearly demonstrates that AskDB is effective in mitigating conformity bias in error detection results for advanced versions of LLMs. However, for base LLMs, the benefits of naively applying AskDB are less evident. Among the five LLM frameworks, only Gemini exhibits a reduced performance gap between $\mathcal{D}$ and $\mathcal{D}'$. We attribute this to the relatively weaker reasoning capabilities of base models. With the naive instruction prompt, these models fail to fully leverage the valuable information provided by the reference solutions, thereby limiting the effectiveness of AskDB in these cases. Comparing $\mathcal{M}_1$ with $\mathcal{M}_2$, we observe that the CoT prompt strategy also helps mitigate conformity bias in LLM-powered error detectors. Nevertheless, in most advanced models, AskDB consistently outperforms CoT in narrowing the gap between conventional and alternative solutions.

To answer RQ2, we compare $\mathcal{M}_2$ with $\mathcal{M}_0$ in the $\mathcal{D}$ and $\mathcal{D}'$ columns. The results indicate a consistent improvement in error detection accuracy after adopting the AskDB framework. This suggests that AskDB not only helps reduce the performance gap between conventional and alternative solutions but also enhances overall detection performance. Comparing $\mathcal{M}_1$ with $\mathcal{M}_2$, we find that AskDB and

Table 4: Error detection performance w/ different baseline methods ($\mathcal{M}_0$: Naive prompt, $\mathcal{M}_1$:CoT prompt, $\mathcal{M}_2$: Naive prompt + AskBD, $\mathcal{M}_3$: CoT prompt + AskBD) on ordinary solutions ($\mathcal{D}$) and alternative solutions ($\mathcal{D}'$). The performance gap is calculated by $\Delta = |\mathcal{D} - \mathcal{D}'|$. The best performed results in each group is marked with underline.

Model	$\mathcal{D}\uparrow$				$\mathcal{D}'\uparrow$				$\Delta\downarrow$			
	$\mathcal{M}_0$	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_0$	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_0$	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$
Small												
GPT-4o	27.2	47.7	48.8	59.1	18.4	36.7	37.8	48.5	8.8	11.0	11.0	10.6
Claude-3.5	38.2	51.1	50.7	56.6	34.7	48.5	44.0	51.8	3.5	2.6	6.7	4.8
Gemini-1.5	46.4	54.2	61.7	62.4	39.5	49.5	55.5	59.5	6.9	4.7	6.2	2.9
Llama-3.1	20.2	34.6	23.5	37.9	20.9	31.0	23.6	32.4	0.7	3.6	0.1	5.4
Qwen-2.5	24.7	40.3	34.4	44.3	16.3	35.3	25.0	38.3	8.4	5.0	9.5	6.0
Qwen-2.5-Math	24.2	23.3	23.4	23.8	20.7	23.0	22.2	23.7	3.5	0.3	1.2	0.1
Large												
GPT-4o	52.9	63.4	67.1	66.3	43.4	59.3	58.0	61.4	9.5	4.1	9.1	4.9
Claude-3.5	59.0	61.7	72.5	73.1	52.7	57.0	67.4	69.5	6.3	4.7	5.2	3.6
Gemini-1.5	65.2	65.6	76.0	71.8	55.6	58.1	72.2	68.8	9.6	7.5	3.8	3.0
Llama-3.1	44.3	64.0	63.4	71.9	37.2	56.4	57.1	67.9	7.2	7.6	6.3	4.0
Qwen-2.5	46.3	57.4	45.4	60.2	38.8	50.6	43.0	58.2	7.5	6.8	2.4	2.0
Qwen-2.5-Math	43.5	41.2	49.6	48.2	39.9	40.0	47.7	46.9	3.6	1.2	1.9	1.3

CoT prompts enable different LLMs to achieve better results. In summary, AskDB is more compatible with advanced models, while CoT demonstrates greater efficacy with base-sized models.

To address RQ3, we compare $\mathcal{M}_3$ with $\mathcal{M}_1$. The results reveal that AskDB is highly compatible with other reasoning-enhancing techniques such as CoT prompts in the context of error detection. For advanced model of Llama-3.1 and base model of Gemini-1.5, combining AskDB with CoT yields significant performance improvements compared to using either method independently. These findings confirm that AskDB is a robust approach for mitigating conformity bias. Moreover, its compatibility with other reasoning-enhancement techniques achieves the best overall performance in error detection tasks.

4.3 Further Studies

It is valuable to explore AskBD's performance when faced with multiple errors, as successful detection of these errors could offer a comprehensive view of the student's problems in a single instance. To explore this, we sampled 100 solutions each from the original and alternative solutions, introducing two sequential errors into them. To evaluate performance, we used two metrics: (1) first error detection accuracy and (2) overall error detection accuracy. The results are shown in the Table 5. From the table, we can confirm that our AskBD ($\mathcal{M}_3$) demonstrates robust performance in detecting the first error, even when multiple errors are present in the solution. Although the overall error detection accuracy is lower than the first error detection accuracy, we can still confirm that the reference-based strategy introduced in AskBD helps LLMs achieve better performance compared to the baselines.

Table 5: Error detection performance w/ different baseline methods ($\mathcal{M}_0$: Naive prompt, $\mathcal{M}_1$:CoT prompt, $\mathcal{M}_2$: Naive prompt + AskBD, $\mathcal{M}_3$: CoT prompt + AskBD) on solutions with multiple errors.

Mode	First				Overall			
	$\mathcal{M}_0$	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_0$	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$
Small								
GPT-4o	18.8	35.9	38.6	65.8	18.8	35.9	38.6	45.2
Claude-3.5	19.0	56.5	38.3	59.5	18.1	40.6	29.0	40.3
Gemini-1.5	20.5	38.2	63.5	69.5	16.3	26.7	38.8	44.8
Large								
GPT-4o	41.7	62.0	69.0	72.2	31.9	41.0	47.4	47.3
Claude-3.5	31.3	57.3	62.2	71.7	29.3	48.9	45.6	54.6
Gemini-1.5	57.3	65.0	80.2	80.0	44.1	45.8	61.2	57.1

5 Related Work

5.1 Automatic Error Detection

Automatic error detection (AED) is a widely studied research task in the field of education (Zamora et al., 2018). Since the advent of pre-trained language models (PLMs) such as BERT (Kenton

and Toutanova, 2019), AED algorithms in language education have achieved significant advancements (Bryant et al., 2023). Applications like grammar error detection have been widely implemented in the teaching of languages (He, 2021). Moreover, by integrating PLMs with acoustic models, AED has also shown promising results in detecting pronunciation errors (Wei et al., 2022b). The recent emergence of large language models (LLMs) has further expanded the scope of AED research beyond language education. Leveraging their advanced capabilities in mathematical reasoning (Ahn et al., 2024), task planning (Huang et al., 2024), and even programming (Nam et al., 2024), LLMs have been increasingly adopted in recent studies to develop AED solutions for complex educational subjects, such as programming (Gabbay and Cohen, 2024) and mathematics (Yan et al., 2024).

5.2 Math Reasoning in LLMs

Reasoning capability is one of the most attractive features reported among the emergent capabilities of large language models (LLMs). Building on approaches such as chain-of-thought (Wei et al., 2022a), LLMs have demonstrated impressive performance in solving complex logical reasoning problems. However, recent studies (Prabhakar et al., 2024) have raised skepticism about these reasoning capabilities, suggesting they may primarily originate from memorization rather than genuine reasoning ability. To address these concerns, numerous new reasoning tasks and benchmark datasets have been introduced (Srivastava et al., 2024). Among these, approaches that involve error detection and correction of flawed solutions have gained popularity in the community as a means to evaluate true mathematical reasoning capabilities, aided by the availability of extensive benchmark datasets (Zhou et al., 2024). To reduce the burden of tedious human annotation, many recent works have proposed algorithms to automatically generate inputs for these tasks based on existing datasets (Li et al., 2024). Through extensive experiments on these newly introduced mathematical reasoning tasks, the reasoning capabilities of LLMs have been rigorously evaluated and significantly validated. Moreover, with the rapid advancements in multi-modal large language models, investigating the multimodal mathematical reasoning capabilities of current vision-language LLMs is becoming an increasingly prominent area of research (Yan et al., 2024).

6 Conclusion

In this work, we investigate the behavior of LLM-powered error detectors when encountering alternative solutions commonly found in real-world math word problems. Through a preliminary study on an alternative solution error detection dataset, we identify and confirm the presence of conformity bias in LLMs during error detection. Motivated by our findings on the impact of incorporating reference solutions, we propose the Ask-Before-Detection (AskBD) framework, which enhances error detection by adaptively generating reference solutions. Comprehensive experiments on 200 examples from GSM8K demonstrate the effectiveness of AskBD in mitigating conformity bias. Furthermore, when combined with reasoning enhancement techniques like chain-of-thought (CoT) prompting, AskBD achieves significant improvements in both bias mitigation and overall performance.

Limitation

In this work, we identify conformity bias in LLM-powered error detectors for math word problem (MWP) solutions using 200 seed samples from the GSM8K dataset. During the data preparation process, we selected four common error types in student solutions as targets to simulate real-world error detection scenarios. However, this approach has limitations, as it overlooks the occurrence of rarer but potentially more challenging error types in student solutions. To address this, we plan to collect samples from real-world student responses in future iterations of our study. Additionally, this work focuses exclusively on alternative solutions for math word problems. The phenomenon of multiple valid solutions to a single problem is widespread across other subjects in education. In future research, we aim to extend our analysis of conformity bias to these subjects, contributing to the development of LLM-powered detectors as more effective tools in educational contexts.

Acknowledge

Hang Li, Kaiqi Yang, Yucheng Chu and Hui Liu are supported by the National Science Foundation (NSF) under grant numbers CNS2321416, IIS2212032, IIS2212144, IOS2107215, DUE2234015, CNS2246050, DRL2405483 and IOS2035472, US Department of Commerce, Gates Foundation, the Michigan Department of Agriculture and Rural Development, Amazon, Meta, and SNAP.

References

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

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

Christopher Bryant, Zheng Yuan, Muhammad Reza Qorib, Hannan Cao, Hwee Tou Ng, and Ted Briscoe. 2023. Grammatical error correction: A survey of the state of the art. *Computational Linguistics*, 49(3):643–701.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Nico Daheim, Jakub Macina, Manu Kapur, Iryna Gurevych, and Mrinmaya Sachan. 2024. Stepwise verification and remediation of student reasoning errors with large language model tutors. *arXiv preprint arXiv:2407.09136*.

Hagit Gabbay and Anat Cohen. 2024. Combining llm-generated and test-based feedback in a mooc for programming. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*, pages 177–187.

Zhenhui He. 2021. English grammar error detection using recurrent neural networks. *Scientific Programming*, 2021(1):7058723.

Xinyi Huang, Di Zou, Gary Cheng, Xieling Chen, and Haoran Xie. 2023. Trends, research issues and applications of artificial intelligence in language education. *Educational Technology & Society*, 26(1):112–131.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*.

Zhuoxuan Jiang, Haoyuan Peng, Shanshan Feng, Fan Li, and Dongsheng Li. 2024. Llms can find mathematical reasoning mistakes by pedagogical chain-of-thought. *arXiv preprint arXiv:2405.06705*.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2. Minneapolis, Minnesota.

Claudia Leacock, Martin Chodorow, Michael Gamon, and Joel Tetreault. 2014. *Automated grammatical error detection for language learners*. Morgan & Claypool Publishers.

Xiaoyuan Li, Wenjie Wang, Moxin Li, Junrong Guo, Yang Zhang, and Fuli Feng. 2024. Evaluating mathematical reasoning of large language models: A focus on error identification and correction. *arXiv preprint arXiv:2406.00755*.

Marcus Messer, Neil CC Brown, Michael Kölling, and Miaojing Shi. 2024. Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computing Education*, 24(1):1–43.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, 56(2):1–40.

Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*.

Akshara Prabhakar, Thomas L Griffiths, and R Thomas McCoy. 2024. Deciphering the factors influencing the efficacy of chain-of-thought: Probability, memorization, and noisy reasoning. *arXiv preprint arXiv:2407.01687*.

Saurabh Srivastava, Anto PV, Shashank Menon, Ajay Sukumar, Alan Philipose, Stevin Prince, Sooraj Thomas, et al. 2024. Functional benchmarks for robust evaluation of reasoning performance, and the reasoning gap. *arXiv preprint arXiv:2402.19450*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022a. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Xing Wei, Catia Cucchiarini, Roeland van Hout, and Helmer Strik. 2022b. Automatic speech recognition and pronunciation error detection of dutch non-native speech: cumulating speech resources in a pluricentric language. *Speech Communication*, 144:1–9.

Yibo Yan, Shen Wang, Jiahao Huo, Hang Li, Boyan Li, Jiamin Su, Xiong Gao, Yi-Fan Zhang, Tianlong Xu, Zhendong Chu, et al. 2024. Errorradar: Benchmarking complex mathematical reasoning of multimodal large language models via error detection. *arXiv preprint arXiv:2410.04509*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

An-Zi Yen and Wei-Ling Hsu. 2023. Three questions concerning the use of large language models to facilitate mathematics learning. *arXiv preprint arXiv:2310.13615*.

Ángela Zamora, José Manuel Suárez, and Diego Ardura. 2018. Error detection and self-assessment as mechanisms to promote self-regulation of learning among secondary education students. *The Journal of Educational Research*, 111(2):175–185.

Zihao Zhou, Shudong Liu, Maizhen Ning, Wei Liu, Jindong Wang, Derek F Wong, Xiaowei Huang, Qiufeng Wang, and Kaizhu Huang. 2024. Is your model really a good math reasoner? evaluating mathematical reasoning with checklist. *arXiv preprint arXiv:2407.08733*.

A Dataset Statistics

Table 6: Statistics on conventional solutions ($\mathcal{D}$) and alternative solutions ($\mathcal{D}'$) across different error categories.

Solution	Correct	Error			
		$\mathcal{E}_C$	$\mathcal{E}_R$	$\mathcal{E}_M$	$\mathcal{E}_H$
$\mathcal{D}$	200	200	200	200	200
$\mathcal{D}'$	200	200	200	200	200

B LLM Details

Table 7: Details about LLM implementation in this paper and source file links.

LLM Name	Model ID
Small	
GPT-4o	gpt-4o-mini-2024-07-18
Claude-3.5	claude-3-5-haiku-20241022
Gemini-1.5	Gemini-1.5-Flash-002
Llama-3.1	Llama-3.1-8B-Instruct
Qwen-2.5	Qwen2.5-7B-Instruct
Qwen-2.5-Math	Qwen2.5-Math-7B-Instruct
Large	
GPT-4o	gpt-4o-2024-08-06
Claude-3.5	claude-3-5-sonnet-20240620
Gemini-1.5	Gemini-1.5-Pro-002
Llama-3.1	Llama-3.1-70B-Instruct
Qwen-2.5	Qwen2.5-72B-Instruct
Qwen-2.5-Math	Qwen2.5-Math-72B-Instruct

C Error Definitions

D Naive Instruction Prompt

E COT Instruction Prompt

Table 8: Detailed definition to each error. The error location for each example is marked with the red color. The original solution is presented at the top of the table for reference.

Error	Definition	Example
-	Correct solution.	Step 1. Each gust blows the leaf forward 5 feet, so 11 gusts will blow it forward $5 \times 11 = 55$ feet. Step 2. Each swirl after a gust blows it back 2 feet, so 11 swirls will blow it back $2 \times 11 = 22$ feet. Step 3. After 11 gusts, the leaf has traveled $55 - 22 = 33$ feet down the sidewalk.
$\mathcal{E}_C$	Operands in expressions are correct but an error occurs in the calculated results.	Step 1. Each gust blows the leaf forward 5 feet, so 11 gusts will blow it forward $5 \times 11 = 50$ feet. Step 2. Each swirl after a gust blows it back 2 feet, so 11 swirls will blow it back $2 \times 11 = 22$ feet. Step 3. After 11 gusts, the leaf has traveled $55 - 22 = 33$ feet down the sidewalk.
$\mathcal{E}_R$	Expression are incorrectly referencing the question conditions or the results from prior steps.	Step 1. Each gust blows the leaf forward 5 feet, so 10 gusts will blow it forward $5 \times 10 = 50$ feet. Step 2. Each swirl after a gust blows it back 2 feet, so 11 swirls will blow it back $2 \times 11 = 22$ feet. Step 3. After 11 gusts, the leaf has traveled $55 - 22 = 33$ feet down the sidewalk.
$\mathcal{E}_M$	Operands or expressions in the step that lack of references or support from the question conditions or prior steps.	Step 1. Each swirl after a gust blows it back 2 feet, so 11 swirls will blow it back $2 \times 11 = 22$ feet. Step 2. After 11 gusts, the leaf has traveled $55 - 22 = 33$ feet down the sidewalk.
$\mathcal{E}_H$	Statements or operands in the listed expression are fabricated or inconsistent with the question's conditions.	Step 1. Each gust blows the leaf forward 5 feet, so 11 gusts will blow it forward $5 \times 11 = 55$ feet. Step 2. Each swirl after a gust blows it back 2 feet, so 11 swirls will blow it back $2 \times 11 = 22$ feet. Step 3. After 11 gusts, the leaf has traveled $55 - 22 = 33$ feet down the sidewalk. Finally, a wind blew the leaf 10 feet forward, and the leaf traveled $33 + 10 = 43$ feet.

Given the <question>, please judge whether each step in <solution> is correct. **During the judging process,you should know that the <question> does not always have only one standard solution, and any reasonable <solution> should be accepted. You should pay attention to both the expressions and the statements in each step, and take care about the logic consistency between different steps. Additionally, consider arithmetic expression equivalency and avoid rejecting solutions solely because they use equivalent expressions.**

In each step, if no errors are found, respond with Step X: <correct>. If you find that the operands in the listed expressions are correct but an error occurs in the calculated result, respond with Step X: <calculation error>. If you find statements or operands in the listed expression are incorrectly referencing the question conditions or the results from prior steps, respond with Step X: <reference error>. If you find operands or expressions in the step that is lack of references or support from the question conditions or prior steps, respond with Step X: <missing step>. If you find statements or operands in the listed expression are fabricated or inconsistent with the question's conditions, respond with: Step X: <hallucination>. If an error is a follow-on issue due to mistakes in previous steps rather than an independent error, respond with: Step X: <secondary error>.

<question> [Question Text] <solution> [Solution Text]
Now, please start to respond.

Figure 5: The example prompt we used to implement the error detector with LLMs includes specific formatting for clarity. Instruction text guiding the LLMs to accept alternative solutions is highlighted in **bold**, while the definitions of error categories are emphasized in *italic*. Text enclosed in square brackets serves as placeholders for the input question and solution text, respectively.

Given the <question>, please judge whether each step in <solution> is correct. **During the judging process,you should know that the <question> does not always have only one standard solution, and any reasonable <solution> should be accepted. You should pay attention to both the expressions and the statements in each step, and take care about the logic consistency between different steps. Additionally, consider arithmetic expression equivalency and avoid rejecting solutions solely because they use equivalent expressions.**

In each step, if no errors are found, respond with Step X: <correct>. If you find that the operands in the listed expressions are correct but an error occurs in the calculated result, respond with Step X: <calculation error>. If you find statements or operands in the listed expression are incorrectly referencing the question conditions or the results from prior steps, respond with Step X: <reference error>. If you find operands or expressions in the step that is lack of references or support from the question conditions or prior steps, respond with Step X: <missing step>. If you find statements or operands in the listed expression are fabricated or inconsistent with the question's conditions, respond with: Step X: <hallucination>. If an error is a follow-on issue due to mistakes in previous steps rather than an independent error, respond with: Step X: <secondary error>.

Before the <response>, you should provide your step-by-step <thinking> about your judging process.

<question> [Question Text] <solution> [Solution Text]
Now, please start to ***think first and then*** respond.

Figure 6: The example CoT prompt we used to implement the error detector with LLMs includes specific formatting for clarity. Instruction text guiding the LLMs to accept alternative solutions is highlighted in **bold**, while the definitions of error categories are emphasized in *italic*. Text enclosed in square brackets serves as placeholders for the input question and solution text, respectively.