Scalable Token-Level Hallucination Detection in Large Language Models

Rui Min[1,2*] **Tianyu Pang**[1†] **Chao Du**[1] **Minhao Cheng**[3†] **Yi R. Fung**[2†]
[1]Sea AI Lab
[2]Hong Kong University of Science and Technology
[3]Pennsylvania State University

Abstract

Large language models (LLMs) have demonstrated remarkable capabilities, but they still frequently produce hallucinations. These hallucinations are difficult to detect in reasoning-intensive tasks, where the content appears coherent but contains errors like logical flaws and unreliable intermediate results. While step-level analysis is commonly used to detect internal hallucinations, it suffers from limited granularity and poor scalability due to its reliance on step segmentation. To address these limitations, we propose **TOKENHD**, a holistic pipeline for training token-level hallucination detectors. Specifically, TOKENHD consists of a scalable data engine for synthesizing large-scale hallucination annotations along with a training recipe featuring an importance-weighted strategy for robust model training. To systematically assess the detection performance, we also provide a rigorous evaluation protocol. Through training within TOKENHD, our detector operates directly on free-form text to identify hallucinations, eliminating the need for predefined step segmentation or additional text reformatting. Our experiments show that even a small detector (0.6B) achieves substantial performance gains after training, surpassing much larger reasoning models (e.g., QwQ-32B), and detection performance scales consistently with model size from 0.6B to 8B. Finally, we show that our detector can generalize well across diverse practical scenarios and explore strategies to further enhance its cross-domain generalization capability. **Code** is publicly available to reproduce all experiments.

1 Introduction

Large language models (LLMs) have achieved remarkable performance on reasoning-intensive tasks, such as solving complex mathematical problems [1, 2, 3] and code generation [4, 5]. Nevertheless, LLMs still suffer from hallucinations [6, 7, 8]: generated content may appear coherent yet contain factual errors or logical inconsistencies, making such errors difficult to detect and undermining response reliability.

To address these challenges, various post-hoc detection methods have been proposed to scrutinize the truthfulness of LLM-generated content. Earlier research primarily focused on factual hallucinations [9, 10, 11], such as detecting contradictions between generated statements and trusted knowledge sources. However, with the emergence of retrieval-augmented generation (RAG) systems [12] and the integration of search tools [13], these knowledge-based hallucinations can be largely mitigated. In contrast, hallucinations in reasoning-intensive tasks (e.g., mathematics) often manifest as subtle logical errors or incorrect intermediate results, making them significantly harder to detect. To locate

*The project was done during Rui Min's internship at Sea AI Lab.
†Correspondence to: Tianyu Pang <tianyupang@sea.com>, Minhao Cheng <mmc7149@psu.edu>, and Yi R. Fung <yrfung@ust.hk>.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

these errors, a common practice is to decompose a solution into individual steps and train a detector with step-level supervision. Specifically, Process Reward Models (PRMs) [14, 15, 16] adopt this paradigm by assigning a correctness label to each step, thereby pinpointing where the internal reasoning process goes wrong.

Nevertheless, PRMs face several limitations because they necessitate explicit step segmentation, which is difficult when model outputs are free-form or lack clear separation boundaries. Furthermore, they are inherently restricted to step-level analysis, lacking fine-grained and flexible hallucination localization. While search-based methods such as Monte Carlo Tree Search (MCTS) [15, 17, 16] can estimate intermediate correctness via sampling statistics, they incur prohibitive computational overhead from intensive policy-model queries, limiting scalability. To address these challenges, we shift our focus to the atomic units of text by proposing TOKENHD, a holistic framework for token-level hallucination detection. Unlike PRMs, TOKENHD operates directly on free-form text and evaluates hallucination for each token. This design enables the precise location of hallucinations while substantially reducing inference latency, since our detector assigns scores directly without the need to generate verbal analysis. Figure 1 shows an example of our detection mechanism in a mathematical task.

The proposed TOKENHD comprises a scalable data synthesis engine and a specialized training recipe. To overcome the scarcity of token-level hallucination labels in existing datasets, we first develop a data synthesis pipeline to obtain high-quality samples with token-level annotations. Specifically, for each candidate sample, we prompt multiple critic models to identify hallucinated text fragments. These fragments are refined through a text restoration process, converted into token space, and merged into a single label sequence through averaging. To improve the aggregation quality beyond simple averaging, we further design an adaptive ensemble strategy that optimizes weights for each critic model to perform weighted aggregation. Leveraging these token-level supervisions, we train our detector using an importance-weighted strategy specifically designed to address the sparsity of hallucinated tokens. We also establish a rigorous evaluation protocol to assess detection performance and conduct extensive experiments on mathematical and STEM benchmarks. Finally, we show that our detector can generalize well across diverse task domains and policy models, while also exploring strategies to enhance its cross-domain generalization capability.

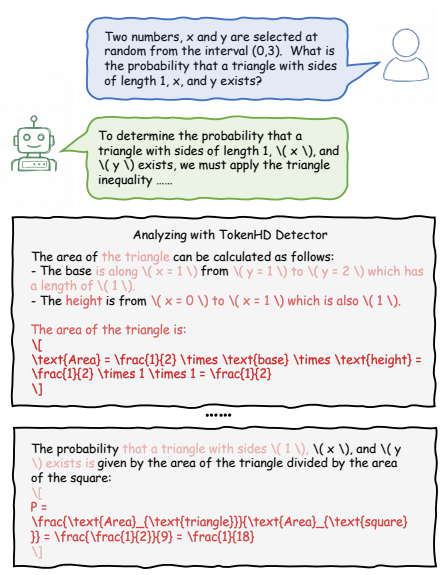


Figure 1: An illustration of the token-level detection mechanism of **TOKENHD**. Our detector identifies hallucinations directly in free-form text without predefined step segmentation. Color intensity reflects predicted hallucination probability: **deeper highlights** indicate higher likelihood, lighter highlights lower likelihood.

2 Preliminaries

We first introduce the basic concepts of tokenization and detokenization. We then formalize our token-level hallucination detection task, and finally describe the evaluation metrics used to assess detection performance.

Tokenization and detokenization. Given an input query $\mathbf{x}$, a policy model $\pi(\cdot)$ generates an output $\mathbf{y} = \pi(\mathbf{x})$. The tokenization function $\tau(\cdot)$ maps $\mathbf{y}$ to a token sequence $\mathbf{t} = \tau(\mathbf{y}) = (t_i)_{i=1}^{|\tau(\mathbf{y})|}$ with length $|\tau(\mathbf{y})|$, where $t_i \in \{0, 1, \ldots, |\mathcal{V}| - 1\}$ and $\mathcal{V}$ is the vocabulary. Conversely, the detokenization function $\tau^{-1}(\cdot)$ maps each token to a string fragment $\tau^{-1}(t_i)$, and the reconstructed text is obtained by string concatenation: $\mathbf{y} = \texttt{concat}_{i=1}^{|\tau(\mathbf{y})|}\tau^{-1}(t_i)$.

Token-level hallucination detection. We define token-level hallucination detection as predicting a hallucination score for each token position in $\mathbf{t}$. Let $\mathcal{H}(\cdot)$ denote a hallucination detector; it outputs a score sequence $\widehat{\mathbf{s}} = \mathcal{H}(\mathbf{x}, \mathbf{y}) = (\widehat{s}_i)_{i=1}^{|\tau(\mathbf{y})|}$, where $\widehat{s}_i \in [0, 1]$ and larger values indicate a higher likelihood of hallucination. Since $\widehat{s}_i$ is a continuous score, to map the predictions back to text fragments, we introduce a threshold $\beta_{\widehat{I}}$ and define the predicted hallucinated token indexes as $\widehat{I} = \{i \mid \widehat{s}_i > \beta_{\widehat{I}}\}$. We then group consecutive indices in $\widehat{I}$ into M segments $\{\widehat{I}_m\}_{m=1}^{M}$, and map each segment back to its corresponding text span $\widehat{r}_m = \texttt{concat}_{i \in \widehat{I}_m} \tau^{-1}(t_i)$.

Evaluation metrics. To evaluate our hallucination detector, we compare the predicted token-level scores $\widehat{\mathbf{s}}$ against the ground-truth annotations $\mathbf{s} = (s_i)_{i=1}^{|\tau(\mathbf{y})|}$. The ground-truth $\mathbf{s}$ is provided by labeler models, typically high-capacity models (e.g., GPT-5). Due to their expensive inference costs, these models are solely reserved for annotating small-scale evaluation samples. For ease of evaluation, we binarize the ground-truth scores with a threshold β_I, yielding ground-truth hallucinated token indexes $I = \{i \mid s_i > \beta_I\}$. We then compute token-level precision, recall, and F_1 as $\mathrm{Precision} = \frac{|\widehat{I} \cap I|}{|\widehat{I}|}$, $\mathrm{Recall} = \frac{|\widehat{I} \cap I|}{|I|}$, and $\mathrm{F}_1 = \frac{2\,\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$. These token-level metrics provide a fine-grained measure of detection performance and serve as our primary evaluation metrics.

3 The TOKENHD framework

We present the core idea of our TOKENHD framework, including how we obtain samples with token-level hallucination annotations and the techniques used for training detectors. In Section 3.1, we describe how we obtain our token-level hallucination annotations to construct the dataset. Then, in Section 3.2, we introduce an ensemble strategy that adaptively and optimally aggregates diverse sources of annotations to improve the annotation quality. Finally, we detail how we leverage the annotations to train the token-level hallucination detector in Section 3.3.

3.1 Token-level hallucination annotations

We start by introducing our data construction process. While labeler models offer high-quality annotations, their prohibitive inference costs make them impractical for large-scale data generation. Therefore, we instead employ more cost-effective critic models $\mathcal{V}(\cdot)$ to produce our training data. Given an input query $\mathbf{x}$ and the corresponding output $\mathbf{y} = \pi(\mathbf{x})$, we prompt a critic model $\mathcal{V}(\cdot)$ to identify hallucinated text fragments in $\mathbf{y}$. The model returns a sequence of M hallucinated text fragments $\mathcal{V}(\mathbf{x}, \mathbf{y}) = (r_m^{\mathcal{V}})_{m=1}^{M}$. Note that $M = 0$ if no hallucinated content is identified by the critic model, which corresponds to an all-zero annotation sequence. In practice, raw annotations may be paraphrased or slightly expanded, making direct token-level alignment difficult. To address this, we design a restoration process. We prompt an LLM to iteratively refine the raw fragments so that each hallucinated fragment corresponds to an exact span in the original $\mathbf{y}$. Experiments demonstrate that our strategy achieves 98.10% recovery performance on average across critics. We provide details of the annotation process and our restoration algorithm in Appendix B. After restoration, we convert the restored fragment-level annotations into token-level annotations over the token sequence $\tau(\mathbf{y}) = (t_i)_{i=1}^{|\tau(\mathbf{y})|}$. Specifically, we map each token t_i to its detokenized string fragment $\tau^{-1}(t_i)$ in $\mathbf{y}$, and assign a binary hallucination label $a_i^{\mathcal{V}} \in \{0, 1\}$. We denote $\mathcal{A}_{\mathcal{V}}$ as the hallucination annotations produced by critic model $\mathcal{V}$:

$$\begin{aligned} \mathcal{A}_{\mathcal{V}} &= \left\{(t_i, a_i^{\mathcal{V}})\right\}_{i=1}^{|\tau(\mathbf{y})|}, \\ a_i^{\mathcal{V}} &= \begin{cases} 1, & \text{if } \tau^{-1}(t_i) \text{ overlaps with } r_m^{\mathcal{V}} \text{ for some } m, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{1}$$

Here, "overlaps" means that the text spans of $\tau^{-1}(t_i)$ and $r_m^{\mathcal{V}}$ in $\mathbf{y}$ have a non-empty intersection. And when there are no hallucinations within $\mathbf{y}$, i.e., $M = 0$, the annotations reduce to $a_i^{\mathcal{V}} = 0$ for all i. To obtain more reliable token-level hallucination labels and mitigate the variance from a single critique, we perform C critiques for each output $\mathbf{y}$ using the same critic model $\mathcal{V}$. For each critique $c \in \{1, \ldots, C\}$, we obtain the corresponding token-level binary annotations $\mathcal{A}_{\mathcal{V}}^{(c)} =$

$\{(t_i, a_i^{\mathcal{V},(c)})\}_{i=1}^{|\tau(\mathbf{y})|}$ as defined in Eq. 1. We then aggregate these C binary annotations into soft hallucination scores by averaging across critiques, denoted as $\bar{a}_i^{\mathcal{V}} = \frac{1}{C}\sum_{c=1}^{C} a_i^{\mathcal{V},(c)}$ and obtain the annotations $\bar{\mathcal{A}}_{\mathcal{V}} = \{(t_i, \bar{a}_i^{\mathcal{V}})\}_{i=1}^{|\tau(\mathbf{y})|}$.

3.2 Ensemble from diverse annotations

While performing multiple critiques helps reduce variance, relying solely on a single critic model may still make the annotations strongly affected by the model's capability and preference. In practice, we can leverage multiple models to detect hallucinations, reducing the risk that the bias of any critic model overly influences the annotations. Given a set of K diverse critic models $\{\mathcal{V}_k\}_{k=1}^{K}$, the hallucination annotations derived from each critic model are $\bar{\mathcal{A}}_{\mathcal{V}_k} = \{(t_i, \bar{a}_i^{\mathcal{V}_k})\}_{i=1}^{|\tau(\mathbf{y})|}$. We therefore need to aggregate these annotations into a single label sequence. To address this, we consider two ensemble strategies:

Uniform ensemble. A straightforward strategy is to average the hallucination scores across all K critic models for each token, which is defined as: $\bar{\mathcal{A}}_{\text{avg}} = \left\{\left(t_i, \frac{1}{K}\sum_{k=1}^{K} \bar{a}_i^{\mathcal{V}_k}\right)\right\}_{i=1}^{|\tau(\mathbf{y})|}$. This simple scheme directly averages the annotations from different critic models by treating them equally. However, these models can differ in their annotation capability, and directly averaging their hallucination scores may result in a "barrel effect", where weaker critic models disproportionately degrade the quality of the averaged annotations. This motivates us to further consider an adaptive ensemble mechanism that accounts for the different annotation capabilities of individual critic models.

Adaptive ensemble. We introduce an adaptive ensemble strategy that assigns a learnable weight to each critic model. Let $\mathbf{w} = (w_1, \ldots, w_K)$ denote the weight vector, where w_k indicates the contribution of $\mathcal{V}_k$. We learn $\mathbf{w}$ on a validation set $\mathcal{D}_{\text{val}}$. For each validation sample $(\mathbf{x}, \mathbf{y}, \mathbf{s}) \in \mathcal{D}_{\text{val}}$, where $\mathbf{s}$ indicates ground-truth hallucination scores, we first leverage the K critic models to obtain their annotation scores $\{(t_i, \bar{a}_i^{\mathcal{V}_k})\}_{i=1}^{|\mathbf{t}|}$ respectively. We then optimize $\mathbf{w}$ by minimizing the Mean Squared Error (MSE) between the ensembled annotation scores and the ground-truth scores:

$$\mathcal{L}_{\text{e}}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},\mathbf{y},\mathbf{s})\sim\mathcal{D}_{\text{val}}}\left[\frac{1}{|\tau(\mathbf{y})|}\sum_{i=1}^{|\tau(\mathbf{y})|}\left(s_i - \sum_{k=1}^{K} w_k\, \bar{a}_i^{\mathcal{V}_k}\right)^2\right], \quad \text{s.t.} \quad \sum_{k=1}^{K} w_k = 1. \tag{2}$$

Using the optimized weights $\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}_{\text{e}}(\mathbf{w})$, we compute the ensembled hallucination score for each token t_i as $\sum_{k=1}^{K} w_k^* \bar{a}_i^{\mathcal{V}_k}$, and obtain the adaptively ensembled annotation scores as: $\bar{\mathcal{A}}_{\text{adapt}} = \left\{\left(t_i, \sum_{k=1}^{K} w_k^*\, \bar{a}_i^{\mathcal{V}_k}\right)\right\}_{i=1}^{|\tau(\mathbf{y})|}$.

Compared with the uniform ensemble, this adaptive scheme assigns smaller weights to critic models whose annotations are less consistent with the ground-truth scores on the validation set, thereby reducing the influence of weaker models and making the final annotations more robust. For simplicity, we define $\bar{\mathbf{a}}^{\text{ens}} = (\bar{a}_i^{\text{ens}})_{i=1}^{|\tau(\mathbf{y})|}$ as the ensembled hallucination score under the chosen strategy.

3.3 Training recipe for the hallucination detector

TOKENHD is built upon a causal transformer with a shared linear classification head. Given a query and response, the head maps each response token's hidden state $\mathbf{h}_i$ to a hallucination score $\widehat{s}_i = \sigma(\mathbf{w}^\top \mathbf{h}_i + b)$, where $\mathbf{w}$ and b are learned parameters and σ is the sigmoid function. The detector scores all response tokens in a single forward pass without step segmentation.

After ensembling annotation scores from diverse critic models, we construct training examples and train the hallucination detector $\mathcal{H}(\cdot)$ on the training set $\mathcal{D}_{\text{train}}$. For each token position i, we use the ensembled score $\bar{a}_i^{\text{ens}}$ as supervision and introduce two training schemes.

Standard training. We first consider a straightforward strategy to optimize the detector using standard cross-entropy loss. For each training example with input query $\mathbf{x}$ and output $\mathbf{y}$, the detector outputs token-level scores $\widehat{\mathbf{s}} = \mathcal{H}(\mathbf{x}, \mathbf{y}) = (\widehat{s}_i)_{i=1}^{|\tau(\mathbf{y})|}$, and we minimize $L = \mathbb{E}_{(\mathbf{x},\mathbf{y},\bar{\mathbf{a}}^{\text{ens}})\sim\mathcal{D}_{\text{train}}}\left[\frac{1}{|\tau(\mathbf{y})|}\sum_{i=1}^{|\tau(\mathbf{y})|} \ell_s\big(\widehat{s}_i, \bar{a}_i^{\text{ens}}\big)\right]$, where $\ell_s\big(\widehat{s}_i, \bar{a}_i^{\text{ens}}\big) = -\bar{a}_i^{\text{ens}} \log \widehat{s}_i - (1 - \bar{a}_i^{\text{ens}}) \log(1 - \widehat{s}_i)$. This training objective allows the detector to learn directly from the soft scores.

Table 1: Detection performance of various models across mathematical benchmarks. Our TOKENHD variants are trained from their corresponding backbone models (e.g., TOKENHD-0.6B is trained based on Qwen3-0.6B). We report the average S_{incor} / S_{cor}, respectively.

Category	Model Name	Math-500	AIME-2024	AIME-2025	Olym-Math
Labeler Models	GPT-5	86.68 / 99.68	84.01 / 100.0	87.89 / 100.0	84.94 / 99.59
	o3	82.03 / 99.86	80.44 / 100.0	85.88 / 100.0	79.36 / 99.77
Critic Models	R1-Qwen3-8B	43.95 / 99.80	34.51 / 100.0	40.14 / 100.0	39.93 / 99.75
	QwQ-32B	55.05 / 99.78	50.59 / 100.0	56.18 / 99.31	52.53 / 99.61
	GPT-4.1	46.60 / 98.56	50.20 / 99.37	56.24 / 100.0	49.80 / 96.99
	o4-mini	71.39 / 99.93	67.67 / 100.0	76.09 / 100.0	70.32 / 99.80
Backbone Models	Qwen3-0.6B	9.86 / 98.94	9.29 / 100.0	6.46 / 95.84	8.42 / 98.37
	Qwen3-1.7B	24.00 / 99.93	18.99 / 100.0	18.76 / 100.0	19.45 / 99.82
	Qwen3-4B	42.23 / 99.91	36.78 / 100.0	44.75 / 100.0	39.39 / 99.86
	Qwen3-8B	49.45 / 99.84	47.03 / 100.0	48.92 / 99.72	46.02 / 99.76
Detector Models	TOKENHD-0.6B	63.64 / 94.83	61.41 / 90.12	64.34 / 89.31	64.99 / 93.21
	TOKENHD-1.7B	64.39 / 98.06	59.57 / 96.79	67.02 / 96.41	65.63 / 96.68
	TOKENHD-4B	64.61 / 98.10	63.70 / 96.62	67.32 / 92.66	66.95 / 96.54
	TOKENHD-8B	64.55 / 98.71	63.62 / 98.44	68.88 / 95.84	68.45 / 97.61

Importance-weighted training. However, after curating the training examples, we find that hallucinated tokens with high hallucination scores are sparse across the dataset, leading to label imbalance and biasing the detector toward predicting lower hallucination scores. To address this, we propose a reweighting scheme to our original training objective. Let `pos_weight` be the proportion of tokens with hallucination scores less than or equal to a threshold β, and let `neg_weight` be the proportion of tokens with scores greater than β, i.e., $\texttt{pos_weight} = \frac{\left|\{\, i | \bar{a}_i^{\text{ens}} \leq \beta \,\}\right|}{|\tau(\mathbf{y})|}$ and $\texttt{neg_weight} = \frac{\left|\{\, i | \bar{a}_i^{\text{ens}} > \beta \,\}\right|}{|\tau(\mathbf{y})|}$. We then define the following weighted token-level cross-entropy loss:

$$\ell_i(\widehat{s}_i, \bar{a}_i^{\text{ens}}) = -\Big[\texttt{pos_weight} \cdot \bar{a}_i^{\text{ens}} \log \widehat{s}_i + \texttt{neg_weight} \cdot (1 - \bar{a}_i^{\text{ens}}) \log(1 - \widehat{s}_i)\Big]. \tag{3}$$

This strategy increases the contribution of hallucinated tokens during training, which effectively mitigates the challenge of label sparsity and stabilizes the training process.

4 Evaluating the effectiveness of TOKENHD

4.1 Experimental settings

Backbone models. To ensure our detector remains efficient for practical use, we focus on small-scale backbones from the Qwen3 series [18], with sizes ranging from 0.6B to 8B. Since the detector is primarily designed as a post-hoc hallucination detection module for much larger LLM systems, low inference latency and deployment costs are critical. While performance scales with model size across this range, even the smallest 0.6B variant substantially outperforms larger reasoning models such as QwQ-32B, confirming that a lightweight detector is practically competitive.

Training settings. We initially train our detector on mathematical tasks and explore its transferability to other domains, such as code generation, in later sections. The training data is curated from three primary sources: Math [19], and subsets of AceReason-Math [20] and Big-Math [21], providing problems with various difficulty levels. To generate the training samples, we use GPT-4o-mini [22] as our policy model and sample two responses for each prompt. For hallucination annotation, we use four critic models: DeepSeek-R1-0528-Qwen3-8B (R1-Qwen3-8B) [2], QwQ-32B [23], GPT-4.1 [24], and o4-mini [25] to annotate each training sample independently, and then aggregate their outputs into a single token-level label sequence using the adaptive ensemble strategy. To improve data quality, we apply several filtering strategies. For instance, we discard samples with incorrect final answers even if the critic models identify no hallucinated tokens. Detector training is conducted on a single node with 8× NVIDIA A100 GPUs. We defer more training details, such as the training data composition and the choice of hyperparameters to Appendix C.

Evaluation protocols. We primarily evaluate our detector on four widely used mathematical benchmarks, including Math-500 [19], AIME-2024 [26], AIME-2025 [27], and OlympiadBench-

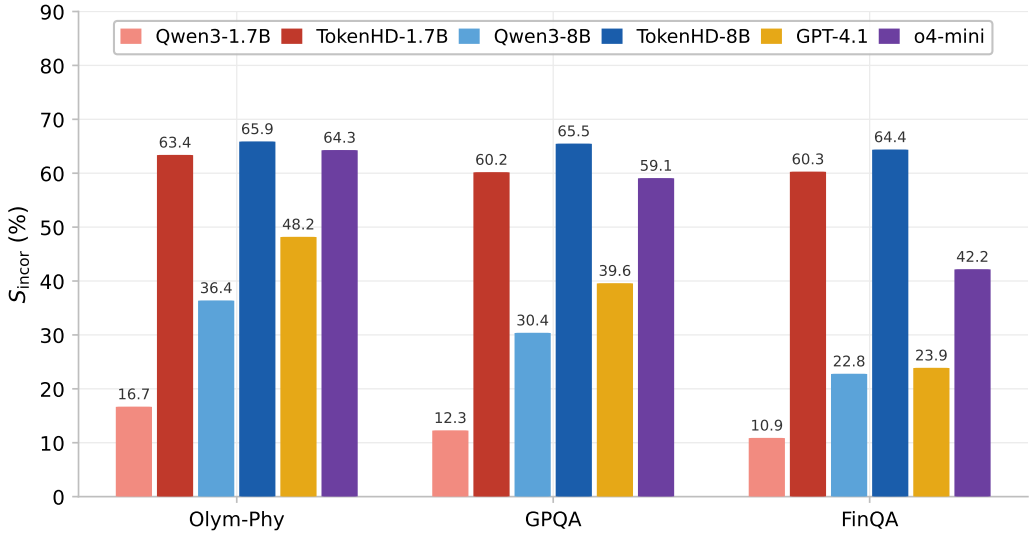


Figure 2: We report S_{incor} across three STEM benchmarks. Qwen3-1.7/8B are backbone models, GPT-4.1 and o4-mini are critic models, and TOKENHD-1.7/8B are our trained hallucination detectors.

Math [28], and later extend our evaluation to additional STEM domains, including GPQA [29], OlympiadBench-Phy [28], and FinQA [30]. To generate evaluation samples, we use the same policy model as in the training data generation, GPT-4o-mini (we discuss the impact of different policy models in Section 5.1), and generate two responses for each prompt. We then obtain ground-truth hallucination annotations using two labeler models, GPT-5 [31] and o3 [25], and use the uniform ensemble strategy for final aggregation. Following the annotation results, we categorize the evaluation samples into two sets: hallucinated samples, containing tokens identified by the labeler models, and non-hallucinated samples, which are hallucination-free and conclude with a correct answer. We employ token-level F1 (defined in Section 2) as the detection metric for the hallucinated samples, denoted as S_{incor}. For non-hallucinated samples, we invert both the ground-truth and predicted labels and report the recall of non-hallucinated tokens as $S_{\text{cor}} = 1 - \frac{|\widehat{I}|}{|\tau(\mathbf{y})|}$. We use this definition for all methods. Both metrics are expressed as percentages, and we report their values as plain numbers throughout this paper. In our experiments, we set $\beta_I = 0.5$ as the default to generate binary ground-truth labels. For the predicted labels, we use the same threshold $\beta_{\widehat{I}} = 0.5$, ensuring a consistent and fair comparison across all models. For comparison, we also report the performance of critic and labeler models by treating their individual annotations as predictions and evaluating them against the aggregated ground-truth labels. We also verify the annotation quality through human evaluation in Appendix D, where human raters assigned high-quality scores of 4.63/5 for math and 4.55/5 for code.

4.2 Experimental results

Effectiveness of token-level hallucination detection. As shown in Table 1, our detectors demonstrate strong performance on the four mathematical benchmarks. We highlight three critical findings from these results. First, our detectors consistently outperform their corresponding backbone models while significantly reducing inference overhead. Unlike critic models, each detector directly outputs token-level scores without generating any reasoning text. Second, although our detector is trained on data labeled by the critic models, it still surpasses most of them, including GPT-4.1, R1-Qwen3-8B, and QwQ-32B, and achieves competitive performance against o4-mini. Notably, our smallest TOKENHD-0.6B significantly outperforms QwQ-32B on S_{incor} across all benchmarks. Third, detection performance scales consistently with model size, with 1.7B–8B variants consistently achieving $S_{\text{cor}} \geq 92$. These results demonstrate that our training framework enables lightweight models to match much larger reasoning models on this task (see Appendix E). We also provide comparisons against PRM baselines in Appendix F, where TOKENHD-8B achieves a 17.90% relative improvement in S_{incor} over the much larger Qwen2.5-Math-PRM-72B on Math-500. In Appendix G,

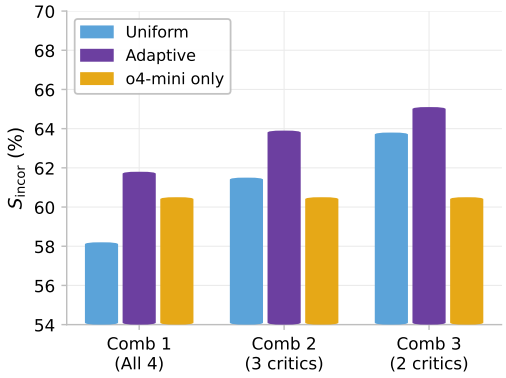


Figure 3: Detection performance under two ensemble strategies.

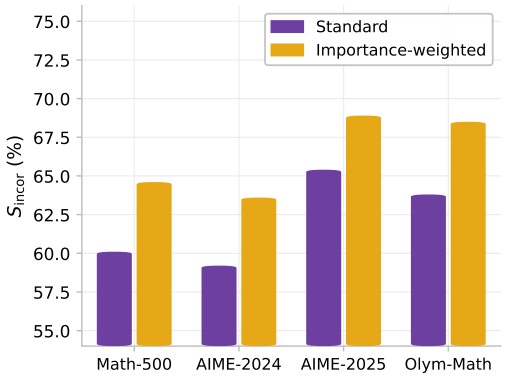


Figure 4: Detection performance under two training strategies.

Table 2: Performance comparison between uniform and adaptive ensemble strategies. We report the average S_{incor} / S_{cor} of ensembled labels from different combinations of critic models.

Ensemble Strategy	**Comb 1**	**Comb 2**	**Comb 3**
Uniform	60.21 / 100.0	66.75 / 99.99	69.97 / 99.98
Adaptive	78.16 / 99.99	78.76 / 99.99	78.77 / 99.99

we report AUROC and AUPRC as metrics, and our results further confirm that TOKENHD's scores provide strong threshold-independent discrimination, substantially surpassing all critic models.

Generalization across STEM task domains. While previous results show that our detector performs well on pure mathematical tasks, many real-world questions require reasoning in other STEM domains, such as science and finance. We therefore collect prompts from GPQA, OlympiadBench-Phy, and FinQA to construct a more comprehensive set of evaluation samples. Since our detector has already demonstrated superior performance over the two relatively weak critic models: R1-Qwen3-8B and QwQ-32B, we only report the performance of GPT-4.1 and o4-mini as a reference. As shown in Figure 2, our detector consistently achieves substantial improvements in hallucination detection over the backbone model across all benchmarks. Similar to the mathematical tasks, our detector outperforms GPT-4.1 and demonstrates competitive performance against o4-mini. This underscores the robustness of our approach as our detector generalizes effectively to diverse STEM domains despite being trained only on pure mathematical samples.

4.3 Ablation studies on training settings

To understand how different training settings influence the detection performance, we conduct ablation studies on three aspects of the training recipe: the ensemble strategy for aggregating hallucination annotations (Section 3.2), the training objective (Section 3.3), and the scalability of training data. Throughout these experiments, we use Qwen3-8B as the backbone and Math as the training dataset.

Adaptive ensemble consistently improves detection performance. We first compare the effectiveness of two ensemble strategies for generating training labels: a simple uniform ensemble that averages the annotations, and an adaptive ensemble that learns weights for each critic model. To obtain the ensemble weights, we sample a small held-out subset from the training samples to form the validation set $\mathcal{D}_{\text{val}}$ and minimize the loss in Eq. 2 to learn weights for individual models. We then use these weights to aggregate the annotations into a single token-level label sequence for each training example. We start by evaluating the quality of the ensembled labels under three combinations. Comb 1 uses all four critic models, Comb 2 uses QwQ-32B, GPT-4.1, and o4-mini, and Comb 3 uses GPT-4.1 and o4-mini. We evaluate the ensembled token-level labels against the ground-truth labels produced by the labeler models. As shown in Table 2, adaptive ensembling consistently improves label quality across all combinations, increasing the S_{incor} score by 17.95, 12.01, and 8.8 points for Comb 1, 2, and 3, respectively, while maintaining S_{cor}. We then evaluate the impact of these ensemble strategies on actual model training. For comparison, we also train a detector using labels

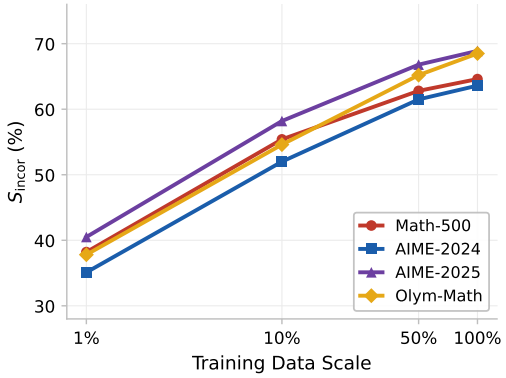


Figure 5: Detection performance across different training data scales on mathematical tasks.

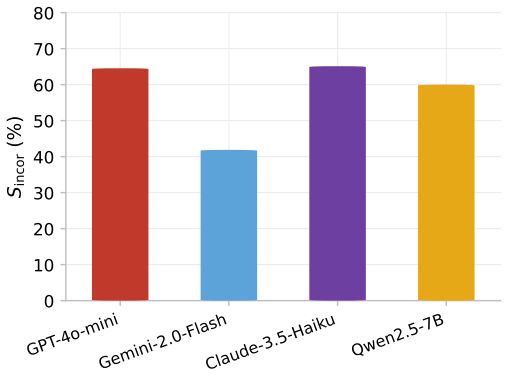


Figure 6: Detection performance across diverse policy models. The open-source policy model is Qwen2.5-7B-Instruct.

solely from the strongest critic model, o4-mini. In Figure 3, we report S_{incor} while controlling all S_{cor} at approximately 96 to ensure a fair comparison. The results indicate that detectors trained with adaptively ensembled data consistently perform better than the uniform counterpart. More importantly, we observe that detectors trained with data annotated by o4-mini alone underperform those using adaptive ensembles.

What if the detector is trained with different loss objectives? We next explore how the training strategy affects detector performance. We consider two training schemes described in Section 3.3: standard training and importance-weighted training (Eq. 3), which upweights tokens with high hallucination scores. We train detectors under both training schemes and evaluate them on four mathematical benchmarks. As shown in Figure 4, we report S_{incor} (S_{cor} is controlled around 96 to ensure a fair comparison) for two training strategies. Our results show that the importance-weighted strategy consistently improves detection performance across all benchmarks. These results suggest that our strategy is effective when hallucinated tokens are sparse, since upweighting hallucinated tokens helps mitigate the resulting class imbalance during training.

Impact of training data scaling on detection performance. To assess the impact of data scale, we train our detectors on subsets of the training set sampled at 1%, 10%, 50%, and 100% and evaluate them on the four mathematical benchmarks. As shown in Figure 5, increasing the data size consistently improves S_{incor} until reaching around 50% of the training dataset across all benchmarks. After this point, we observe that the improvement in detection performance slows down significantly, indicating that the gains from adding more training data start to saturate.

5 Generalization of hallucination detection

While our detector performs well on several mathematical and STEM tasks, real-world deployment still faces multiple challenges. In particular, the *policy model* that produces the responses may differ from the one used to generate our training data, and user queries may come from non-mathematical *task domains* where hallucination patterns may differ substantially. In the following sections, we evaluate our detector under these scenarios and explore simple strategies to enhance its generalization.

5.1 Shifts in policy models and task domains

Policy models. To evaluate the performance across different policy models, we generate evaluation samples with multiple policies on mathematical tasks. We consider two closed-source models, Gemini-2.0-Flash [32] and Claude-3.5-Haiku [33], and an open-source model, Qwen2.5-7B-Instruct. We report results on Math-500.

Task domains. In addition to diverse policy models, real-world queries may also come from a wide range of task domains. Here, we consider a practical task, code generation. We evaluate on Code-Elo [34] and LiveCodeBench-Lite [35] and generate evaluation samples using GPT-4o-mini

Table 3: Performance comparison between different strategies to improve the detector's generalization. We report S_{incor} / S_{cor}.

Strategy	GPT-4o-mini			Gemini-2.0-Flash		
	Math-500	Code-Elo	LiveCode-Lite	Math-500	Code-Elo	LiveCode-Lite
Baseline TOKENHD-8B	64.55 / 98.71	49.37 / 98.95	41.30 / 98.70	41.42 / 95.92	22.35 / 94.66	27.57 / 97.59
Mix Training	64.56 / 98.63	73.61 / 95.14	70.16 / 93.55	42.47 / 97.24	68.21 / 91.01	56.80 / 90.02
Average Merging	48.75 / 93.53	68.00 / 91.29	63.53 / 87.74	33.42 / 61.09	67.23 / 91.14	51.88 / 87.80
Task Vector	41.17 / 95.57	68.16 / 92.96	63.63 / 90.57	32.12 / 81.04	65.23 / 92.84	48.08 / 91.86
TIES-Merging	50.47 / 91.47	67.84 / 90.81	63.35 / 87.24	33.61 / 56.34	67.25 / 89.76	52.14 / 85.80
DARE-Merging	49.25 / 91.77	67.58 / 90.89	63.43 / 87.70	32.94 / 26.88	66.78 / 89.97	51.93 / 86.66

and Gemini-2.0-Flash as policy models. To obtain high-quality hallucination annotations for coding tasks, we employ GPT-5 and Claude-4.5-Sonnet [5] as our labeler models, and apply the uniform ensemble strategy for aggregating annotations.

Preliminary results on generalization. We begin with our "baseline" detector (baseline TOKENHD-8B) trained only with mathematical data generated by GPT-4o-mini (discussed in Section 4.1). For different policy models, Figure 6 (S_{cor} is controlled around 96 to ensure a fair comparison) shows that our detector demonstrates robust generalization across various policy models, including Claude-3.5-Haiku and Qwen2.5-7B-Instruct, even though our training data is solely sourced from GPT-4o-mini. Nevertheless, Gemini-2.0-Flash presents a more challenging distribution relative to our GPT-4o-mini training data, motivating the generalization strategies in Section 5.2. For task-domain generalization, Table 3 shows that our baseline TOKENHD-8B achieves 49.37 / 98.95 on Code-Elo and 41.30 / 98.70 on LiveCodeBench-Lite under the GPT-4o-mini policy. These results, obtained zero-shot with a math-only model on code tasks, highlight the need to expand training coverage to broader hallucination patterns. We investigate two practical strategies for this in the following section.

5.2 Improving detector's generalization capability

To augment the training set, we collect additional samples that vary both the policy model and the task domain. For policy-model diversity, we use Gemini-2.0-Flash to generate additional mathematical training samples on the same Math dataset. For task-domain diversity, we collect code generation samples using the OpenCodeReasoning dataset [36] using both GPT-4o-mini and Gemini-2.0-Flash as policy models. In all cases, we sample two responses per prompt following our previous settings. To handle the newly added data more efficiently, we consider two strategies with different computational costs and compare their effectiveness. **Mix Training Data**: We aggregate training data from different sources and train our detector on this combined dataset. This strategy serves as a strong baseline, though it requires retraining the detector whenever new training data is updated, which incurs a high computational cost. **Model Merging:** We train separate specialized detectors for different domains and then merge their weights into a single detector. This enables modular updates and aggregates expertise from multiple specialized models without large-scale retraining. Specifically, we adopt several common methods from previous studies [37], including: average merging [38], task vector [39], TIES-Merging [40] and DARE-Merging [41] (details can be accessed in Appendix H). As shown in Table 3, mix training on combined math and code data substantially improves code detection while maintaining math performance. Model merging can also improve generalization at relatively low computational cost. We also explore practical downstream applications of TOKENHD, including best-of-N candidate selection and targeted self-correction, demonstrating further utility of token-level error scores beyond detection (see Appendix I).

6 Related work

Hallucination detection and mitigation. Despite the exceptional capabilities of LLMs, they still suffer from hallucinations [7, 8, 42], where the generated text appears fluent but contains errors within the reasoning process. Many prior studies primarily investigate factual hallucinations, where LLM-generated contexts conflict with real-world knowledge or include unverifiable claims. To mitigate these issues, various strategies have been proposed, such as leveraging specialized decoding strategies [43] or RAG systems [12, 44] to prevent hallucination during inference, fine-tuning policy models to mitigate hallucinations [45, 46, 47], and post-hoc strategies to detect hallucination after

generation [10, 9, 48, 49, 11]. In contrast, we study hallucinations in reasoning-intensive tasks, where models generate contents that appear coherent but contain logical flaws, leading to incorrect final answers. Moreover, existing detection studies often rely on synthesized hallucinations, resulting in limited hallucination patterns, and may not reflect models' behaviors in practice. We avoid manually crafted hallucinations and instead build an automated data engine that samples reasoning traces from policy models and annotates token-level hallucinations using our scalable annotation framework, capturing more authentic hallucination behaviors beyond a fixed set of curated patterns.

Reward models for LLMs. Reward models (RMs) are critical in defining preference criteria and shaping the generation quality of LLMs. Conventional RMs typically assign a numerical score to an entire sequence to rank candidate responses [50, 51, 52, 53, 54], yet they provide sparse feedback and struggle to identify where errors occur within the reasoning. To address this, the research community proposed Process Reward Models (PRMs) to supervise intermediate reasoning steps [14, 15, 55, 16, 56]. However, existing PRMs depend on heuristic step boundaries that are hard to define for free-form text or require intensive MCTS [57] sampling, incurring high computational overhead. In contrast, TOKENHD provides dense reward supervision on free-form content without requiring step segmentation or costly MCTS sampling.

7 Conclusion and limitations

In this paper, we present TOKENHD, a holistic framework for developing fine-grained hallucination detectors in reasoning-intensive tasks. TOKENHD establishes a complete pipeline, integrating a scalable data engine for synthesizing high-quality annotations with an importance-weighted training strategy to mitigate label sparsity. Unlike conventional PRMs, our detector operates directly on free-form text at the token level, enabling precise hallucination localization without requiring predefined step separation. To validate the effectiveness of TOKENHD, we establish a rigorous evaluation framework and conduct extensive experiments on mathematical and STEM benchmarks. Experimental results demonstrate that our lightweight detector can surpass much larger reasoning models while maintaining a significantly lower inference overhead. In sum, TOKENHD offers a highly scalable and cost-efficient solution for fine-grained hallucination detection, providing a practical tool for enhancing the reliability of LLMs in complex reasoning.

References

[1] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 1

[2] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. 1, 4.1

[3] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. 1

[4] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. 1

[5] Anthropic. Claude sonnet 4.5, 2025. 1, 5.1

[6] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38, 2023. 1

[7] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025. 1, 6

[8] Adam Tauman Kalai, Ofir Nachum, Santosh S Vempala, and Edwin Zhang. Why language models hallucinate. *arXiv preprint arXiv:2509.04664*, 2025. 1, 6

[9] Yuyan Chen, Qiang Fu, Yichen Yuan, Zhihao Wen, Ge Fan, Dayiheng Liu, Dongmei Zhang, Zhixu Li, and Yanghua Xiao. Hallucination detection: Robustly discerning reliable answers in large language models. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 245–255, 2023. 1, 6

[10] Potsawee Manakul, Adian Liusie, and Mark Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 9004–9017, 2023. 1, 6

[11] Oscar Obeso, Andy Arditi, Javier Ferrando, Joshua Freeman, Cameron Holmes, and Neel Nanda. Real-time detection of hallucinated entities in long-form generation. *arXiv preprint arXiv:2509.03531*, 2025. 1, 6

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023. 1, 6

[13] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025. 1

[14] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023. 1, 6

[15] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024. 1, 6

[16] Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025. 1, 6

[17] Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*, 2024. 1

[18] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 4.1

[19] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. 4.1

[20] Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning. *arXiv preprint arXiv:2505.16400*, 2025. 4.1

[21] Alon Albalak, Duy Phung, Nathan Lile, Rafael Rafailov, Kanishk Gandhi, Louis Castricato, Anikait Singh, Chase Blagden, Violet Xiang, Dakota Mahan, et al. Big-math: A large-scale, high-quality math dataset for reinforcement learning in language models. *arXiv preprint arXiv:2502.17387*, 2025. 4.1

[22] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. 4.1

[23] Qwen Team. Qwq-32b: Embracing the power of reinforcement learning, 2025. 4.1

[24] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 4.1

[25] OpenAI Team. Introducing openai o3 and o4-mini, 2025. 4.1

[26] AIME-2024. Aime-2024, 2024. 4.1

[27] AIME-2025. Aime-2025, 2025. 4.1

[28] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, 2024. 4.1

[29] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. 4.1

[30] Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R Routledge, et al. Finqa: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, 2021. 4.1

[31] OpenAI Team. Introducing gpt-5, 2025. 4.1

[32] Google. Introducing gemini 2.0: our new ai model for the agentic era, 2024. 5.1

[33] Anthropic. Claude 3.5 sonnet, 2024. 5.1

[34] Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, et al. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *arXiv preprint arXiv:2501.01257*, 2025. 5.1

[35] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. 5.1

[36] Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding. *arXiv preprint arXiv:2504.01943*, 2025. 5.2

[37] Han Wu, Yuxuan Yao, Shuqi Liu, Zehua Liu, Xiaojin Fu, Xiongwei Han, Xing Li, Hui-Ling Zhen, Tao Zhong, and Mingxuan Yuan. Unlocking efficient long-to-short llm reasoning with model merging. *arXiv preprint arXiv:2503.20641*, 2025. 5.2

[38] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022. 5.2

[39] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022. 5.2

[40] Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36:7093–7115, 2023. 5.2

[41] Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024. 5.2

[42] Zhitao He, Sandeep Polisetty, Zhiyuan Fan, Yuchen Huang, Shujin Wu, and Yi R Fung. Mm-boundary: Advancing mllm knowledge boundary awareness through reasoning step confidence calibration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16427–16444, 2025. 6

[43] Xintong Wang, Jingheng Pan, Liang Ding, and Chris Biemann. Mitigating hallucinations in large vision-language models with instruction contrastive decoding. *arXiv preprint arXiv:2403.18715*, 2024. 6

[44] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020. 6

[45] Hanning Zhang, Shizhe Diao, Yong Lin, Yi Fung, Qing Lian, Xingyao Wang, Yangyi Chen, Heng Ji, and Tong Zhang. R-tuning: Instructing large language models to say 'i don't know'. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7106–7132, 2024. 6

[46] Yuji Zhang, Sha Li, Cheng Qian, Jiateng Liu, Pengfei Yu, Chi Han, Yi R Fung, Kathleen McKeown, Chengxiang Zhai, Manling Li, et al. The law of knowledge overshadowing: Towards understanding, predicting, and preventing llm hallucination. *arXiv preprint arXiv:2502.16143*, 2025. 6

[47] Weilei He, Feng Ju, Zhiyuan Fan, Rui Min, Minhao Cheng, and Yi R Fung. Empowering reliable visual-centric instruction following in mllms. *arXiv preprint arXiv:2601.03198*, 2026. 6

[48] Ning Miao, Yee Whye Teh, and Tom Rainforth. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. *arXiv preprint arXiv:2308.00436*, 2023. 6

[49] Gaurang Sriramanan, Siddhant Bharti, Vinu Sankar Sadasivan, Shoumik Saha, Priyatham Kattakinda, and Soheil Feizi. Llm-check: Investigating detection of hallucinations in large language models. *Advances in Neural Information Processing Systems*, 37:34188–34216, 2024. 6

[50] Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, Karthik Ganesan, Wei-Lin Chiang, Jian Zhang, and Jiantao Jiao. Starling-7b: Improving helpfulness and harmlessness with rlaif. In *First Conference on Language Modeling*, 2024. 6

[51] Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-reward: Bag of tricks for reward modeling in llms. *arXiv preprint arXiv:2410.18451*, 2024. 6

[52] Jialun Zhong, Wei Shen, Yanzeng Li, Songyang Gao, Hua Lu, Yicheng Chen, Yang Zhang, Wei Zhou, Jinjie Gu, and Lei Zou. A comprehensive survey of reward models: Taxonomy, applications, challenges, and future. *arXiv preprint arXiv:2504.12328*, 2025. 6

[53] Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. Rm-bench: Benchmarking reward models of language models with subtlety and style. *arXiv preprint arXiv:2410.16184*, 2024. 6

[54] Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. Pairjudge rm: Perform best-of-n sampling with knockout tournament. *arXiv preprint arXiv:2501.13007*, 2025. 6

[55] Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. Prmbench: A fine-grained and challenging benchmark for process-level reward models. *arXiv preprint arXiv:2501.03124*, 2025. 6

[56] Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1009–1024, 2025. 6

[57] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012. 6

[58] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. C

[59] An Yang, Beichen Zhang, Binyuan Chen, et al. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024. F

A Broader impacts

This paper introduces TOKENHD, a framework designed for token-level hallucination detection in LLM-generated responses. Our trained detector is highly efficient and effective in pinpointing subtle hallucinations within reasoning-intensive tasks. Since our detector is lightweight and inference-efficient, it can be seamlessly integrated into most existing LLM systems to detect the potential hallucinations of generated text without significant computational overhead. In sum, our detector contributes to enhancing the truthfulness of LLM-generated content, which is a crucial step toward the reliable deployment of LLMs in real-world applications.

B A closer look at how we obtain the token-level hallucination annotations

B.1 Prompts for data annotation

This section describes the prompt designs used in our data annotation process. To identify hallucinated text within generated responses, we prompt LLMs to achieve this. Specifically, we leverage advanced and highly capable models as labeler models to obtain ground-truth annotations for evaluation. We use relatively cheaper or open-source models as critic models to generate large-scale training data due to cost considerations. We apply the same annotation prompts for both labeler and critic models. Our experiments primarily involve three categories of data sources, including the mathematical tasks: Math training data, AceReason-Math, Big-Math, Math-500, AIME-2024, AIME-2025 and Olym-Math, STEM tasks: Olym-Phy, GPQA and FinQA and code generation tasks: OpenCodeReasoning, Code-ELO and LiveCodeBench-Lite. For both mathematical and STEM tasks, we use the prompt shown in Figure 8. For code generation tasks, we use the prompt in Figure 9.

B.2 Details of text restoration

After obtaining the textual annotations, we map these text fragments back to the token space. This process requires each identified fragment to match its corresponding part in the original response exactly. However, we found that the raw text from the LLM often differs slightly from the original text, typically in formula formatting or paragraph breaks. To address this, we prompt o4-mini to restore the raw text (the prompt is shown in Figure 10) so that each fragment can be correctly aligned with the original response. Since a single restoration is often insufficient, we developed an iterative restoration strategy, detailed in Algorithm 1. This method effectively restores fragments to their original text with high efficiency. In our experiments, we set the max iteration rounds to 3. Table 4 shows the direct-match rate (fraction of raw critic spans that already match the original text verbatim) and the post-restoration rate (fraction successfully aligned after iterative restoration). On average across all four critics, the restoration process recovers 98.10% of spans, a large improvement over the 64.25% direct-match baseline.

Table 4: Text restoration rates across four critic models. Our algorithm largely improves the match rate of text spans after restoration.

Critic	Direct Match (%)	Post-Restoration (%)
DeepSeek-R1-Qwen3-8B	26.57	96.43
QwQ-32B	44.18	97.46
GPT-4.1	95.66	99.45
o4-mini	83.19	98.74
Average	**64.25**	**98.10**

B.3 How to filter high-quality data

Before training our detector, we apply a filtering strategy to improve the quality of the dataset. First, we remove samples with incomplete or corrupted generation traces. For the remaining data, we filter out samples that have incorrect final results but contain no identified hallucinations. We also remove samples with low annotation consistency. Specifically, a sample is removed from our dataset if the maximum aggregated hallucination score is below 0.5.

C Details of training hallucination detector

For generating mathematical training samples, we leverage the whole Math training dataset and subsets of AceReason-Math and Big-Math, obtaining around 49,000 valid samples after data filtering. To balance the dataset, we supplement it with a small set of non-hallucinated samples (those with correct answers and no identified errors). We train for one epoch using the Adam optimizer [58] with a cosine decay schedule and linear warmup, setting the peak learning rate to 1×10^{-5}.

D Ground-truth label quality verification

D.1 Annotation rubric

Table 5 lists the rubric provided to human annotators for identifying and marking hallucinated spans in model-generated solutions. The criteria are applied in order; later criteria address ambiguous edge cases that arise in complex multi-step reasoning.

Table 5: Rubric for hallucination span annotation. Human annotators apply these criteria in order to identify and mark erroneous spans in model-generated solutions.

#	Criterion	Guideline
R1	Error definition	Mark a span as hallucinated if it contains: (a) an incorrect mathematical operation or numerical result; (b) a logically invalid inference (e.g., incorrectly applying a theorem, reversing an implication, or making an unsupported leap); (c) a factual claim not supported by the problem statement or standard mathematical knowledge; or (d) an incorrect intermediate conclusion that directly propagates error to subsequent reasoning steps or the final answer.
R2	Span granularity	Mark the *minimal* contiguous span that contains the error; do not extend the marked span to include surrounding correct text. If the error involves a single symbol, value, or expression, mark the smallest clause or equation that makes the error meaningful and interpretable in context.
R3	Multiple errors	When multiple distinct errors exist, mark each independently using separate, non-overlapping spans. If error A causes a downstream error B, mark both independently. Do not merge errors separated by correct intervening text, even if they are logically related.
R4	Correct solutions	If the solution is entirely correct and reaches the correct final answer, output "No errors!". Do not mark minor stylistic differences, equivalent reformulations of correct expressions, or unnecessarily verbose but correct steps.
R5	Ambiguous steps	Do not mark a step that is redundant but mathematically valid. When a step follows from an earlier erroneous assumption, prioritize marking the original erroneous assumption as the primary error span; mark the downstream step only if it introduces an *additional* independent error beyond what was already caused by the upstream mistake.

D.2 Annotation quality assessment

To verify that the rubric produces reliable annotations in practice, we recruited human annotators to rate a held-out set of annotated samples on a 1–5 scale along two axes: **Accuracy** (whether the marked spans actually contain errors) and **Completeness** (whether the major errors in the solution are covered). Annotators followed the rubric above and were assisted by an advanced LLM (Gemini-3.1-Pro) to help verify their judgments and improve rating consistency; final scores reflect human decisions.

As shown in Figure 7, overall scores are high across all task domains (math + STEM: 4.63/5, code: 4.55/5). Accuracy is particularly strong for math and STEM tasks (4.69), indicating that annotated

spans are rarely false positives. Completeness (4.57) is modestly lower, consistent with the difficulty of exhaustively identifying all errors in complex multi-step solutions. For code tasks, the pattern inverts (accuracy 4.40, completeness 4.69), which we attribute to token-boundary ambiguity in code blocks rather than incorrect error identification. Taken together, these scores confirm that our rubric-guided annotation pipeline produces high-quality token-level supervision.

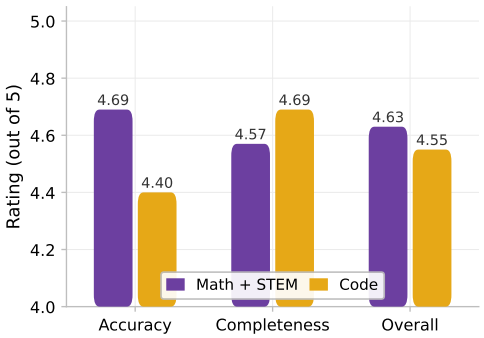


Figure 7: Human annotation quality assessment (1–5 scale). Annotators rated GT annotations on accuracy and completeness across two task domains, assisted by an advanced LLM for verification.

E Extended discussions of our TOKENHD framework

E.1 Relationship between our detector and reward models

Reward models are typically designed to provide training signals that reflect specific preferences, allowing subsequent training procedures such as reinforcement learning to select better responses and improve the policy model's performance. While our work shares a similar form with reward models, we differ in our objectives. Our detector is specifically designed to identify hallucinations at the token level, serving as an indicator of where hallucinations occur. Despite this difference in usage, our paradigm could potentially provide dense, token-level rewards, offering more fine-grained supervision for training and ranking responses than existing response-level or step-level signals.

E.2 Ablation of training settings

Our primary experiments utilize the Qwen3 series as backbone models, with a focus on small-scale architectures. We make these choices for two key considerations. First, we observe that training with a larger backbone, such as Qwen3-14B, yields an improvement of approximately 2 points in S_{incor} over the 8B version on the Math-500 dataset. As detailed in Section 4.1 and Table 1, performance scales consistently with model size across this range. Even the smallest 0.6B variant achieves 63.64 on Math-500, outperforming QwQ-32B (55.05) by a substantial margin. We also use other model families as backbones, such as Qwen2.5 and Llama3, and find that the Qwen3 series consistently performs better. Second, we prioritize efficiency to ensure the detector is lightweight for practical deployment. Since our detector is primarily designed to serve as a plug-in hallucination detection module following much larger LLM systems, using a small detector is critical to maintain high inference efficiency and minimize additional computational overhead.

E.3 Advantages compared to larger reasoning models

In our evaluation, we observe that our detector significantly outperforms its original backbone models in hallucination detection, even though those backbones provide much more reasoning content. We also find that our detector can outperform larger, more advanced critic models, despite being trained on data labeled by them. While we acknowledge that o4-mini still leads on Math-500, our approach offers two distinct advantages. First, our model is much more lightweight than o4-mini, leading to faster inference speeds and significantly lower API costs. Second, our detector does not rely on text restoration, which can directly identify hallucinated text fragments within the original response. In contrast, o4-mini requires an extra text restoration step to extract exact fragments, which adds

computational cost and complexity to the pipeline. With only a modest gap in detection performance, our model provides a much simpler and more efficient solution for hallucination detection in practice.

E.4 Different roles between critic and labeler models

Our methodology relies on two sets of models for hallucination annotation: critic models for generating training data and a labeler model for producing ground-truth labels for evaluation. Although these models follow a similar annotation workflow and use identical prompts, they serve distinct purposes. First, there is a trade-off regarding annotation costs. Advanced models like GPT-5 are extremely expensive, making them less practical for generating large-scale training data. Therefore, we utilize a set of more cost-effective critic models for training while reserving the most powerful labeler for annotating evaluation samples. Second, employing distinct model sets for annotating training and evaluation data prevents direct model distillation. This ensures a more rigorous and fair assessment. Notably, although our detector is trained on data annotated by the critic models, it outperforms most of these critic models themselves. This demonstrates that our approach goes beyond simple imitation. Instead, it provides an efficient and robust training framework that effectively identifies hallucinations at the token level.

F Comparison with process reward models

Process Reward Models (PRMs) assign correctness scores to each reasoning step, making them the closest existing paradigm to our token-level hallucination detection task. We compare TOKENHD against two strong open-source PRM baselines, Qwen2.5-Math-PRM-7B and Qwen2.5-Math-PRM-72B [59], designed for mathematical reasoning. On Math-500, we follow the official PRM implementation to divide and score each response. We convert each step's correctness score q_j to a hallucination score $1 - q_j$ and assign it to every token within that step. As shown in Table 6, TOKENHD-8B achieves relative improvements of 43.70% and 17.90% in S_{incor} over Qwen2.5-Math-PRM-7B and Qwen2.5-Math-PRM-72B on Math-500, respectively.

Table 6: Token-level S_{incor} comparison between PRMs and TOKENHD-8B.

Model	Math-500	Olym-Math	GPQA	Olym-Phy	FinQA
Qwen2.5-Math-PRM-7B	44.92	18.41	5.17	8.29	8.35
Qwen2.5-Math-PRM-72B	54.75	21.62	8.14	13.77	13.69
TOKENHD-8B	**64.55**	**68.45**	**65.49**	**65.85**	**64.35**

Inference latency. Table 7 reports median inference latency on NVIDIA H20 GPUs. TOKENHD-8B has similar latency to PRM-7B, with higher S_{incor} on Math-500. PRM-72B takes more than 8 times as long as TOKENHD-8B, while TOKENHD-8B outperforms PRM-72B in S_{incor} on Math-500.

Table 7: Median inference latency (ms) at different input lengths on NVIDIA H20 GPUs.

Input tokens	TOKENHD-8B	PRM-7B	PRM-72B
1,024	130.0	116.8	1,128.2
2,048	263.8	234.0	2,260.0
4,096	550.8	479.4	4,649.8
8,192	1,188.1	1,010.1	9,707.8

G AUROC and AUPRC

Beyond the primary S_{incor} and S_{cor} metrics, we report AUROC and AUPRC to provide a threshold-independent view of detection performance. Both metrics are computed at the token level on incorrect samples only. For each benchmark, we pool all tokens from all incorrect samples and form a set of (predicted score, ground-truth label) pairs, where the predicted score is the continuous output $\widehat{s}_i \in [0, 1]$ from the detector and the ground-truth label is the binarized annotation $\mathbb{I}[s_i > \beta_I]$. AUROC

is then computed as the area under the ROC curve over this pooled set, measuring the probability that a randomly chosen hallucinated token receives a higher predicted score than a randomly chosen non-hallucinated token. AUPRC is the area under the precision-recall curve over the same pooled set, and is more informative in our setting because hallucinated tokens are sparse within a response: the precision-recall curve directly captures the trade-off between localization precision and recall at every operating point, and is more sensitive to the detection of rare positive tokens than AUROC.

For critic models, the predicted score for each token is its binary annotation: a token inside a flagged error span receives a score of 1, and all other tokens receive a score of 0. This binary scoring is used directly to compute AUROC and AUPRC for the critics. The asymmetry in score type (critics output binary values while TOKENHD outputs continuous scores) reflects a genuine capability difference rather than an unfair evaluation design. Table 8 reports results for TOKENHD-8B and four critic models on four mathematical benchmarks. TOKENHD-8B achieves the best AUROC on all four benchmarks, ranging from 0.8739 to 0.9004, surpassing the strongest critic per benchmark by 0.04–0.09.

This advantage stems from a fundamental difference in the nature of the two approaches' outputs. Critic models produce hard binary annotations: a token is either inside a flagged span or not, leaving no room for confidence or gradation. When a critic misses an error entirely, the predicted score for every token in that sample collapses to zero. TOKENHD, by contrast, outputs continuous scores for every token, assigning high values to likely hallucinated positions and near-zero values elsewhere. This graded output gives a richer signal for ranking tokens, which is directly reflected in the higher AUROC and AUPRC values.

Table 8: AUROC and AUPRC at the token level on four mathematical benchmarks.

Metric	Model	Math-500	Olym-Math	AIME-2024	AIME-2025
AUROC	TOKENHD-8B	**0.8877**	**0.9004**	**0.8739**	**0.8947**
	o4-mini	0.8489	0.8288	0.7880	0.8226
	GPT-4.1	0.7042	0.7591	0.7206	0.7543
	QwQ-32B	0.7358	0.7283	0.6904	0.7082
	R1-Qwen3-8B	0.6850	0.6703	0.6484	0.6435
AUPRC	TOKENHD-8B	**0.7537**	**0.7760**	**0.7359**	**0.8014**
	o4-mini	0.6910	0.6423	0.5908	0.6088
	GPT-4.1	0.4271	0.4878	0.4426	0.4987
	QwQ-32B	0.5545	0.5292	0.4919	0.5212
	R1-Qwen3-8B	0.5031	0.4696	0.4562	0.4663

H Details of model merging

Let θ_{math} and θ_{code} denote the parameters of the math-only and code-only detectors, and θ_{base} the shared Qwen3-8B backbone. We define the task vectors as $\tau_{\text{math}} = \theta_{\text{math}} - \theta_{\text{base}}$ and $\tau_{\text{code}} = \theta_{\text{code}} - \theta_{\text{base}}$. We also denote the classification head (which maps the final hidden state to token-level hallucination probabilities) as θ_{head}, which is the most calibration-sensitive component of the model.

- **Average Merging**: $\theta_{\text{merged}} = \frac{1}{2}(\theta_{\text{math}} + \theta_{\text{code}})$.
- **Task Vector**: $\theta_{\text{merged}} = \theta_{\text{base}} + \alpha(\tau_{\text{math}} + \tau_{\text{code}})$, $\alpha = 1.0$.
- **TIES-Merging**: For each parameter dimension, we prune entries of τ_{math} and τ_{code} with absolute value below the top-20% threshold (i.e., retaining the 20% largest-magnitude entries), resolve sign conflicts between the two pruned vectors by majority vote, and add the merged result back to θ_{base}, $\alpha = 1.0$.
- **DARE-Merging**: Randomly drop 80% of entries in τ_{code}, rescale survivors by $\frac{1}{1-0.8}$ to preserve expectation, and merge the result with θ_{math} using task arithmetic: $\theta_{\text{merged}} = \theta_{\text{math}} + \alpha\,\tilde{\tau}_{\text{code}}$, $\alpha = 1.0$.

Algorithm 1 Iterative Text Restoration

Input: Hallucinated text fragments $(r_m)_{m=1}^{M} = [r_1, \ldots, r_M]$; Restoration Model $\mathcal{R}(\cdot)$; Max iteration rounds N; Original text response $\mathbf{y}$.
Output: List of successfully restored fragments $\mathbf{r}_{\text{res}}$.
Initialization: $\mathbf{r}_{\text{res}} \leftarrow []$, $\mathbf{r}_{\text{unres}}^{0} \leftarrow (r_m)_{m=1}^{M}$.
for $n = 1$ **to** N **do**
  Step 1: Restore text candidates
  $\widetilde{\mathbf{r}}^{n} \leftarrow \mathcal{R}(\mathbf{r}_{\text{unres}}^{n-1}, \mathbf{y})$
  Step 2: Verify the success of restoration
  $\mathbf{r}_{\text{unres}}^{n} \leftarrow []$
  for each fragment $\widetilde{r} \in \widetilde{\mathbf{r}}^{n}$ **do**
    if $\widetilde{r}$ is contained within $\mathbf{y}$ **then**
      Append $\widetilde{r}$ to $\mathbf{r}_{\text{res}}$
    else
      Append $\widetilde{r}$ to $\mathbf{r}_{\text{unres}}^{n}$
    end if
  end for
  Step 3: Early stopping when all text fragments are restored
  if $\mathbf{r}_{\text{unres}}^{n}$ is empty **then**
    break
  end if
end for
return $\mathbf{r}_{\text{res}}$

I Applications: best-of-N selection and self-correction

We demonstrate two practical downstream applications of TOKENHD: using token-level error scores as a scoring function for best-of-N candidate selection, and providing fine-grained error hints for self-correction. Both experiments are conducted on Math-500 with GPT-4o-mini as the policy model, focusing on samples where the initial response is incorrect.

I.1 Best-of-N selection

A natural application of a hallucination detector is to score multiple candidate solutions and select the most reliable one. For each incorrect sample, we generate ten candidate solutions and use TOKENHD-8B to assign a per-token error probability to each candidate, aggregate these probabilities into a single candidate-level score, and select the candidate with the lowest aggregated score (i.e., the candidate with the least predicted hallucination). We compare three aggregation strategies: (1) **Full-Response Mean**: average token error probability across the entire response; (2) **Full-Response Min**: minimum token error probability across the response; (3) **Worst-10% Mean**: average error probability of the 10% of tokens with the highest scores, capturing the worst predicted region. We measure selection accuracy relative to the oracle ceiling of 56.4%, the fraction of samples where at least one of the ten candidates is correct.

As shown in Table 9, using the Full-Response Mean improves accuracy from the majority-vote baseline of 25.8% to 31.8%. By restricting aggregation to the minimum token score, Full-Response Min achieves the best result at 35.2%, reaching 62.4% of the oracle ceiling. This finding indicates that a response's most reliable token (the one with the lowest predicted error probability) provides a particularly discriminative signal for ranking candidates. Worst-10% Mean (34.3%) also substantially outperforms full-sequence averaging, further corroborating this observation. These results demonstrate that TOKENHD can serve as an effective scoring function for best-of-N selection without any additional training or reward modeling.

I.2 Self-correction with token-level hints

We investigate whether token-level error localization from TOKENHD can assist a language model in correcting its own mistakes. For each incorrect sample, a correction model (we use GPT-4o-mini to perform the self-correction since it is the primary policy model used in our experiments) is provided

Prompt for Identifying Hallucinations in Math and STEM Tasks

```
You are given a question and its corresponding solution. Your task is to analyze whether the
solution completely and correctly solves the given problem.

**Your Tasks**:
- Carefully examine the solution step by step
- Identify if there are any errors in the reasoning, logic, facts, calculations, or final result
- If errors are found, extract the exact erroneous parts from the solution and list them using
error tags

**Instructions**:
- **You MUST identify all distinct errors in the solution. Do not stop after finding the first
error.**
- Analyze the solution thoroughly for factual errors, mathematical errors, logical flaws,
incorrect reasoning or wrong final answers.
- **If the solution contains any errors, you MUST extract and tag every erroneous text segment you
find.**
- **IMPORTANT**: Use the original text from the solution without any modifications - maintain the
exact same format, wording, punctuation, and mathematical notation as in the original.
- Keep the extracted error text **complete** and **relevant** to the specific error. Do **NOT**
extract only mathematical symbols or formulas.
- Tag each error using `<error n></error n>` format, starting from index 1 and numbering
sequentially.
- If the solution is completely correct, simply state "No errors!".

**Input Format**:
Problem: [PROBLEM_STATEMENT]
Solution: [SOLUTION_TEXT]

**Output Format**:
If errors are found:

<error 1>[EXACT_TEXT_FROM_SOLUTION_1]</error 1>
<error 2>[EXACT_TEXT_FROM_SOLUTION_2]</error 2>
<error 3>[EXACT_TEXT_FROM_SOLUTION_3]</error 3>
...

If no errors are found, simply output:
No errors!

Please analyze the following:

Problem: {problem}

Solution: {solution}
```

Figure 8: Prompts used for identifying hallucinations in mathematical and STEM tasks.

Table 9: Best-of-N selection results on incorrect Math-500 samples using TOKENHD-8B as the scorer. Oracle ceiling = 56.4% (fraction where at least one of ten candidates is correct). Majority vote is the baseline.

Strategy	**Accuracy**	**% of Oracle**
Majority vote (baseline)	25.8%	45.7%
Full-Response Mean	31.8%	56.4%
Worst-10% Mean	34.3%	60.8%
Full-Response Min	**35.2%**	**62.4%**
Oracle	56.4%	100%

with the original problem, the previous incorrect solution, and a condition-specific hint. We run up to three correction iterations per sample: if the correction model produces the correct answer on iteration k, the sample is counted as successfully corrected at iteration k; otherwise, the corrected solution from iteration k is fed back as the new "previous solution" for iteration $k+1$. We compare four conditions that differ in the type of hint provided.

Baseline: the correction model is informed that its previous answer is incorrect and asked to retry, with no localization information provided. **TokenHD**: suspected error regions are highlighted directly

Prompt for Identifying Hallucinations in Code Generation Tasks

```
You are given a coding problem and its corresponding solution. Your task is to analyze whether the
solution completely and correctly solves the given problem.

**Your Task**:
- Carefully examine the solution step by step, including the code implementation and any
accompanying explanation.
- Identify if there are any errors in the syntax, logic, algorithm, implementation, or final
result.
- If errors are found, extract the exact erroneous parts from the solution and list them using
error tags.

**Instructions**:
- **You MUST identify all distinct errors in the solution. Do not stop after finding the first
error.**
- Treat as an error any of the following: syntax or runtime errors, logical flaws or incorrect
algorithms, wrong or incomplete handling of edge cases (e.g., empty inputs, boundary values
defined by the problem), violations of the problem's constraints (e.g., required input/output
format, time/space limits), obviously incorrect or incomplete implementation of the required
logic, misuse or misunderstanding of encodings, data types, or return values.
- Verify that all built-in functions and libraries are used correctly and that their behavior
matches the problem requirements.
- Analyze both the code and any explanation text; contradictions with the problem or with the code
are also errors.
- **If the solution contains any errors, you MUST extract and tag every erroneous text segment you
find.**
- **Use only original text from the solution**: do not modify wording, formatting, punctuation, or
indentation; do not invent any text.
- Keep each extracted error segment complete and relevant to that specific error. Do not extract
only single symbols or isolated keywords unless they alone form the entire error.
- If one logical error spans multiple lines, you may include the entire block in a single error
tag.
- Tag each error using `<error n></error n>` format, starting from 1 and numbering sequentially.
- If the solution is completely correct, output exactly: "No errors!".

**Input Format**:
Problem: [PROBLEM_STATEMENT]
Solution: [SOLUTION_TEXT]

**Output Format**:
If errors are found:

<error 1>[EXACT_TEXT_FROM_SOLUTION_1]</error 1>
<error 2>[EXACT_TEXT_FROM_SOLUTION_2]</error 2>
<error 3>[EXACT_TEXT_FROM_SOLUTION_3]</error 3>
...

If no errors are found, simply output:
No errors!

Please analyze the following:

Problem: {problem}

Solution: {solution}
```

Figure 9: Prompts used for identifying hallucinations in code generation tasks.

in the solution text using inline markers («<. . .»>), generated by TOKENHD-8B (prompt design shown in Figure 11). The hints direct the correction model's attention to the most suspect token spans. **Step**: errors are marked at the paragraph level by aggregating token-level scores to the nearest paragraph boundary, simulating the coarser granularity of a PRM-style hint. **Oracle**: ground-truth error spans are directly provided as hints, representing the theoretical performance upper bound.

As shown in Table 10, the baseline correction rate is 16.9%, reflecting the difficulty of self-correction without any localization cues. Both the TokenHD condition (19.9%) and the step condition (19.1%) significantly outperform the baseline, demonstrating that even approximate error localization enables more targeted revision. Crucially, TOKENHD reaches 98% of the oracle upper bound (19.9% vs. 20.3%), confirming that fine-grained token-level localization translates directly into effective correction guidance. The higher first-iteration success rate for TokenHD (11.4%) compared to the step

Prompt for Text Restoration

```
You are given one original text and multiple extracted text segments that were supposedly
extracted from the original text, but may contain slight variations, errors, or modifications.
1. **Original Text**: A long document or passage
2. **Extracted Text**: A segment that was supposedly extracted from the original text, but may
contain slight variations, errors, or modifications

**Your Task**:
For each extracted text segment, locate the corresponding section in the original text that best
matches it, and output the exact original text segment with appropriate tags.
**Instructions**:
- The extracted text may have minor differences from the original, such as: spelling variations or
typos, punctuation differences, minor word substitutions, and formatting changes.
- Find the most similar section in the original text for each extracted segment. If multiple
potential matches exist for one segment, choose the one with the highest similarity.
- Only return the portion of the original text that corresponds to the extracted segment. Do NOT
add extra content before or after the matching section, and do NOT remove any content that should
be included.
- **IMPORTANT**: Return the **EXACT** text from the original document that corresponds to each
extracted segment. Pay attention to differences, such as *whitespace*, *punctuation*, *brackets*,
and *line breaks*, etc.
- If no reasonable match is found, output "NO_MATCH_FOUND".

**Input Format**:
Original Text: [ORIGINAL_TEXT]
<extract1>[EXTRACTED_TEXT_1]</extract1>
<extract2>[EXTRACTED_TEXT_2]</extract2>
<extract3>[EXTRACTED_TEXT_3]</extract3>
...

**Output Format**:
<result1>[EXACT_SEGMENT_FROM_ORIGINAL_1]</result1>
<result2>[EXACT_SEGMENT_FROM_ORIGINAL_2]</result2>
<result3>[EXACT_SEGMENT_FROM_ORIGINAL_3]</result3>
...

Please process the following input:
Original Text: {original_text}
Extracted Text: {extracted_text}
```

Figure 10: Prompts used for restoring the identified hallucinated text to match the original response.

condition (8.9%) further shows that finer-grained localization enables the correction model to identify and fix errors more immediately, reducing the number of revision attempts required. Together, these results confirm that TOKENHD provides a practical and efficient mechanism for targeted self-correction, approaching oracle-level performance while requiring no ground-truth information.

Table 10: Self-correction results on incorrect Math-500 samples using GPT-4o-mini as the correction model, run for up to three iterations. Correction Rate: fraction of samples successfully corrected. 1st-Iter Rate: fraction corrected on the first iteration. % of Oracle: correction rate as a fraction of the oracle upper bound (20.3%).

Condition	Correction Rate	1st-Iter Rate	% of Oracle
Baseline	16.9%	8.5%	83.3%
Step	19.1%	8.9%	94.1%
TokenHD	19.9%	11.4%	**98.0%**
Oracle	20.3%	14.4%	100%

J Robustness of the evaluation protocol

Our evaluation relies on two design choices that could each affect the validity of reported results: the ensemble weights that aggregate critic annotations into training labels, and the binarization thresholds used for evaluation. We conduct two validation experiments to confirm that these choices are stable.

Prompt for Self-Correction with Token-Level Hints

```
You are given a math problem and a previous solution attempt that was incorrect. Suspected error
regions in the previous solution are marked with triple angle brackets: «<error region»>.
Your task is to carefully review the previous solution, paying special attention to the marked
regions, identify the mistakes, and produce a corrected solution.

**Instructions**:
- Re-examine the flagged regions carefully; they are likely where the error occurred.
- Provide a complete corrected solution, not just the corrected portions.
- Ensure your final answer is placed inside \boxed{}.

**Input Format**:
Problem: [PROBLEM_STATEMENT]
Previous Solution (with suspected error regions marked): [PREVIOUS_SOLUTION_WITH_«<...»>_MARKERS]

**Output Format**:
Corrected Solution: [YOUR_CORRECTED_SOLUTION]

Please analyze and correct the following:

Problem: {problem}

Previous Solution: {previous_solution_with_hints}
```

Figure 11: Prompt used for self-correction with token-level hints. Suspected error regions identified by TOKENHD-8B are highlighted inline using «<...»> markers, directing the correction model's attention to the most suspect token spans.

J.1 Ensemble weight stability across data subsets

Our adaptive ensemble assigns learned weights to each critic model by minimizing Eq. 2 on a held-out validation set $\mathcal{D}_{\text{val}}$. To verify that these weights are stable and do not depend on the specific data used for optimization, we repeat the weight optimization on two independent held-out subsets from our training data: Math-Train (drawn from the Math dataset) and AceReason (drawn from AceReason-Math). Both subsets are mutually disjoint from each other and from the samples used for model training. We optimize the weights separately on each subset using the same procedure described in Section 3.2. To quantify variability across data samples within each subset, we apply bootstrap resampling: for each subset, we draw 1,000 bootstrap resamples by sampling the same number of individual (query, response, critic annotation) tuples with replacement as in the original subset, run the weight optimization on each resample, and compute 95% confidence intervals from the 2.5th and 97.5th percentiles of the resulting weight distributions.

The weight ordering is consistent across both subsets. o4-mini receives the highest weight ($\approx$0.60; 95% CI: [0.52, 0.65] on Math-Train, [0.53, 0.64] on AceReason), followed by GPT-4.1 ($\approx$0.30) and QwQ-32B ($\approx$0.09). The point estimates differ by less than 2% across the two subsets for all critics. These results show that the learned weights are not sensitive to the choice of optimization data, and that the assigned weights reflect genuine capability differences among the critics rather than subset-specific noise.

J.2 Threshold sensitivity

We use fixed binarization thresholds $\beta_I = 0.5$ for ground-truth labels and $\beta_{\widehat{I}} = 0.5$ for predicted scores throughout all experiments. To assess how sensitive the reported results are to this choice, we evaluate TOKENHD-8B on all five benchmarks under perturbed threshold settings. Specifically, we vary each threshold one at a time over three values $\{0.45, 0.50, 0.55\}$ while holding the other fixed at its default of 0.5. This yields four non-default configurations in total: $(\beta_I, \beta_{\widehat{I}}) \in \{(0.45, 0.5), (0.55, 0.5), (0.5, 0.45), (0.5, 0.55)\}$. For each configuration, we compute the absolute deviation in S_{incor} relative to the default setting (β_I=0.5, $\beta_{\widehat{I}}$=0.5), and report the maximum deviation across all four configurations for each benchmark in Table 11.

For the four mathematical and scientific reasoning benchmarks (Math-500, Olym-Math, GPQA, Olym-Phy), the maximum deviation is at most 1.66 percentage points. FinQA, the specialized

Table 11: Maximum absolute S_{incor} deviation (percentage points) from the default threshold pair (β_I=0.5, $\beta_{\hat{I}}$=0.5) when each threshold is independently varied by ± 0.05. Default S_{incor} and the observed range are shown for reference.

Benchmark	**Default (%)**	**Range (%)**	**Max $\lvert\Delta\rvert$ (pts)**
Math-500	60.73	[59.65, 61.26]	1.08
Olym-Math	65.42	[64.78, 65.42]	0.64
GPQA	63.47	[61.81, 64.63]	1.66
Olym-Phy	63.68	[63.51, 64.46]	0.78
FinQA	61.34	[57.97, 63.47]	3.37
Mean	–	–	**1.51**

financial QA benchmark, shows a larger maximum deviation of 3.37 percentage points; the mean across all five benchmarks is 1.51 percentage points. These results confirm that our evaluation is not sensitive to the exact threshold values, and that the reported performance differences remain stable under reasonable threshold perturbations.

K Detection performance across response lengths

We examine whether detection performance varies with response length. We partition the incorrect evaluation samples into three bins by absolute token count (<500, 500–1000, >1000) and report the average S_{incor} across all seven benchmarks for both TOKENHD-1.7B and TOKENHD-8B.



Figure 12: Average S_{incor} across seven benchmarks for TOKENHD-1.7B and TOKENHD-8B, grouped by response length. Samples are partitioned by absolute token count into three bins: <500, 500–1000, and >1000.

As shown in Figure 12, both models achieve their highest S_{incor} in the 500–1000 token bin (TOKENHD-1.7B: 62.6, TOKENHD-8B: 63.9). Both models show modest declines in the <500 token bin (TOKENHD-1.7B: 60.5, TOKENHD-8B: 61.0). In the >1000 token bin, TOKENHD-1.7B shows a similar decline (60.9), while TOKENHD-8B remains close to its peak (63.7). TOKENHD-8B consistently outperforms TOKENHD-1.7B across all bins, in line with the scaling trend observed in the main results.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Please see the experiments.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: In the conclusion.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: No theoretical proof.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Attach codes.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Attach codes and data examples.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Please see the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Please see the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Please see the experimental settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: This paper conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Please see the appendix.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: No such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have incorporated the appropriate citations.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Please see the supplemental results.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We employ LLMs as the primary annotators for hallucination detection.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.