Multi-step Large Language Model for Fine-Grained Feedback in Stepwise Linear Equation Solutions

Imran Chamieh[1], Torsten Zesch[2], Klaus Giebermann[1]
[1]Hochschule Ruhr West, Germany
[2]CATALPA, FernUniversität in Hagen, Germany

Abstract

This paper addresses the problem of fine-grained error classification in stepwise algebraic problem solving, with the objective of enabling precise and timely feedback in large-scale educational environments. Using authentic student data, we compare a carefully engineered rule-based baseline with large language models (LLMs) in zero-shot and few-shot configurations, as well as multi-step LLM-based approaches. We further consider hybrid architectures that combine symbolic computation with LLM-based reasoning. Our empirical results indicate that, although the baseline model delivers strong and reliable performance for narrowly defined error categories, the proposed multi-step LLM framework surpasses the rule-based model by achieving superior precision, F1 scores, and overall accuracy. These findings underscore the limitations of unstructured end-to-end prompting for error classification.

1 Introduction

Digital mathematics learning platforms routinely mark intermediate solution steps as correct or incorrect yet often fail to explain *why* a step is wrong. Whereas short feedback can lead the student to the correct solution. Consider the following example from Figure 1

$$\begin{aligned} S_2: \quad & 6q - 4 = 3 \\ S_3: \quad & 6q \quad\;\; = 3 - 4 \end{aligned}$$

For a human instructor, it is readily apparent that the student made a sign error in the step S_3 by failing to change the sign of -4 when transposing it to the right-hand side of the equation. In conventional classroom settings, instructors typically identify such errors through detailed examination of students' solution processes and subsequently provide targeted, formative feedback. At a large scale, such as with intelligent tutoring systems (ITS), however, this process becomes increasingly difficult. ITS rely heavily on computer algebra systems (CAS) to verify whether a student's transformation is mathematically valid. While CAS reliably detects incorrect steps, it typically does not explain why a step is incorrect, so students receive binary correctness signals without diagnostic specificity, which lacks the useful feedback that allows students to self-correct their mistakes, as illustrated in 1. As a result, students may struggle to interpret their mistakes. In such cases, the educational value lies not in telling the student that the answer is wrong but in understanding *why*.

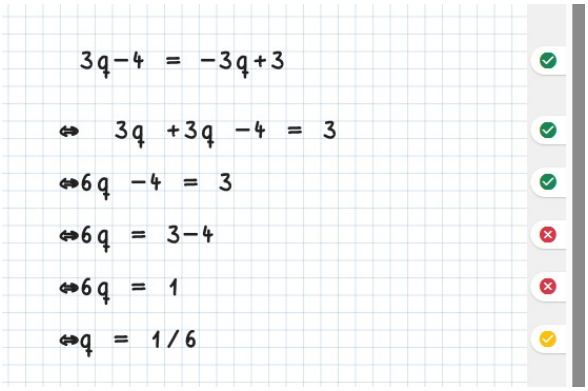

Figure 1: Example of binary error detection in an algebraic task. The system identifies incorrect steps but fails to provide explanatory feedback to the student.

This paper examines the problem of fine-grained error classification in stepwise algebraic solutions, in contrast to the more conventional focus on assessing the correctness of complete solutions. We delineate a narrowly defined yet pedagogically meaningful error category that supports a direct mapping to targeted instructional feedback. Specifically, we address the detection of explicit algebraic sign errors ($+ \leftrightarrow -$) in linear equations, using step-level interaction data collected from an intelligent tutoring system.

Proceedings of the 21st Workshop on Innovative Use of NLP for Building Educational Applications, pages 419–431

We formalize this construct as follows: given an initial algebraic equation S_i and the subsequent step S_{i+1} produced by a student, S_{i+1} is labeled `True` if and only if a sign inversion $(+ \leftrightarrow -)$ is sufficient to render the transformation mathematically valid. In cases where multiple errors occur, the label is `True` if a sign inversion restores the mathematical validity of the affected term.

The central research question motivating this work is the following:

To what extent can different modeling paradigms reliably detect explicit algebraic sign errors at the step level in a manner that can be generalized to other error categories?

To address this question, we design an experimental framework that operationalizes the identification of explicit sign inversions in linear equations in accordance to the above definition. Within this framework, we compare three modeling approaches:

1. a **rule-based model** that explicitly enumerates valid algebraic transformations, matches them to student-generated steps, identifies the closest correct transformation, and then determines whether a sign error is present;

2. a **single-prompt LLM model** that frames sign-error detection as a zero-shot and few-shot classification task over (S_i, S_{i+1}) step pairs;

3. a **multi-step LLM pipeline** that decomposes the reasoning process into distinct stages (simplification, intent inference, and error classification), with each stage handled by a dedicated LLM or CAS component.

The final approach is structurally analogous to chain-of-thought reasoning in that it decomposes a complex decision into a sequence of specialized subtasks, thereby reducing the cognitive and computational burden on any single model invocation. Consequently, we refer to this configuration as a multi-step LLM pipeline composed of multiple specialized components, rather than a single monolithic prompt.

Our contributions are threefold:

1. We introduce a step-level dataset of annotated linear-equation transformations that specifically targets explicit sign errors[1].

2. We conduct a controlled comparison between symbolic (rule-based) and LLM-based methods for fine-grained algebraic error detection.

3. We propose a decompositional, multi-step LLM framework designed to mitigate overgeneralization and enhance diagnostic precision.

2 Related Work

Beyond the binary verification provided by standard CAS, the field has long pursued the automation of fine-grained diagnostic feedback through multiple technical paradigms. Early approaches focused on model tracing and symbolic reasoning, which depend on explicitly defined domain models to map student deviations against an "expert" path (Anderson et al., 1995). In contrast to generic computer algebra verification, these systems use expert-defined rules to infer which specific algebraic transformation a student likely attempted. Although such rule-based classifiers can attain high precision under controlled conditions (McTavish and Larusson, 2014), they are commonly constrained by a pronounced "authoring bottleneck" (Bent et al., 2019). Systematically encoding the full space of student misconceptions as rigid symbolic rules is highly labor-intensive and does not scale with increasing problem complexity or heterogeneity in student behavior.

With the advent of data-driven methodologies, researchers address the limitations of manual rule engineering by adopting supervised learning–based classifiers trained on labeled student solutions. Recent work ranges from statistical diagnostic models for algebra misconceptions, like McNichols et al. (2023) works, to neural architectures that categorize fine-grained algebraic error types from stepwise solutions or written responses (Gomes and Jaques, 2020; Russell et al., 2009; Kirve et al., 2025). While these models reduce the inflexibility associated with rule-based systems, they function as "black-box" predictors and require substantial quantities of labeled training data. This requirement poses a significant limitation in contexts involving rare or highly specialized mathematical misconceptions, for which annotated corpora are typically scarce.

Recent advancements in LLMs have introduced a new paradigm, offering a balance between the structured reasoning capabilities traditionally associated with symbolic systems and the flexibility

[1] https://github.com/Imrancham/Step-Level-Error-Detection-in-Linear-Equations

ID	S_i	S_{i+1}	Sign Error
1	$6q - 4 = 3$	$6q = 3 - 4$	True
2	$6q - 4 = 3$	$6q = -1$	False
3	$5x + 5 = x - 1$	$4x = 4$	False
4	$y + x = -x + 2$	$y = 2$	False
5	$q(h - y) = 0$	$qh + qh = 0$	True
6	$2a + 4 = 0$	$2x = -4$	False
7	$-3(x - 1) = 0$	$-3x = 3$	True
8	$x + 2 = 3$	$x = 5$	False

Table 1: Sample of equation pairs (S_i, S_{i+1}) that have different error types.

of data-driven models, while reducing the need for the huge datasets typically required by purely data-driven systems. McNichols et al. (2024) investigated the zero-shot diagnostic capabilities of LLMs such as GPT-4 in educational contexts. A recurring observation in this literature is the presence of a "precision gap" namely, the tendency of LLMs to overgeneralize when fine-grained diagnostic specificity is needed (Chamieh et al., 2024). Our work addresses this by moving away from unstructured, end-to-end prompting toward a multi-step framework that reintroduces symbolic-like rigor into the multi-step LLM process.

3 Dataset

The dataset described in this study was collected from a digital mathematics learning platform used in undergraduate engineering courses at Hochschule Ruhr West (Ruhr West University of Applied Sciences). Approximately 144 first- and second-semester students used the platform as part of mandatory mathematics courses. The platform generates individualized exercises using parameterized mathematical templates and records student solutions at the step level. Whenever a submitted step does not follow validly from the preceding one, the system stores a record comprising the previous equation (S_i) and the student's produced next step (S_{i+1}). Although the platform detects the occurrence of an error, it does not provide a classification of the underlying error type and, hence, no targeted feedback.

3.1 Data Filtering and Annotation

Since the winter semester of 2022/23, 49,661 incorrect solution steps have been collected. These range from basic arithmetic slips to complex conceptual errors in trigonometry or calculus. For the purposes of the present study, we restrict our focus to linear equations. We filtered the raw logs to exclude entries containing non-linear terms (e.g., higher-degree polynomials, square roots, or variables in denominators), resulting in a subset of 4,024 equation pairs.

Each pair (S_i, S_{i+1}) was manually annotated by a domain expert actively involved in teaching the corresponding courses, with a binary label indicating the presence (`True`) or absence (`False`) of an **explicit sign error**.

Note that, according to our definition of sign error in the introduction, we exclude arithmetic mistakes in which a student carries out a numerical computation incorrectly while otherwise maintaining valid algebraic reasoning. For instance, in the second example in Table 1, the step $S_{i+1} : 6q = -1$ is not classified as a sign error, because no single modification of a sign would convert the transformation into a correct one, even though the incorrect result stems from an erroneous evaluation of $3 - 4$, in which the term -4 appears with an incorrect sign. In contrast, in the first example in Table 1, the step yielding $S_{i+1} : 6q = 3 - 4$ is categorized as an explicit sign error, since changing the minus in -4 to a plus would render the equation correct. Overall, there are 780 of the annotated instances that exhibit such explicit sign errors, which makes approximately 19%.

3.2 Dataset statistics

Figure 2 provides a visualized distribution of the dataset. We observe that student-produced equations S_{i+1} are, on average, shorter than the source equation S_i, reflecting term aggregation during manipulation. A slight increase in the number of distinct variables is observed in S_{i+1}, can be seen in the Figure 2 on the right, primarily due to handwriting-recognition artifacts (e.g., confusion between α and a, ω and w, or 0 and o).

4 Experimental Setup

This section describes the models and configurations used to identify explicit sign errors in stepwise linear equation transformations. Figure 3 provides an overview of the processing pipeline shared across approaches.

Models Evaluated To ensure our findings generalize across different architectures, we evaluate both commercial and open-weight large language models across all single-step and multi-step configurations. We consider the following models: `gemini-2.5-flash`, `gpt-5-mini`, `mistral-large-latest`, and `Llama-3-70b`.

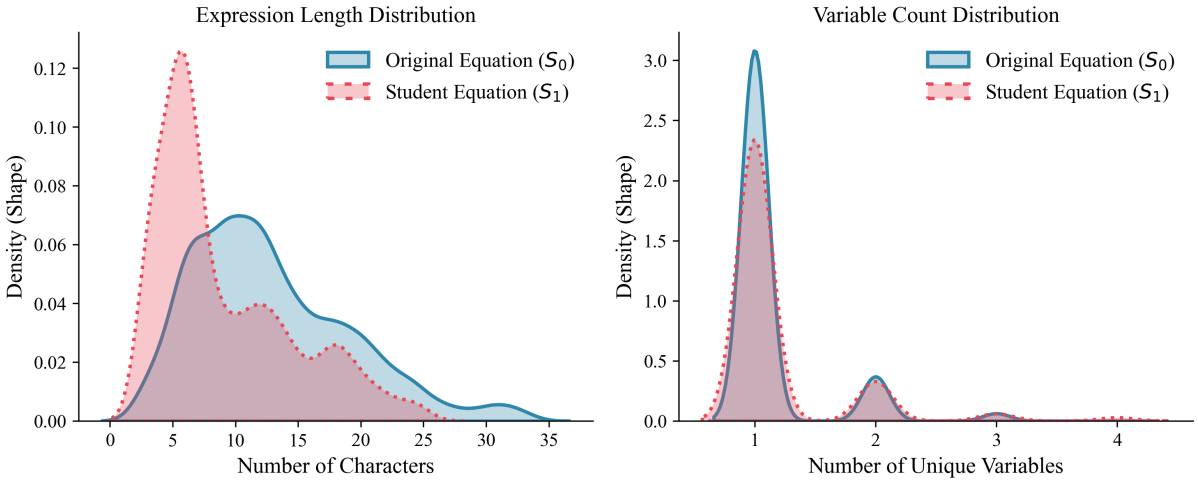

Figure 2: Distributional statistics for the dataset

Models are accessed via their respective APIs using default decoding parameters (temperature and sampling). `Llama-3-70b` is additionally evaluated locally on a machine equipped with 8 NVIDIA Quadro RTX 8000 GPUs, using the same generation settings to mirror the API configuration. To mitigate run-to-run variability inherent in generative models and to ensure stable performance estimates, all LLM-based experiments are repeated five times, and we report the mean and standard deviation performance across these runs.

4.1 Rule-Based Baseline

We implement a rule-based model as a high-precision and fully interpretable baseline for sign error detection. Rule-based systems allow explicit control over error definitions and decision criteria, which is particularly important in educational feedback settings where false positives can negatively affect student trust. This model also serves as a reference point for evaluating the behavior of LLM-based approaches. The pipeline consists of three steps.

Transformation Enumeration Given an initial equation S_i, we construct a transformation tree that enumerates all possible subsequent algebraic operations, including term transposition, coefficient normalization via division, distribution, and progressive variable isolation. Each node in this tree corresponds to an equation that is algebraically equivalent to S_i; see Figure 4. This procedure establishes the foundation for inferring student intent by enabling the identification, in a subsequent stage, of the equation that is most similar to the student's produced step.

Similarity Matching. To approximate the student's intended transformation, we compute a similarity score between the student's equation S_{i+1} and each equation in the transformations tree. Prior to comparison, equations are represented in two normalized forms: (i) a structural form derived from SymPy's internal expression representation and (ii) a canonical string form obtained by rewriting equations into a normalized representation (e.g., $a = b \rightarrow a - b = 0$). For each representation, similarity is computed using Python's `SequenceMatcher`, based on the Ratcliff–Obershelp algorithm Ratcliff et al. (1988), which estimates the degree of overlap between two string representations by identifying matching subsequences. The final similarity score is obtained as a weighted combination of the similarity scores computed over the structural and canonical representations. These weights are fixed across all experiments. The candidate transformation with the highest combined similarity score is selected as the reference equation.

Sign Error Detection Finally, we compare S_{i+1} against the selected reference equation term by term. Matching terms are removed, and remaining terms are inspected for sign inversion. If a term in S_{i+1} differs from its counterpart in the reference equation only by the sign attached to that term, and flipping this sign would make the step valid, the pair (S_i, S_{i+1}) is classified as containing an explicit sign error.

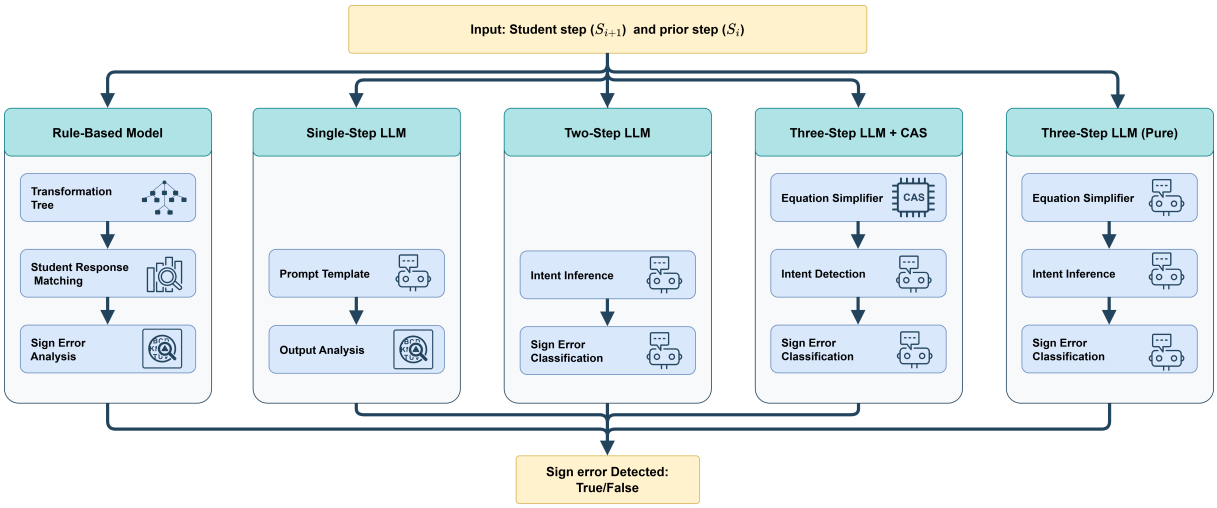

Figure 3: Comparative architecture of three diagnostic approaches—rule-based, multi-agent LLM, and prompt-based LLM—designed to identify sign errors by evaluating student input S_1 against the preceding step.

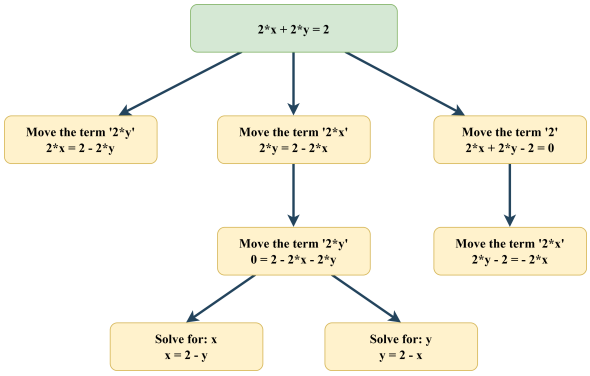

Figure 4: Tree-structured graph depicting the sequence of derived equation transformations, with the corresponding operations explicitly annotated at each transformation step.

4.2 Single-Step LLM

In the single-step paradigm, we evaluate the models as direct sign error classifiers under two distinct settings: zero-shot and few-shot. Each model receives the equation pair (S_i, S_{i+1}) along with a prompt defining explicit versus implicit sign errors in accordance with our annotation scheme. The models are instructed to output a binary decision (`True` or `False`) indicating the presence of an explicit sign error.

Prompt Engineering We experiment with multiple prompt variants that describe the task and define the notion of the explicit sign error, including different numbers of labeled examples. These prompt variants are evaluated on a subset of 200 examples of the data for each model, and we select a single best-performing prompt that is then used for all reported results.

Output Normalization LLMs occasionally produce explanatory text, despite being instructed to output only `True` or `False`. We therefore post-process outputs using simple regular expressions to extract the final binary decision.

4.3 Multi-Step LLM

To address the precision limitations and overgeneralization observed in the single-step paradigm, we hypothesize that decomposing sign-error detection into a sequence of focused subtasks improves diagnostic reliability. Rather than relying on a single model invocation to simultaneously perform mathematical simplification, intent inference, and error classification, we distribute these operations across distinct components with structured input–output contracts. We evaluate three variations of this multi-step architecture:

1. Three-Step Pure LLM Pipeline This configuration relies entirely on LLMs for all operations:

- **Step 1 (Canonical Simplification):** An LLM agent simplifies both the original equation S_i and the student equation S_{i+1} into canonical forms by combining similar terms. Where applicable, it also computes the correct final solution for S_i.

- **Step 2 (Intent Inference and Correct Transformation):** A second LLM receives the raw and simplified forms of both equations. It infers the student's intended algebraic operation

(e.g., transposing a term) and applies it correctly to S_i to generate a "gold" intermediate equation. It also flags cases where the student's canonical form deviates substantially from expectations, signaling non-sign-related errors (e.g., summation errors).

- **Step 3 (Sign-Error Classification):** A final LLM agent compares the student's raw equation with the "gold" intermediate equation from Step 2, determining if the discrepancy is strictly a single-term sign flip.

2. Hybrid CAS-LLM Pipeline To evaluate the benefit of integrating deterministic mathematical tools, this variant replaces the generative simplification step with a CAS tool. We utilize SymPy to programmatically simplify S_i and S_{i+1} and compute the correct final solution. The CAS-generated symbolic representations are then fed directly into the second step for Intent Inference and Classification in step 3, functioning exactly as described in the pure LLM pipeline.

3. Two-Step LLM Pipeline To determine whether canonical simplification is strictly necessary for accurate classification, this ablation omits Step 1 entirely. The raw equations (S_i, S_{i+1}) are passed directly to an LLM for Intent Inference to generate the "gold" intermediate step, followed immediately by the Sign-Error Classification step.

4.4 Evaluation Metrics

All models are evaluated using standard binary classification metrics, including precision (P), recall (R), F1-score (F1), and accuracy (Acc). Following common practice in educational error diagnosis, we place particular emphasis on precision, since incorrectly flagging correct solution steps or providing wrong feedback is undesirable in instructional settings. For all non-deterministic models (LLMs), reported metrics correspond to the mean over five independent runs.

5 Results and Discussion

Table 2 summarizes the performance of the rule-based baseline, the single-step LLM configurations (zero-shot and few-shot), and the proposed multi-step LLM framework for detecting explicit sign errors.

Overall Performance. The rule-based system constitutes a strong and well-balanced baseline

Model	P	R	F1	Acc
Rule-Based				
	.84	.82	.83	.93
Zero-Shot LLMs				
LLaMA	$.21_{\pm.000}$	$.77_{\pm.000}$	$.33_{\pm.000}$	$.44_{\pm.000}$
Mistral	$.47_{\pm.000}$	$.84_{\pm.002}$	$.60_{\pm.000}$	$.80_{\pm.000}$
GPT	$.37_{\pm.030}$	$.79_{\pm.150}$	$.51_{\pm.060}$	$.80_{\pm.000}$
Gemini	$.52_{\pm.001}$	$.81_{\pm.003}$	$.62_{\pm.004}$	$.83_{\pm.000}$
Few-Shot LLMs				
LLaMA	$.21_{\pm.005}$	$.78_{\pm.024}$	$.34_{\pm.008}$	$.45_{\pm.007}$
Mistral	$.53_{\pm.014}$	$.85_{\pm.027}$	$.65_{\pm.017}$	$.83_{\pm.009}$
GPT	$.55_{\pm.055}$	$.97_{\pm.019}$	$.70_{\pm.024}$	$.84_{\pm.006}$
Gemini	$.52_{\pm.017}$	$.94_{\pm.008}$	$.67_{\pm.011}$	$.82_{\pm.007}$
Two-Step LLMs				
LLaMA	$.19_{\pm.000}$	$.95_{\pm.006}$	$.32_{\pm.001}$	$.28_{\pm.002}$
Mistral	$.32_{\pm.008}$	$.73_{\pm.026}$	$.45_{\pm.013}$	$.68_{\pm.006}$
GPT	$.80_{\pm.015}$	$.55_{\pm.039}$	$.65_{\pm.021}$	$.56_{\pm.017}$
Gemini	$.95_{\pm.018}$	$.40_{\pm.030}$	$.56_{\pm.030}$	$.89_{\pm.005}$
Three-Step LLMs				
LLaMA	$.29_{\pm.159}$	$.96_{\pm.004}$	$.42_{\pm.158}$	$.37_{\pm.143}$
Mistral	$.30_{\pm.006}$	$.82_{\pm.019}$	$.44_{\pm.007}$	$.61_{\pm.010}$
GPT	$.85_{\pm.009}$	$.81_{\pm.026}$	$.83_{\pm.014}$	$.94_{\pm.002}$
Gemini	$.93_{\pm.019}$	$.78_{\pm.023}$	$.85_{\pm.026}$	$.95_{\pm.005}$
Three-Step CAS-LLMs				
LLaMA	$.24_{\pm.003}$	$.85_{\pm.017}$	$.37_{\pm.006}$	$.49_{\pm0.005}$
Mistral	$.24_{\pm.002}$	$.84_{\pm.005}$	$.38_{\pm.003}$	$.51_{\pm.003}$
GPT	$.87_{\pm.009}$	$.81_{\pm.025}$	$.83_{\pm.008}$	$.94_{\pm.002}$
Gemini	$.90_{\pm.013}$	$.82_{\pm.018}$	$.86_{\pm.020}$	$.95_{\pm.003}$

Table 2: Performance comparison of rule-based, prompt-based, and multi-agent models for sign error detection.

($P = 0.84$, $R = 0.82$, $F1 = 0.83$, $Acc = 0.93$), demonstrating that symbolic approaches remain highly effective when the target error category is narrowly specified and closely aligned with a fixed set of transformation rules.

Among all evaluated approaches, the highest-performing models are the **three-step LLM configurations**, in particular GPT and Gemini, which achieve performance comparable to, and in some cases marginally surpassing, the rule-based system. Gemini attains the highest overall scores ($P = 0.93$, $F1 = 0.85$, $Acc = 0.95$), while GPT exhibits similarly strong performance ($P = 0.85$, $F1 = 0.83$, $Acc = 0.94$). These findings suggest that decomposing the task into structured intermediate steps substantially enhances LLM performance relative to single-pass inference.

Zero-shot, Few-shot, and Multi-Step LLMs: Across all models, zero-shot and few-shot prompting display a consistent precision-recall trade-off, characterized by high recall but comparatively low precision. This pattern indicates that LLMs tend to over-predict sign errors when guided solely by

prompt-based conditioning, pointing to a misalignment between natural language-based reasoning and the formal definition of explicit sign errors.

The introduction of structure via multi-step decomposition fundamentally alters this behavior. Although two-stage pipelines remain unstable, three-step architectures systematically enhance the balance between precision and recall. This pattern indicates that an explicit factorization of the overall task into sub-components—such as simplification, intent inference, and classification—is crucial for aligning model predictions with well-defined formal error categories.

Importantly, augmenting the pipeline with a CAS component does not yield consistent performance gains, suggesting that the primary bottlenecks are not computational in nature but instead arise from difficulties in interpreting and propagating intermediate representations. Collectively, these results indicate that *structured reasoning, rather than the provision of additional training examples or external computational resources, constitutes the principal driver of the observed performance improvements*.

Comparison with Rule-Based Approach: While the rule-based system remains a strong baseline, the proposed multi-step LLM framework exhibits several distinct advantages. First, it attains comparable or marginally superior peak performance without dependence on manually crafted transformation rules. Second, it exhibits more robust generalization to a broader spectrum of transformation patterns, including those not encountered during development, whereas the rule-based system is intrinsically constrained by its fixed, predefined transformation hierarchy.

Crucially, the multi-step framework mitigates the systematic over-prediction characteristic of single-step LLMs by imposing intermediate structural constraints, thereby yielding more accurate and calibrated classifications. This finding indicates that explicit structured reasoning—rather than model scale alone—is a key factor in aligning LLM outputs with formally specified error taxonomies.

5.1 Error Analysis

Misclassifications across all approaches can be attributed to three primary sources: structural mismatch, intent ambiguity, and output-format inconsistency.

For the **rule-based system**, errors arise when the

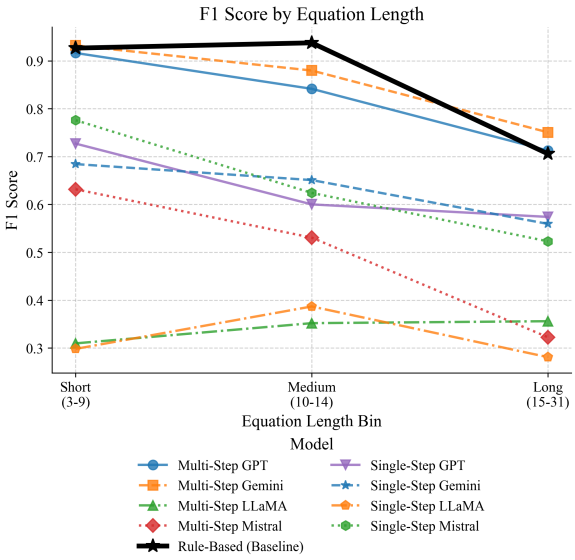

Figure 5: F1 scores across different equation length bins (Short, Medium, and Long).

similarity function fails to align a student's transformation with any rule present in the transformation tree. For instance, consider the transition,

$$\begin{aligned} S_i: \quad & -4t & &= 1-5y \\ S_{i+1}: \quad & t & &= (5y+1)/4 \end{aligned}$$

which is erroneously matched to,

$$best\_match: y = (4t+1)/5,$$

thereby resulting in an incorrect classification.

In the **multi-step architecture**, errors are primarily driven by difficulties in reliably inferring the student's underlying intent, particularly when the observed transformations deviate substantially from expected intermediate forms or involve several implicit operations. For instance:

$$\begin{aligned} S_i: \quad & z = -10(x-2) + 11(y+1) - 21 \\ S_{i+1} \quad & 10 = -10x + 11y \end{aligned}$$

In this case, the model assigns a coarse-grained intent category (distribution and simplification) but fails to correctly classify the step due to the omission of the variable z and the substantial structural divergence between the two expressions.

For **single-step LLMs**, the dominant source of error is a discrepancy between the model's internal reasoning and the strict operational definition of explicit sign errors. The models frequently produce correct mathematical interpretations but fail

to assign the corresponding label. For example:

$$\begin{aligned} S_i: &\quad -2p - 2 = 10 \\ S_{i+1}: &\quad p = 6 \end{aligned}$$

GPT response: *"No — explicit sign error: False. Reason: The correct solution is p = -6, and the student wrote p = 6. That is a wrong final sign but no incorrect sign operation is shown in the student's work."*
Although the model correctly infers that the mathematically valid solution is $p = -6$, it classifies the step as not containing an explicit sign error because no explicit sign manipulation is syntactically present in the student's work.

Open-Weight vs. Commercial Models: Open-weight models like Mistral and LLaMA often struggle to follow strict output formats, tending to produce free-form explanations instead of schema-compliant labels. This makes automatic parsing harder, complicates integration with downstream components, and can hurt overall pipeline performance. For instance, a typical LLaMA reply might say: *"The student's final simplified value has a sign error compared to the correct final value..."*. Although this is semantically correct, its unstructured form makes it difficult to automatically extract the underlying error category (e.g., a sign error). In contrast, commercial systems such as GPT and Gemini more reliably follow predefined schemas and generate structured, machine-interpretable outputs, which simplifies parsing and limits error propagation throughout the pipeline.

Impact of Equation Length: Equation length exerts a systematic negative effect on performance: all models do very well on short equations, begin to diverge on equations of medium length, and drop sharply on long equations (Figure 5). Rule-based methods and multi-step LLM approaches maintain relatively high accuracy on equations of medium length; however, all models struggle with long equations, where their performance converges and single-step LLMs are particularly impaired. These findings suggest that the increased compositional complexity associated with longer equations poses a specific challenge for single-step inference. Although multi-step reasoning strategies mitigate this effect to some extent, they do not fully resolve the underlying difficulty.

5.2 Summary

The results show that structured multi-step LLM systems can combine rule-based reasoning with generative flexibility to achieve higher precision and better performance by breaking tasks into subtasks and enforcing explicit structure on outputs. However, they still struggle with complex transformations, clarifying ambiguous student intent, and consistently following strict output formats, especially for open-weight models.

6 Conclusion and Future Work

In this study, we investigate the application of LLMs in various configurations for the detection of explicit sign errors in linear equation solving, with the objective of enabling accurate and timely feedback in large-scale educational environments. Using authentic student response data, we compare a carefully engineered rule-based baseline against zero-shot, few-shot, two-step, and three-step LLM-based pipelines, as well as hybrid variants augmented with CAS.

Our findings indicate that although the rule-based system constitutes a strong baseline, structured multi-step LLM approaches improve performance relative to single-step methods. In particular, three-step pipelines achieve the highest overall effectiveness, yielding a favorable balance between precision and recall and matching or surpassing the performance of the rule-based approach. In contrast, two-step pipelines exhibit instability, underscoring the importance of sufficient task decomposition. The integration of CAS results in only modest performance gains, suggesting that the primary challenges are rooted in reasoning and representation rather than in symbolic computation.

Future research will aim to generalize the proposed framework to encompass a wider spectrum of algebraic error categories and to address more structurally complex mathematical domains. Furthermore, we intend to systematically investigate hybrid architectures that more tightly integrate symbolic verification mechanisms with LLM-based inferential processes, with particular emphasis on enhancing the robustness and faithfulness of intermediate representations and mitigating error propagation across successive stages of the computational pipeline.

7 Limitations

This study is restricted to the detection of explicit algebraic sign errors occurring during transformations of linear equations, which limits the generalizability of our findings to other procedural errors commonly observed in student work, as well as to the treatment of more complex algebraic expressions that do not share the structural simplicity of linear equations. Extending the framework to additional error types would require corresponding modifications to the proposed multi-step architecture to handle a broader range of algebraic transformations. From a technical standpoint, we employed an identical prompt across all models while acknowledging that each model possesses distinct characteristics and prompt-sensitivity profiles. These model-specific properties may have influenced their respective outputs and, consequently, could have impacted the comparability of the results.

8 Ethics

This study utilizes screenshots of student solution steps, specifically those containing errors, collected from a digital mathematics learning platform. All data were fully anonymized prior to analysis, and no personally identifiable information was accessible to the authors at any stage of the study. All participating students provided informed consent via the platform's data usage agreement, permitting the use of their anonymized solution data for research purposes.

References

John R Anderson, Albert T Corbett, Kenneth R Koedinger, and Ray Pelletier. 1995. Cognitive tutors: Lessons learned. *The journal of the learning sciences*, 4(2):167–207.

R. V. D. Bent, J. Jeuring, and B. Heeren. 2019. The diagnosing behaviour of intelligent tutoring systems. pages 112–126.

Imran Chamieh, Torsten Zesch, and Klaus Giebermann. 2024. LLMs in short answer scoring: Limitations and promise of zero-shot and few-shot approaches. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*, pages 309–315, Mexico City, Mexico. Association for Computational Linguistics.

Joice Cazanoski Gomes and Patricia A Jaques. 2020. A data-driven approach for the identification of misconceptions in step-based tutoring systems. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 1122–1131. SBC.

Harish Kirve, Prajakta Ghugare, Ayush Telrandhe, and Prajwal Mahale. 2025. A fine-tuned transformer framework for the automated classification of algebraic misconceptions. *International Journal of Innovative Research in Technology*, 12(1):3648–3651.

Hunter McNichols, Jaewook Lee, Stephen Fancsali, Steve Ritter, and Andrew Lan. 2024. Can large language models replicate its feedback on open-ended math questions? *arXiv preprint arXiv:2405.06414*.

Hunter McNichols, Mengxue Zhang, and Andrew Lan. 2023. Algebra error classification with large language models. In *International Conference on Artificial Intelligence in Education*, pages 365–376. Springer.

Thomas McTavish and Johann Larusson. 2014. Discovering and describing types of mathematical errors. In *Educational Data Mining 2014*.

John W Ratcliff, David E Metzener, and 1 others. 1988. Pattern matching: The gestalt approach. *Dr. Dobb's Journal*, 13(7):46.

Michael Russell, Laura M O'dwyer, and Helena Miranda. 2009. Diagnosing students' misconceptions in algebra: Results from an experimental pilot study. *Behavior research methods*, 41(2):414–424.

9 Appendix

9.1 Confusion Matrices

	Predicted Negative	Predicted Positive
Actual Negative	3104	153
Actual Positive	143	624

Table 3: Confusion matrix Rule-Based

	Predicted Negative	Predicted Positive
Actual Negative	3007	250
Actual Positive	438	329

Table 4: Confusion Matrix Mistral Model

	Predicted Negative	Predicted Positive
Actual Negative	2661	596
Actual Positive	21	746

Table 5: Confusion Matrix GPT-5-mini Model

	Predicted Negative	Predicted Positive
Actual Negative	3090	167
Actual Positive	619	148

Table 6: Confusion Matrix LLaMA 3 Model

	Predicted Negative	Predicted Positive
Actual Negative	2460	797
Actual Positive	11	756

Table 7: Confusion Matrix Gemini Model

	Predicted Negative	Predicted Positive
Actual Negative	2750	507
Actual Positive	355	412

Table 8: Confusion Matrix Multi-Agent LLaMA-3

	Predicted Negative	Predicted Positive
Actual Negative	2950	307
Actual Positive	365	402

Table 9: Confusion Matrix Multi-Agent Mistral

	Predicted Negative	Predicted Positive
Actual Negative	3210	36
Actual Positive	158	620

Table 10: Confusion Matrix Multi-Agent Gemini

	Predicted Negative	Predicted Positive
Actual Negative	3150	96
Actual Positive	179	599

Table 11: Confusion Matrix Multi-Agent GPT-5-mini

9.2 Prompt Examples

Prompt 1:

Detect whether there is an EXPLICIT sign error.
Explicit = wrong +/- is written.
Other =No Explicit sign error appears.
Original: {t0}
Student: {t1}
Explicit Sign Error: True/False

Prompt 2:

You are an expert at detecting sign errors in algebra.
A sign error is TRUE only if an incorrect sign (+ or -) is explicitly written.
Examples:
Original: x + 5 = 15
Student: x = 15 + 5
Explicit Sign Error: True
Original: x + 5 = 15
Student: x = 20
Explicit Sign Error: False
Now evaluate:
Original: {t0}
Student: {t1}
Explicit Sign Error: True or False

Prompt 3:

You are an expert at detecting sign errors in mathematical expressions.
Affirm whether there is a sign error in the student's work or not.
A sign error is TRUE only when the student's written work shows an incorrect sign for a variable or numeric (+ instead of -, or vice versa).

EXPLICIT vs IMPLICIT:
- EXPLICIT (TRUE): Original "x + 1 = 2" Student writes "x = 2 + 1" when it should be "x = 2 - 1" → wrong sign is visible -> sign error: TRUE
- IMPLICIT (FALSE): Original "x + 1 = 2" Student writes -> "x = 3" when answer is 1 -> might be sign error mentally, but no sign is shown -> sign error: FALSE Example:

Original: x + 5 = 15
Student: x = 15 + 5
Explicit Sign Error: True
Implicit Sign Error: False

Original: x + 5 = 15
Student: x = 20
Explicit Sign Error: False
Implicit Sign Error: True

Original: -(3x + 4) = 12
Student: -3x - 4 = 12
Explicit Sign Error:: False
Implicit Sign Error: False Does the following transformation contain an explicit sign error?

Original: t0
Student: t1

Prompt 4:

You are an expert at detecting sign errors in mathematical expressions. Affirm whether there is a sign error in the student's work or not. A sign error is TRUE only when the student's written work shows an incorrect sign for a variable or numeric (+ instead of -, or vice versa).
EXPLICIT vs IMPLICIT:
- EXPLICIT (TRUE): Original "x + 1 = 2" Student writes "x = 2 + 1" when it should be "x = 2 - 1" → wrong sign is visible -> sign error: TRUE
- IMPLICIT (FALSE): Original "x + 1 = 2" Student writes -> "x = 3" when answer is 1 → might be sign error mentally, but no sign is shown -> sign error: FALSE
Example:
Original: x + 8 = 15
Student: x = 15 + 8
Explicit Sign Error: True
Probability of an Explicit Sign Error: 100
Original: -(3x - 4) = 12
Student: -3x - 4 = 12
Sign Error: True
Probability of an Explicit Sign Error: 100
Original: -(3x + 4) = 12
Student: -3x - 4 = 12
Sign Error:: False
Probability of an Explicit Sign Error: 1
Does the following transformation contain an explicit sign error?
Original: {t0}
Student: {t1}

9.3 Multi-Step Prompts

Step 1:

You are a math simplification agent. Your task is to mathematically simplify and canonicalize two equations.
ORIGINAL_EQUATION: "{original_eq}" STUDENT_EQUATION: "{student_eq}"
1. Combine all like terms (variables and constants) in the ORIGINAL_EQUATION to produce a canonical form.
2. Do the same for the STUDENT_EQUATION.
3. If the ORIGINAL_EQUATION can be fully solved, calculate the correct final value.
Respond with JSON in this exact format:
{{ "original_simplified": "canonical form of original", "student_simplified": "canonical form of student", "original_final_value": "final value if solvable, or 'unsolvable'" }}

Step 2:

You are an equation transformation agent. Your task is to infer the single logical step the student was attempting.
ORIGINAL_EQUATION: "{original_eq}"
STUDENT_EQUATION: "{student_eq}"
1. INFER INTENT: Describe the single major step (e.g., 'Transposing 3x to the RHS', 'Distributing -5').
2. PERFORM CORRECT TRANSFORMATION: Apply the inferred intent correctly to the ORIGINAL_EQUATION.
3. CHECK FOR PRE-EXISTING ERRORS: Compare canonical forms:
- Original Simplified: '{simplified_data.original_simplified}'
- Student Simplified: '{simplified_data.student_simplified}'
If the student's canonical form is radically different (indicating severe error like incorrect summation), set 'pre_existing_summation_error' to true.
Respond with JSON in this exact format:
{{ "student_intent": "description of intended operation",
"correct_intermediate_eq": "correct result of that operation",
"pre_existing_summation_error": true or false }}

Step 3:

You are the final error classification agent. Isolate a single explicit sign error by comparing the correct transformation result with the student's actual result.
1. CORRECT INTERMEDIATE EQUATION (Target): "{correct_intermediate_eq}"
2. STUDENT'S RAW EQUATION: "{student_eq_raw}"
3. STUDENT'S SIMPLIFIED EQUATION: "{simplified_data.student_simplified}"
4. CORRECT FINAL VALUE: "{simplified_data.original_final_value}"
5. PRE-EXISTING SUMMATION ERROR: {transform_data.pre_existing_summation_error}
Classification Logic:
Single Sign Flip Check:
- Check if STUDENT'S RAW EQUATION could be made identical to CORRECT INTERMEDIATE EQUATION by flipping the sign of ONLY ONE TERM. Set 'is_single_flip_correction_possible' accordingly.
Rule 1: Implicit/Summation Errors - IF PRE-EXISTING SUMMATION ERROR is True OR is_single_flip_correction_possible is False:
- Set error_type='Summation_Error'
- Set is_error_in_intermediate_step=True
- Set error_found=False
Rule 2: Explicit Sign Errors
- ELSE (Single flip is True AND No pre-existing summation error):
- Set error_found=True
- Find the single sign-flipped term
- Determine if error was due to Transposition (moving term across '=') or Distribution (expanding parentheses)
- Set is_error_in_intermediate_step=False
Rule 3: Arithmetic Simplification - If not Transposition/Distribution/Summation, check if student's final simplified value has a sign error compared to correct final value and classify as 'Arithmetic_Simplification'.
Respond with JSON in this exact format:
{{ "error_found": true or false,
"error_type": "Transposition" or "Distribution" or "Arithmetic_Simplification" or "Summation_Error" or "None",
"is_single_flip_correction_possible": true or false,
"is_error_in_intermediate_step": true or false,
"original_term": "term that was handled incorrectly",
"correct_sign": "sign it should have",
"student_sign": "sign student used",
"reasoning": "explanation of classification"
}}