History-Aware Two-Stage Mathematical Verifier: A Coarse-to-Fine Method for Step-Level Mistake Finding

Marcus Lee
Independent
mark.2025010101@gmail.com

***Abstract*—Large language models (LLMs) are strong reasoners yet prone to hallucinations, which limits their reliability in error-sensitive applications such as education. The existing mistake finding methods typically provide two output labels such as correct and incorrect, and struggle to integrate historical data without substantial computational overhead, limiting both interpretability and scalability. We propose a novel two-stage, coarse-to-fine framework that couples confidence-based binary detection with rich, explanatory outputs and incorporates simulated interaction history through token-thrifty user embedding techniques. Our method achieves competitive performance in weighted average F1 and binary classification accuracy scores on challenging benchmarks by iteratively refining errorful solution steps and conditioning re-evaluation on targeted corrections. Our approach offers a practical and effective solution for reducing LLM hallucinations and enhancing the reliability of LLM reasoning.**

***Index Terms*—Large Language Models, Intelligent Education, Mistake Finding**

I. Introduction

Large Language Models (LLMs) [1], [2] have been widely used in various fields in recent years. From a technical perspective, reducing LLM hallucinations [3] is of great importance, effective mistake finding [4]–[7] can identify hallucinations and therefore mitigate them. From an application perspective, such as in education, mistake finding can reveal student knowledge gaps and improve learning efficiency. However, most existing methods [5], [8], [9] output only a binary decision on whether an error exists, which fails to capture the specific causes and types of errors, moreover, many approaches require multiple LLM calls, leading to increased inference latency and operational costs. Addressing these issues is challenging: the error types underlying hallucinations are diverse and difficult to exhaustively enumerate [5]. Currently, most LLMs struggle with this task, leading to complex prompting and frequent model invocations. In practice, prompt-engineering-based binary judgments along one or multiple dimensions are often used. In this paper, we incorporate coarse-to-fine ideas and propose a two-stage framework that improves mistake finding accuracy.

These issues pose significant challenges. First, LLMs are inherently limited in performing mistake finding directly [5]. Second, enriching the binary output with explanations may induce hallucinations and introduce new errors. Third, LLMs are not adept at producing calibrated continuous scores. Fourth, leveraging previously annotated LLM error data would likely improve detection accuracy, however, naively concatenating raw historical annotations into the prompt, consumes many tokens especially for mathematics, and slows inference [10], [11]. Finally, historical and current data may differ in distribution, introducing noise. Robustly handling historical data is a challenge.

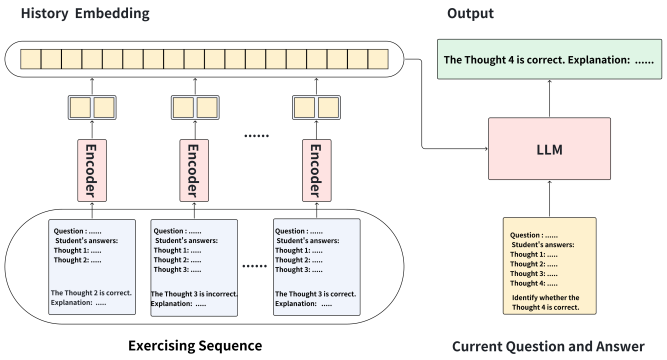

Fig. 1: Stage-1 model.

We address these challenges by fine-tuning on a small distilled dataset [12]. We apply the idea of knowledge tracing [13] to the field of mistake finding, and combine students' past answer records to assist in mistake finding. For questions with long answer records, we encode the answer records through the language model, project them into the LLM embedding space, and concatenate them with the current input, thereby minimizing the input token footprint while preserving historical context. For outputs, we jointly produce a binary decision and richer explanatory information. Our framework operates in two stages: Stage-1 performs coarse detection with explanations; Stage-2 focuses on difficult cases identified in Stage-1, uses the Stage-1 decision and explanations to guide targeted repair of suspected errorful steps, and then reclassifies, achieving higher accuracy. Providing explanatory text alongside the binary label helps cope with heterogeneous error causes, facilitates downstream remediation, and improves accuracy.

Our main contributions are as follows: 1) We incorporate the concept of knowledge tracing into the field of mistake finding by leveraging constructed student interaction sequences, enabling more personalized mistake finding. 2) For the output

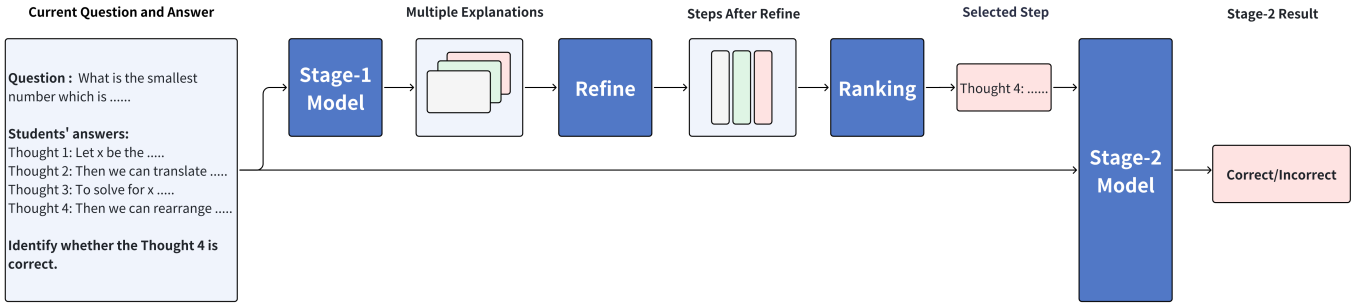

Fig. 2: Stage-2 model.

of mistake finding results, our method provides a probabilistic representation along with explanatory text, enhancing interpretability and facilitating practical deployment. 3) We develop a two-stage mistake finding framework: the first stage performs coarse-grained detection, and the second stage refines the results using outputs from the initial phase.

II. Related Work

a) Mistake Finding: Prompt-based mistake finding remains a challenge for current LLMs without external feedback [14]. Recent methods such as SelfCheck [9] invoke the LLM multiple times to detect errors, which improves accuracy to some extent. In the trained-model paradigm, prior work trains outcome-level or process-level reward models (ORM/PRM) to detect errors [15], [16]. For example, [14] several studies train models to output an error probability, and further enhances performance by requiring explanations. In terms of output, they typically provide only binary labels, which constrains downstream error correction. On the input side, LLM hallucinations exhibit correlations, yet prior work rarely leverages historical hallucination data. With regard to accuracy, previous methods often attain relatively low performance and are difficult to deploy in practice. Meanwhile, using more accurate reasoning models increases inference time, and thus harms timeliness.

b) Knowledge Tracing: Knowledge tracing(KT) is an essential task for tracing the knowledge states of each student separately, so that we can predict her performance on future exercising activities, where the basic idea is similar to the typical sequential behavior mining [13], [17]. While earlier methodologies leveraged Bayesian networks to model the student learning process [18], [19], the emergence of Deep Knowledge Tracing integrated recurrent neural networks into the KT task [20], [21]. Several studies [22]–[24] leverage textual information to perform knowledge tracing through LLMs. For mistake finding, knowledge tracing can serve as a tool. however, knowledge tracing primarily models latent knowledge states, whereas mistake finding focuses on whether the causal logic of the reasoning in the current case is correct. Consequently, pure knowledge tracing is often insufficient for mistake finding. In our method, knowledge tracing is used to help the model focus on possible errors in the current case.

III. Methodology

A. Stage-1 Model

As shown in Figure 1, we integrate knowledge tracing into the mistake-finding task. Given a math problem Q and the first i steps of a solution, denoted by $A_i = \{a_1, \ldots, a_i\}$, we model each simulated exercising sequence as follows. For a student $s \in \mathcal{S}$, the sequence is

$$\mathcal{S}_s = \big\{(Q_1, A_1^1, y_1^1, e_1^1), \ldots, (Q_n, A_i^n, y_i^n, e_i^n)\big\}, \tag{1}$$

where $Q_1, \ldots, Q_n$ are the questions attempted by the student, A_i^n represents the set of the first i steps for the n-th problem. $y_i^n \in \{\text{correct}, \text{incorrect}\}$ is the binary label of correctness for the i-th step on question n, and e_i^n is the explanation associated with y_i^n.

For each historical record, we form two components: the question with its solution steps and the correctness label with the explanatory text. We extract features for both components using the BGE-M3 [25] as encoder, and then map the extracted representations through a linear projection module to produce soft prompt embeddings for the LLM input [10], [11]. In the experiments each of the two text segments was mapped to a distinct token.

The linear projection module is a three-layer fully connected network that projects the 1024-dimensional BGE embedding into the target hidden space:

$$\mathbf{h}_{\text{out}} = W_3 \cdot \text{ReLU}\big(W_2 \cdot \text{ReLU}\big(W_1 \cdot \mathbf{h}_{\text{in}}\big)\big). \tag{2}$$

Here, $\mathbf{h}_{\text{in}}$ denotes the output of the feature extractor, and $\mathbf{h}_{\text{out}}$ is the resulting soft prompt embedding fed to the LLM. This approach follows established methodologies in recent research [11], [26].

Because mathematical problems and solutions often contain many numerals and formulas, the resulting tokenized inputs can be long. To optimize the representation of user history for low-latency inference, we map historical records to the LLM's embedding layer as soft prompts. Specifically, we encode the pair "(question, first i solution steps)" as $\mathbf{h}_i$, and the pair "(correctness label, explanation)" as $\mathbf{h}_f$. We also prepend a data-description soft prompt $P_h \in \mathbb{R}^{k \times h}$ to these embedding sequences, where k is the number of soft-prompt tokens and h is the LLM's embedding size. The training loss for a sample can be defined as follows:

$$\mathcal{L}_{\text{classify}} = \sum_{t=1}^{M_1} \log p_\theta\big(y_t^n \mid q^n, A_i^n, y_{<t}^n, \mathcal{S}_s^{n-1}\big), \tag{3}$$

Datasets	CoT methods	Avg. F1	Cls. Acc.	MF. Acc.	P_+	R_+	F_+	P_-	R_-	F_-
BIG-Bench Mistake	Direct-Step†	78.67	-	42.67	-	-	-	-	-	-
	Vanilla†	75.19	76.00	46.67	40.38	33.87	36.84	83.47	86.97	85.19
	Zero-shot CoT†	72.82	73.33	46.33	33.93	30.65	32.20	82.38	84.45	83.40
	Plan-and-Solve†	73.17	74.33	47.00	34.69	27.42	30.63	82.07	86.55	84.25
	SelfCheck†	88.37	88.33	81.00	71.43	72.58	72.00	92.83	92.44	92.63
	PedCoT†	88.81	89.33	83.00	81.25	62.90	70.91	90.87	96.22	93.47
	Qwen2.5-Math-PRM-7B	91.58	91.47	75.67	**100.00**	85.56	92.22	93.38	93.41	93.33
	Qwen2.5-Math-PRM-72B	94.64	94.50	85.67	**100.00**	87.04	93.07	96.98	96.95	96.96
	o1	95.98	95.87	**91.33**	**100.00**	86.30	92.64	99.03	99.02	99.03
	o3	95.81	95.69	**91.33**	**100.00**	85.56	92.22	99.03	99.02	99.03
	o4-mini	95.90	95.78	**91.33**	**100.00**	85.56	92.22	**99.15**	**99.15**	**99.15**
	gpt5	95.71	95.60	90.33	**100.00**	86.30	92.64	98.67	98.65	98.66
	stage1	88.72	88.62	70.00	**100.00**	84.81	91.78	89.75	89.88	89.67
	stage1 + stage2	**96.62**	**96.61**	91.00	**100.00**	**92.59**	**96.15**	97.78	97.84	97.80
PRM800K	Direct-Step†	67.10	66.00	33.33	42.20	54.12	47.42	79.58	70.70	74.88
	Vanilla†	73.23	73.00	37.00	52.22	55.29	53.71	81.90	80.00	80.94
	Zero-shot CoT†	74.89	74.00	41.33	53.15	69.41	60.20	86.24	75.81	80.69
	Plan-and-Solve†	74.79	74.00	38.00	53.27	67.06	59.37	85.49	76.74	80.88
	SelfCheck†	71.79	70.33	40.00	48.59	81.18	60.79	89.87	66.05	76.14
	PedCoT†	79.82	79.33	45.33	61.39	72.94	66.67	88.44	81.86	85.02
	Qwen2.5-Math-PRM-7B	91.76	92.20	56.00	**100.00**	**97.08**	**98.51**	88.78	87.94	87.81
	Qwen2.5-Math-PRM-72B	91.93	91.65	59.00	**100.00**	94.60	97.23	89.36	89.57	89.36
	o1	90.48	89.19	66.33	**100.00**	91.34	95.47	87.68	90.44	88.49
	o3	91.67	90.67	**67.33**	**100.00**	92.35	96.02	**91.31**	89.49	90.06
	o4-mini	90.81	89.65	66.33	**100.00**	92.23	95.96	90.37	87.83	88.60
	gpt5	89.92	88.40	62.67	**100.00**	90.44	94.98	90.50	86.97	87.95
	stage1	90.34	89.70	59.67	**100.00**	92.35	96.02	87.93	87.84	87.88
	stage1 + stage2	**92.91**	**92.80**	**68.00**	**100.00**	95.72	97.82	90.43	**90.76**	**90.55**

TABLE I: Comparison of various methods on the BIG-Bench Mistake and PRM800K datasets.

$$\mathcal{L}_{\text{explanation}} = \sum_{t=1}^{M_2} \log p_\theta\big(e_t^n \mid q^n, A_i^n, y^n, e_{<t}^n, \mathcal{S}_s^{n-1}\big), \quad (4)$$

$$\mathcal{L}_{\text{stage1}} = \mathcal{L}_{\text{classify}} \ + \ \lambda \mathcal{L}_{\text{explanation}}, \quad (5)$$

$\mathcal{L}_{\text{classify}}$ denotes the classification loss, and $\mathcal{L}_{\text{explanation}}$ the explanation loss. Where n is the number of training samples, for sample n, the input question is q^n, $y^n \in \{\text{correct}, \text{incorrect}\}$ is the ground-truth label. The M_1 is the total length of y and y_t^n is the previous tokens. The M_2 is the total length of e , e_t^n is the previous tokens. In practice, the output prefix can explicitly include "correct / incorrect, explanation: ". The hyperparameter $\lambda \geq 0$ balances the binary classification and the explanation-generation losses. In the experiment, λ was set to 1. $\mathcal{S}_s^{n-1}$ is the simulated exercising sequence before n.

At decoding time, we constrain the token at the classification position to one of the two tokens corresponding to "correct" and "incorrect", retrieve the probabilities of these tokens, and apply a softmax to obtain the class-confidence score. The model then generates the accompanying explanation. The output format is fixed: for a correct answer, "correct, explanation: ...", for an incorrect answer, "incorrect, explanation: ...". The probabilities corresponding to "correct" and "incorrect" are obtained and normalized via softmax to produce the confidence scores for the output categories, which can be calculated as follows:

$$p(\text{correct}) = \frac{\exp(l_c)}{\exp(l_c) + \exp(l_i)}. \quad (6)$$

Let $p(\text{correct})$ denote the probability of a step being correct. Let t denote the position of the classification label in the sequence. Let l_c and l_i be the logits output by the model for the tokens "correct" and "incorrect" at position t. In this experiment, t denotes the position of the first token in the output, and accordingly, our training data is structured to align with this textual configuration.

B. Stage-2 Model

As shown in Figure 2, in accordance with [27], [28], each step can be represented as a vector in a high-dimensional space, where incorrect steps are distributed within a certain neighborhood around the correct ones. With regard to the different steps pertaining to the same procedural objective, correct steps often follow similar reasoning patterns while incorrect steps exhibit diverse error types, we assume that correct steps form denser clusters in the vector space compared to incorrect steps. For the n-th step in the solution, multiple valid textual representations of correct steps may exist. For a student's step requiring evaluation in the current test case, we aim to select the most plausible correct step from the candidate set for comparison. This necessitates ranking the candidate steps. The confidence scores assigned by Model 1 to different steps are interpreted as the distance from each point to the classification hyperplane in the high-dimensional space [29], [30]. A higher score indicates a greater distance from the hyperplane, which corresponds to a larger separation from incorrect steps and facilitates more straightforward discrimination.

Following this approach, for the results output by the test set in Stage-1, we filter them by confidence level $p(\text{correct})$ and further process the cases with confidence levels $p(\text{correct})$ below the threshold in Stage-2. In this experiment, the confidence threshold was selected as 0.5. These low-confidence error cases are subsequently refined using a ReAct method [31], [32], where the explanations obtained from Stage-1 are used to prompt the LLM for correction. Specifically, each explanation yields one refined candidate step. In this experiment, we employ beam search at Stage-1 to generate five explanations, which in turn produce five refined candidate correct steps. For the i-th step a_i of A_i, we are given m candidate explanation e_{ij} from Stage-1, for $j = 1, 2, \ldots, m$. The refinement stage

Fine-Tuning	KT	BIG-Bench Mistake			PRM800K		
		F1	Acc.	MF.Acc.	F1	Acc.	MF.Acc.
✗	✗	77.24	76.33	39.33	80.36	76.98	26.33
✓	✗	85.12	84.40	60.33	88.93	87.94	31.33
✓	✓	88.72	88.62	70.00	90.34	89.70	59.67

TABLE II: Ablation experimental results of Stage-1.

integrates Q, A_i, and an explanation e_{ij} together with a prompt P into a single input $I = (Q, A_i, e_{ij}, P)$, which is then processed by an LLM to produce a refined version of step a_i, Here we use the Llama3.3-70B-Instruct model. This can be calculated as follows:

$$a_i^{\text{refined}_j} = \text{LLM}\big(Q, A_i, e_{ij}, P\big), \quad j = 1, 2, \ldots, m. \tag{7}$$

Subsequently, we reuse the Stage-1 model to select the best candidate from these refined steps. Each candidate step is concatenated with preceding steps and other relevant contextual information to form inputs compatible with the Stage-1 model's expected format. These inputs are then evaluated by the Stage-1 model to estimate confidence scores for correctness, and the candidate steps are ranked based on these scores. the candidate steps with the highest confidence level is selected as a_i^{selected}.

Next, we verify the correctness of the original step a_i of A_i once again. We provide the Stage-2 model with the problem Q and the prefix A_i, and the refined step a_i^{selected} as input to classify whether a_i is correct, which can be calculated as follows:

$$\mathcal{L}_{\text{stage-2}} = \sum_{t=1}^{M} \log p_\theta\big(y_t^n \mid q^n, A_i^n, y_{<t}^n, a_i^{\text{selected}}\big) \tag{8}$$

The M is the total length of y and y_t^n is the previous tokens.

We evaluate the correctness of the original step a_i by comparing it with the refined version. This approach transforms the assessment into a task similar to LLM-as-a-judge, which is more tractable than direct classification. If the original step a_i is correct, the explanation generated by the Stage-1 model will guide the LLM to either enrich the existing text with additional details or reformulate the problem-solving objectives associated with a_i during the refinement stage. These two scenarios can yield similar intermediate or final outcomes, allowing the Stage-2 model to recognize them accordingly. If the original step contains errors, the explanation from the Stage-1 model will guide the LLM to make corrections during refinement. When repair attempts inadvertently introduce new errors, the model ranks candidate fixes and selects the most accurate one. Furthermore, we restrict interventions to targets with a high probability of being erroneous, thereby minimizing the risk of introducing new errors during the repair process.

IV. Model Training

We leveraged the capabilities of GPT-4[1] to obtain explanations for error causes. Training and testing data were extracted from the BIG-Bench Mistake dataset and PRM800K

[1]gpt-4-0314

Length	BIG-Bench Mistake			PRM800K		
	F1	Acc.	MF.Acc.	F1	Acc.	MF.Acc.
0	85.12	84.40	60.33	88.93	87.94	31.33
1	87.96	88.07	66.66	89.69	88.68	60.67
3	88.27	88.34	67.33	89.70	88.63	61.67
6	88.72	88.62	70.00	90.34	89.70	59.67
10	88.29	88.44	68.33	90.55	89.93	60.33

TABLE III: Ablation study on the length of student interaction history in Stage-1.

(Lightman et al., 2023). The BIG-Bench Mistake dataset comprises five subsets. We selected the multistep arithmetic subset as the test set, while the remaining four subsets, Word Sorting, Tracking Shuffled Objects, Logical Deduction, and Dyck Languages, were used as the training set. The training set contains a total of 17,447 step-level test cases, with each step categorized as either correct or incorrect. We use Multistep Arithmetic as the test dataset. In total, 1,506 steps. PRM800K (Lightman et al., 2023) provides its own split of training and test sets, and labels are assigned as correct, neutral, or incorrect. To ensure balanced sampling, we randomly selected 25,000 step-level steps: 10,000 correct, 10,000 incorrect, and 5,000 neutral. For the ease of comparison, we reused the extracted sub-test set used in the pedCoT [8] paper as the evaluation benchmark and this test set only has two labels: correct and incorrect. In total, 3,736 steps.

We adopted a data synthesis method based on ReAct. Prompts were used to ask GPT-4 to predict each case, where each case consists of the problem statement and the previous i steps. The prediction includes both a correctness label and an associated explanation. If the predicted label differs from the label in the dataset, ChatGPT is used to refine the previous prediction based on this discrepancy. This process is iterated until the predicted label matches the annotated label. The resulting explanation is then paired with the corresponding problem, solution steps, and label to form the training data. Since both BIG-Bench Mistake and PRM800K are generated by large language models, these historical problems may contain the inherent knowledge limitations of those models. The training set can be viewed as a collection of exercises previously completed by a student. Due to the limited availability of real-world data with step-level labels and explanations, synthetic data was employed in our experiments. We acknowledge that verifying our findings with actual student interactions is a critical prerequisite for practical deployment in classroom settings.

V. Experiments

A. Experiment Setup

We utilize the Llama3.3-70B-Instruct model as the base language model in our experiments. We employ the Low-Rank Adaptation (QLoRA [33]) technique during both Stage-1 and Stage-2 fine-tuning. The query and value matrices in QLoRA are optimized with a rank of $r = 8$. We adhere to the default configuration settings of Llama3. The learning rate is initialized at 1×10^{-4} and gradually decayed to zero following a cosine annealing schedule. We set the batch size to 4 and

Length	Tokens		FLOPs Reduction
	Text	Soft Embedding	
1	683	302	2.26X
3	1449	306	4.74X
6	2598	312	8.32X
10	4130	320	12.91X

TABLE IV: Input token counts for soft embedding v.s. text prompt For Stage-1.

Num	BIG-Bench Mistake			PRM800K		
	F1	Acc.	MF.Acc.	F1	Acc.	MF.Acc.
1	91.29	91.19	77.33	92.51	92.34	66.33
3	94.02	93.82	81.66	92.91	92.80	68.00
5	96.62	96.61	91.00	92.91	92.80	68.00

TABLE V: Ablation study for the number of refinement steps.

the gradient accumulation steps to 8. Training is conducted for one epoch. We synthesize the student profiles by randomly sampling 6 prior interaction cases from the training corpus to serve as simulated history. Following previous work on mistake finding [5], [8], we employ the weighted average F1-score (denoted as F1), binary classification accuracy (denoted as Acc.), and mistake finding accuracy (denoted as MF. Acc.) as the evaluation metrics. MF. Acc. is the percentage of cases where a model either correctly identifies the exact step of the first logical error or correctly states that no mistakes are present. F1 and Acc. are more suited for step-level evaluation, whereas MF. Acc. operates at the trace level.

B. Main Results

As shown in Table 1, the baseline methods are primarily categorized into three types. The first category consists of methods based on Chain-of-Thought, denoted with a † symbol, which utilize evaluation metrics obtained from pedCOT. These metrics were evaluated using GPT-4. The second category includes PRM models, for which we selected the relatively recent models: Qwen2.5-Math-PRM-7B and Qwen2.5-Math-PRM-72B [34], represented as PRM-7B and PRM-72B in the table, respectively. The testing parameters followed the recommendations provided in the model descriptions, with a classification threshold set to 0.5. The third category involves OpenAI's reasoning model API(o1[2], o3[3], o4-mini[4], gpt5[5]), with the temperature set to 0 and reasoning_effort set to "high" during testing. The prompt used was identical to the Direct-Step Prompting described in [5].

Our Stage-2 model achieves competitive performance on F1 and Acc., outperforming prior methods and ChatGPT-style reasoning models on both datasets. On BIG-Bench Mistake, Stage-2 exceeds the best previous baseline by 0.64% (F1) and 0.74% (Acc.) , and reduces the number of erroneous cases by 19.67%. The MF. Acc. metric is 0.33% lower than the best previous baseline. On PRM800K, Stage-2 surpasses the best prior baseline by 0.98% (F1) and 0.60% (Acc.) and 0.67% (MF.Acc.) , and reduces erroneous cases by 7.69%. When errors occur during the intermediate calculations of a step while the final result of the step remains correct, it remains considerably challenging for the Stage-2 model to detect such errors within the step.

[2]o1-2024-12-17
[3]o3-2025-04-16
[4]o4-mini-2025-04-16
[5]gpt-5-2025-08-07

C. Analysis

As shown in Table 2, we performed an ablation study on the Knowledge tracing module in Stage-1. Without training the Llama3 model, the F1, Acc., and MF.Acc. scores declined to 77.24, 76.33, and 39.33 respectively on the BIG-Bench Mistake dataset, and to 80.36, 76.98, and 26.33 on the PRM800K dataset, indicating that training improved model's ability of mistake finding. Without Knowledge tracing, the F1, Acc., and MF.Acc. scores declined to 85.12, 84.40, and 60.33 respectively on the BIG-Bench Mistake dataset, and to 88.93, 87.94, and 31.33 on the PRM800K dataset, demonstrating its importance for mistake finding. These results confirm that knowledge tracing enhances mistake analysis by leveraging learners' historical interaction data.

Table 3 shows the results of the Stage-1 model on the BIG-Bench Mistake and for varying lengths of simulated exercising sequence. The performance curve shows a gradual ascent, confirming that longer sequences benefit the model, although the marginal gain decreases. As the length of the input sequence increases from 0 to 10, the f1 metric of the BIG-Bench Mistake dataset rises from 85.12 to 88.29, while the MF.Acc. metric increases from 60.33 to 68.33. The f1 metric of the PRM800K datasets rises from 88.93 to 90.55, and the MF.Acc. metric increases from 31.33 to 60.33. We also observe fluctuations in the overall metrics.

For Stage-1, we can approximate FLOPs using the heuristic $FLOPs \approx 6ND$ [11], [35], where N represents the number of LLM parameters, and D represents the total number of training tokens ($D = BSL$, with batch size B, training steps S, and input length L). The parameters of the embedding module are significantly smaller than those of the LLM and are thus neglected here. Employing this embedding module reduces computational cost compared to the text-prompt baseline. For instance, at a length of 10, using embeddings achieves a 12.9X reduction in FLOPs. Furthermore, at this length, the use of embeddings increases the number of input tokens by merely 6.67% compared to using no historical record. This indicates that incorporating user history via embeddings introduces negligible overhead relative to employing no historical data. On the PRM800K dataset, the activation probability of Stage-2 is 38%, while on the BIG-Bench Mistake dataset, it is 48%. The average activation rate across both datasets is 41%. While Stage-2 is computationally demanding, its selective activation driven by Stage-1's filtering mitigates excessive computational overhead.

As shown in Table 5, we performed an ablation study on the number of refinement steps in the Stage-2 model. On the BIG-Bench Mistake dataset, the overall metric improves as the step number increases. On the PRM800K dataset, performance initially improves but plateaus after a certain scale. For the

BIG-Bench Mistake dataset, the training data consisted of Word Sorting, Tracking Shuffled Objects, Logical Deduction, and Dyck Languages, while the test set was Multistep Arithmetic. Because the training and test sets exhibit a substantial distributional shift, the model's generated explanations have higher token-level entropy. Correct refinement steps tend to occur later (i.e., at larger step indices). As a result, when more refinement steps are allowed, evaluation metrics continue to improve over longer step budgets, and the overall rate of improvement is greater.

VI. Conclusion

We presented a coarse-to-fine two-stage framework for LLM mistake finding. Stage-1 performs coarse detection with constrained-decoding explanations and a token-distribution-derived confidence score, while Stage-2 employs targeted repair-and-rejudge for hard cases. We achieve competitive performance on BIG-Bench Mistake and PRM800K, while reducing token overhead, improving interpretability, and avoiding auxiliary classifiers. The approach is practical for educational settings. We look forward to further exploring the integration of reinforcement learning, among other approaches, within a wider range of scenarios in the future.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv e-prints*, pp. arXiv–2407, 2024.

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

[4] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, "Let's verify step by step," in *The Twelfth International Conference on Learning Representations*, 2023.

[5] G. Tyen, H. Mansoor, V. Cărbune, P. Chen, and T. Mak, "Llms cannot find reasoning errors, but can correct them given the error location," *arXiv preprint arXiv:2311.08516*, 2023.

[6] S. Farquhar, J. Kossen, L. Kuhn, and Y. Gal, "Detecting hallucinations in large language models using semantic entropy," *Nature*, vol. 630, no. 8017, pp. 625–630, 2024.

[7] S. Kirtania, P. Gupta, and A. Radhakirshna, "Logic-lm++: Multi-step refinement for symbolic formulations," *arXiv preprint arXiv:2407.02514*, 2024.

[8] Z. Jiang, H. Peng, S. Feng, F. Li, and D. Li, "Llms can find mathematical reasoning mistakes by pedagogical chain-of-thought," *arXiv preprint arXiv:2405.06705*, 2024.

[9] N. Miao, Y. W. Teh, and T. Rainforth, "Selfcheck: Using llms to zero-shot check their own step-by-step reasoning," *arXiv preprint arXiv:2308.00436*, 2023.

[10] S. Doddapaneni, K. Sayana, A. Jash, S. Sodhi, and D. Kuzmin, "User embedding model for personalized language prompting," *arXiv preprint arXiv:2401.04858*, 2024.

[11] L. Ning, L. Liu, J. Wu, N. Wu, D. Berlowitz, S. Prakash, B. Green, S. O'Banion, and J. Xie, "User-llm: Efficient llm contextualization with user embeddings," in *Companion Proceedings of the ACM on Web Conference 2025*, 2025, pp. 1219–1223.

[12] K. Feng, C. Li, X. Zhang, J. Zhou, Y. Yuan, and G. Wang, "Keypoint-based progressive chain-of-thought distillation for llms," *arXiv preprint arXiv:2405.16064*, 2024.

[13] Q. Liu, Z. Huang, Y. Yin, E. Chen, H. Xiong, Y. Su, and G. Hu, "Ekt: Exercise-aware knowledge tracing for student performance prediction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 1, pp. 100–115, 2019.

[14] B. Gao, Z. Cai, R. Xu, P. Wang, C. Zheng, R. Lin, K. Lu, D. Liu, C. Zhou, W. Xiao *et al.*, "Llm critics help catch bugs in mathematics: Towards a better mathematical verifier with natural language feedback," *arXiv preprint arXiv:2406.14024*, 2024.

[15] P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui, "Math-shepherd: Verify and reinforce llms step-by-step without human annotations," *arXiv preprint arXiv:2312.08935*, 2023.

[16] F. Yu, A. Gao, and B. Wang, "Ovm, outcome-supervised value models for planning in mathematical reasoning," *arXiv preprint arXiv:2311.09724*, 2023.

[17] L. Fu, H. Guan, K. Du, J. Lin, W. Xia, W. Zhang, R. Tang, Y. Wang, and Y. Yu, "Sinkt: A structure-aware inductive knowledge tracing model with large language model," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 2024, pp. 632–642.

[18] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User modeling and user-adapted interaction*, vol. 4, no. 4, pp. 253–278, 1994.

[19] T. Kaser, S. Klingler, A. G. Schwing, and M. Gross, "Dynamic bayesian networks for student modeling," *IEEE Transactions on Learning Technologies*, pp. 1–1, 2017.

[20] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein, "Deep knowledge tracing," *Advances in neural information processing systems*, vol. 28, 2015.

[21] A. Ghosh, N. Heffernan, and A. S. Lan, "Context-aware attentive knowledge tracing," in *Knowledge Discovery and Data Mining*, 2020.

[22] A. Scarlatos, R. S. Baker, and A. Lan, "Exploring knowledge tracing in tutor-student dialogues using llms," in *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, 2025, pp. 249–259.

[23] J. Kim, S. Chu, B. Wong, and M. Yi, "Not all options are created equal: Textual option weighting for token-efficient llm-based knowledge tracing," *arXiv preprint arXiv:2410.12872*, 2024.

[24] U. Lee, J. Y. Kim, R. Ju, M. Jung, and J. Eo, "A training-free large reasoning model-based knowledge tracing framework for unified prediction and prescription," *arXiv preprint arXiv:2601.01708*, 2026.

[25] M.-L. M.-F. Multi-Granularity, "M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," 2024.

[26] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, pp. 34 892–34 916, 2023.

[27] X. Qiu and R. Miikkulainen, "Semantic density: Uncertainty quantification for large language models through confidence measurement in semantic space," *Advances in neural information processing systems*, vol. 37, pp. 134 507–134 533, 2024.

[28] B. Li, G. Deng, R. Chen, J. Yue, S. Zhang, Q. Zhao, L. Song, and L. Wen, "Rema: A unified reasoning manifold framework for interpreting large language model," *arXiv preprint arXiv:2509.22518*, 2025.

[29] T. A. Dinh and J. Niehues, "Are generative models underconfident? better quality estimation with boosted model probability," *arXiv preprint arXiv:2502.11115*, 2025.

[30] S. Kadavath, T. Conerly, A. Askell, T. Henighan, D. Drain, E. Perez, N. Schiefer, Z. Hatfield-Dodds, N. DasSarma, E. Tran-Johnson *et al.*, "Language models (mostly) know what they know," *arXiv preprint arXiv:2207.05221*, 2022.

[31] M. Wadhwa, X. Zhao, J. J. Li, and G. Durrett, "Learning to refine with fine-grained natural language feedback," *arXiv preprint arXiv:2407.02397*, 2024.

[32] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang *et al.*, "Self-refine: Iterative refinement with self-feedback," *Advances in Neural Information Processing Systems*, vol. 36, pp. 46 534–46 594, 2023.

[33] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in neural information processing systems*, vol. 36, pp. 10 088–10 115, 2023.

[34] Z. Zhang, C. Zheng, Y. Wu, B. Zhang, R. Lin, B. Yu, D. Liu, J. Zhou, and J. Lin, "The lessons of developing process reward models in mathematical reasoning," *arXiv preprint arXiv:2501.07301*, 2025.

[35] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.