

# Revisiting Judge Decoding from First Principles via Training-Free Distributional Divergence

Anonymous ACL submission

## Abstract

Judge Decoding accelerates LLM inference by relaxing the strict verification of Speculative Decoding, yet it typically relies on expensive and noisy supervision. In this work, we revisit this paradigm from first principles, revealing that the “criticality” scores learned via costly supervision are intrinsically encoded in the draft-target distributional divergence. We theoretically prove a structural correspondence between learned linear judges and Kullback-Leibler (KL) divergence, demonstrating they rely on the same underlying logit primitives. Guided by this, we propose a simple, training-free verification mechanism based on KL divergence. Extensive experiments across reasoning and coding benchmarks show that our method matches or outperforms complex trained judges (e.g., AutoJudge), offering superior robustness to domain shifts and eliminating the supervision bottleneck entirely.

## 1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in reasoning and generation (OpenAI et al., 2024; DeepSeek-AI et al., 2025; Grattafiori et al., 2024; Yang et al., 2025), yet their deployment is severely constrained by the high latency and memory bandwidth costs of autoregressive decoding (Pope et al., 2023; Ivanov et al., 2021; Naveed et al., 2025). To mitigate this, Speculative Decoding (SD) has emerged as a standard acceleration paradigm (Leviathan et al., 2023; Chen et al., 2023; Xia et al., 2024; Sun et al., 2025). By leveraging a smaller draft model to propose candidate token sequences that are verified in parallel by the larger target model, SD converts memory-bound sequential generation into compute-bound efficient parallel verification.

Standard SD, however, adheres to a strict lossless criterion. It rejects any draft token that does not strictly align with the target model’s distribu-

tion. This verification is often overly conservative, rejecting semantically equivalent tokens (e.g., “6+1=7” vs. “6 plus 1 equals 7”) and limiting potential speedups. To address this, recent research has pivoted towards *Judge Decoding* (Bachmann et al., 2025), a lossy variant that employs a learned classifier (a “judge”) to assess the semantic validity of draft tokens. These methods rely on expensive supervision (detailed in Section 3), including manual annotation (Bachmann et al., 2025) or heuristic mining (Garipov et al., 2025; Yoon et al., 2025; Fu et al., 2025), to teach a classifier to distinguish between “critical” errors and harmless deviations.

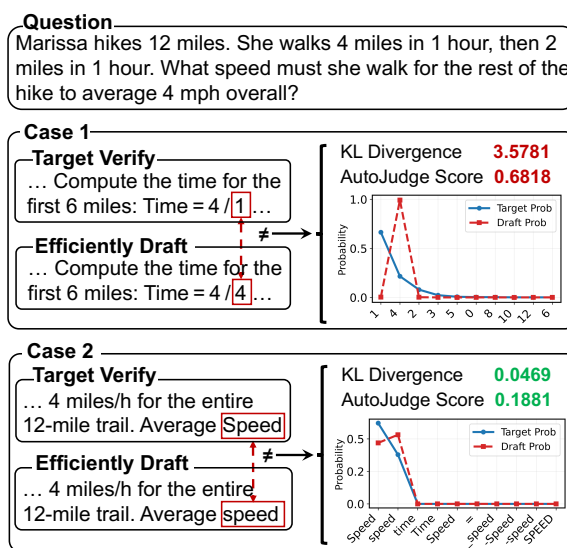


Figure 1: Relationship between AutoJudge score and token-level KL divergence. Higher AutoJudge scores indicate more critical tokens, which coincide with larger KL divergence and stronger target–draft disagreement (Case 1: inconsistent key numbers). Conversely, low scores correspond to small KL divergence and minor, non-semantic deviations (Case 2: capitalization only).

Despite the empirical success of judge decoding, the underlying nature of the “criticality” captured by these methods remains opaque. In this work, we investigate the relationship between the learned

scoring mechanism of the classifier and the intrinsic distributional statistics of the draft and target models, uncovering a fundamental connection that prior works have overlooked. As illustrated in Figure 1, taking AutoJudge (Garipov et al., 2025) as a representative instance, we observe that its assigned criticality scores exhibit a high degree of alignment with the token-level Kullback-Leibler (KL) divergence. In Case 1, where the draft model introduces a factual error by miscalculating the time (predicting “4/4” instead of the correct “4/1”), this discrepancy manifests as a sharp distributional shift between the draft and target models, resulting in both a high AutoJudge score and a large KL divergence. Conversely, in Case 2, a non-semantic capitalization difference triggers neither a high judge score nor a significant distributional divergence.

These observations motivate a first-principle hypothesis: *the “criticality” learned by complex judges is already inherently encoded in the draft–target distributional divergence.* We argue that the scoring function acquired through costly supervision (e.g., in AutoJudge) is essentially a proxy for intrinsic model disagreement. By bridging learned scoring mechanisms with these intrinsic statistics, we propose that a simple, training-free distributional check can effectively substitute for judges that rely on complex supervision signal mining. We ground this proposal theoretically by proving that learned judges and intrinsic divergence share identical logit primitives (linear vs. quadratic), suggesting that the efficacy of expensive supervision can be achieved through purely statistical means while naturally avoiding the generalization fragility of task-specific training. Our contributions are summarized as follows:

1. We empirically revisit judge decoding and find that supervision-trained criticality scores are strongly aligned with draft–target distributional disagreement at the token level, suggesting the “criticality” signal is model-intrinsic.
2. We theoretically establish a structural correspondence between learned judges and KL divergence, proving that the learned “criticality” relies on the same logit primitives as intrinsic divergence metrics.
3. Guided by our theoretical findings, we propose a training-free verification method. Empirical validation confirms that simple distributional metrics (e.g., KL divergence) effectively replace costly trained classifiers, achieving comparable or superior performance.

## 2 Background

To contextualize our approach, we briefly review the paradigm of speculative decoding and its recent evolution towards semantic-level verification, specifically judge decoding.

### 2.1 Speculative Decoding

To accelerate the memory-bound autoregressive inference of LLMs, SD employs a *draft-then-verify* strategy to increase concurrency (Chen et al., 2023; Leviathan et al., 2023). Instead of generating one token at a time, a smaller *draft* model  $\mathcal{M}_D$  first autoregressively produces a candidate block of  $\gamma$  tokens,  $\mathbf{d}_{t:t+\gamma-1}$ , given a prefix  $\mathbf{x}_{<t}$ . The large *target* model  $\mathcal{M}_T$  then evaluates all candidates in a single parallel forward pass. To ensure the output distribution matches  $\mathcal{M}_T$  exactly (lossless decoding), a rejection sampling mechanism is applied: a token  $d_{t+i}$  is accepted with probability  $\min(1, \frac{P_T(d_{t+i})}{P_D(d_{t+i})})$ . This paradigm allows the system to generate multiple tokens per expensive model call, effectively converting memory-bound sequential operations into compute-bound parallel verification.

### 2.2 Judge Decoding

While SD guarantees mathematical equivalence to the target distribution, this strict alignment can be overly conservative. For instance, a draft “6+1=7” might be rejected if the target model prefers “6 plus 1 equals 7”, despite their semantic equivalence. This strictness limits the effective speedup.

To address this, recent works have proposed judge decoding (Bachmann et al., 2025; Garipov et al., 2025). The core motivation stems from the insight that a model’s reaction to processing a token reveals more than just softmax probabilities. Specifically, the last hidden layer embeddings of erroneous tokens effectively “flag” errors and contradictions, reflecting the model’s intrinsic tendency to rectify mistakes immediately (Bachmann et al., 2025; Servedio et al., 2025; Azaria and Mitchell, 2023). Leveraging this latent error-signaling behavior, judge decoding employs a learnable verifier (or “judge”)  $f_\phi$ , typically a lightweight classifier on top of the embeddings  $\mathbf{h}$ , to assess the validity of the drafted block. A candidate is accepted if  $f_\phi(\mathbf{h}) > \tau$  for a predefined threshold  $\tau$ . By shifting from rigid probability comparisons to analyzing these internal validity signals, judge decoding optimizes the trade-off between throughput and generation fidelity, accepting plausibly correct to-

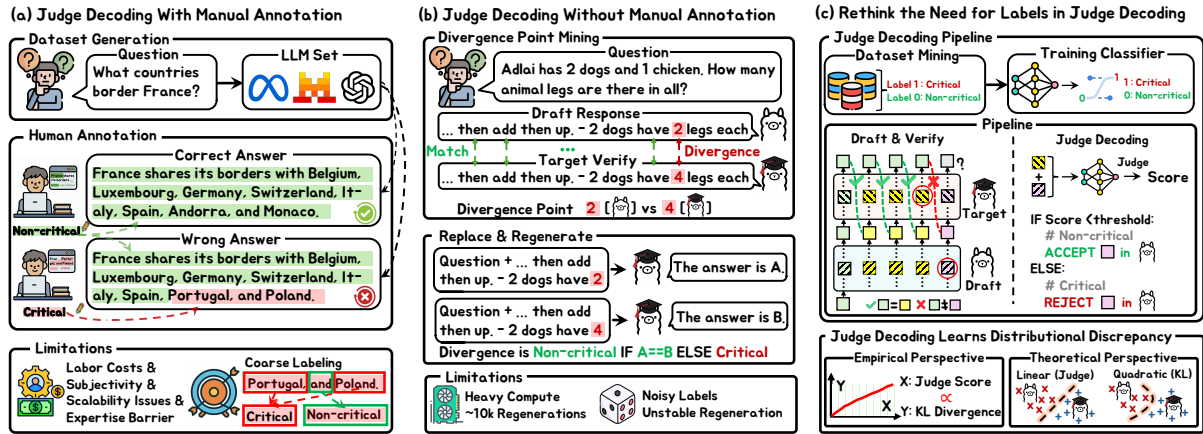


Figure 2: Judge decoding as learning distributional discrepancy. (a) Manual-annotation pipeline. (b) Divergence-point mining without manual labels; (c) We show that a linear judge’s score is empirically correlated with and theoretically connected to distributional divergence (e.g., KL divergence).

159 kens that standard SD would reject. The pipeline of judge decoding is depicted in Figure 2 (c).

### 161 3 The Bottlenecks of Supervision in Judge Decoding

162 While judge decoding represents a paradigm shift in speculative decoding by introducing semantic verification, its efficacy is fundamentally constrained by the quality and acquisition cost of the supervision signals used to train the judge. As shown in Figure 2, current approaches typically derive supervision either from expensive manual annotation or heuristic model rollouts, inevitably inheriting the generalization fragility of task-specific training. This section critically examines these two paradigms to highlight the trade-offs between annotation cost, label granularity, and signal stability.

#### 175 3.1 Judge Decoding With Manual Annotation

176 [Bachmann et al. \(2025\)](#) introduced the judge decoding framework as a verification paradigm that augments the target model with a lightweight classifier to assess token acceptability. This design effectively highlights the promise of learning correctness-aware verification beyond traditional probability alignment. However, two critical limitations hinder its widespread adoption:

184 1. **Costly and Misaligned Supervision.** Training relies on manual annotations of factual or reasoning mistakes. This process is not only prohibitively expensive but also prone to *alignment mismatch*: human annotators judge plausibility based on external knowledge, which may differ from the model’s intrinsic “knowledge

191 boundary”. A token might be factually true but out-of-distribution for the model, or vice versa. 192  
 2. **Coarse-grained Span Labeling.** Existing 193 schemes typically employ a span-based labeling strategy: once an error occurs, all subsequent 194 tokens are labeled as **Critical Tokens** (negative 195 samples). As shown in Figure 3, this design 196 fails to isolate the **Logic-pivoting Token**—the 197 specific token that triggers the reasoning shift. 198 Once the answer includes “Portugal”, every 199 subsequent token is labeled as critical, even 200 though only “Portugal” and “Poland” are the 201 true logic-pivoting tokens. 202  
 203

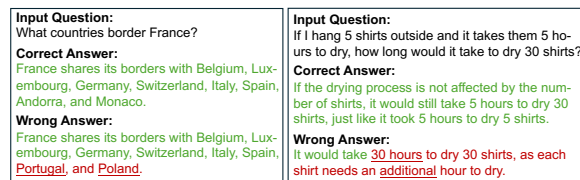


Figure 3: Illustration of overextended labeling boundaries in manual annotation, adapted from [Bachmann et al. \(2025\)](#). Key: **Non-critical Tokens**, **Critical Tokens**, **Logic-pivoting Tokens**. The span-based labeling (red) obscures the true logic-pivoting tokens (red underline), creating noisy supervision signals.

#### 204 3.2 Judge Decoding Without Manual Annotation

205 To mitigate human dependency, [Garipov et al. \(2025\)](#) proposed AutoJudge, which mines critical 206 tokens by checking if a token substitution alters 207 the final answer. While this approach automates 208 data collection, it introduces new computational 209 and statistical challenges: 210  
 211

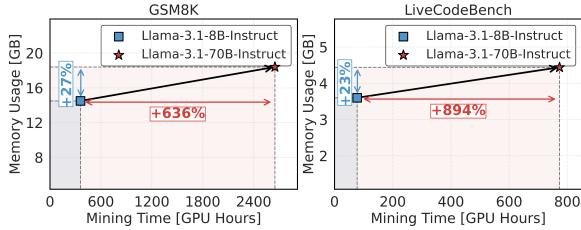


Figure 4: Efficiency analysis of AutoJudge dataset mining. The substantial growth in GPU hours and memory footprint for larger models highlights a significant scalability bottleneck.

- Prohibitive Mining Costs.** AutoJudge necessitates counterfactual rollouts at each divergence point. Figure 4 shows that mining supervision for GSM8K with a 70B model consumes about 2,700 GPU hours (NVIDIA L40). This prohibitive cost of data mining undermines the efficiency benefits of speculative decoding and hinders scalability, implicitly constraining the judge’s generalization capabilities due to the difficulty of covering diverse data distributions.
- Stochastic Instability.** LLM generation is inherently stochastic. A token labeled as “critical” in one rollout might be deemed “non-critical” in another simply due to inherent non-determinism in the generation process. Figure 5 reveals that only 25.4% of tokens maintain consistent criticality across four trials. This high variance injects significant label noise, causing the judge to learn spurious correlations rather than distinguishing between *causal logic errors* and *aleatoric uncertainty*.

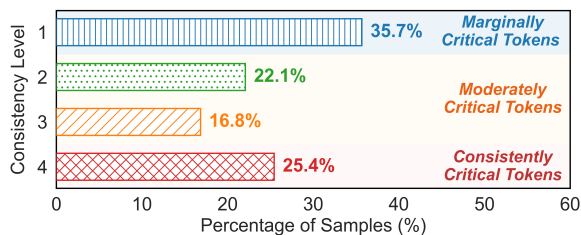


Figure 5: Consistency analysis of critical tokens. The low percentage of consistently critical tokens (Level 4) indicates that heuristic mining is heavily influenced by generation randomness.

**Summary.** Current paradigms face a dilemma: manual annotation provides stable labels but is labor-intensive and coarse, while heuristic mining offers automation but is plagued by high computational costs and stochastic noise. These limitations underscore the need for an efficient and robust al-

ternative, motivating the following section.

## 4 Can We Bypass Data Annotation?

The revisiting analysis in Section 3 exposes critical bottlenecks in current paradigms, thereby naturally leading to a key question: *Must the identification of critical tokens rely on external supervision, or can such signals be revealed directly through the models themselves?*

### 4.1 Empirical Perspective

To answer the above question, we compare the “criticality” captured by the classifier in AutoJudge against the intrinsic probabilistic states derived from the target distribution  $\mathbf{p}_t = \mathcal{M}_T(\mathbf{x}_{<t})$  and the draft distribution  $\mathbf{q}_t = \mathcal{M}_D(\mathbf{x}_{<t})$ . We quantify these states using the following two metrics:

- Uncertainty (Entropy)** quantifies the predictive uncertainty of the model. For any distribution  $\pi \in \{\mathbf{p}_t, \mathbf{q}_t\}$ , the entropy is defined as:

$$H(\pi) = - \sum_{v \in \mathcal{V}} \pi(v) \log \pi(v), \quad (1)$$

where higher entropy indicates a more dispersed probability distribution over the vocabulary  $\mathcal{V}$ .

- Disagreement (KL Divergence)** captures the structural deviation between the draft and target reasoning paths via:

$$D_{\text{KL}}(\mathbf{p}_t || \mathbf{q}_t) = \sum_{v \in \mathcal{V}} \mathbf{p}_t(v) \log \frac{\mathbf{p}_t(v)}{\mathbf{q}_t(v)}, \quad (2)$$

which quantifies the distributional disagreement between the two models.

Using these metrics, we investigate the correlation between the learned criticality scores from AutoJudge and intrinsic statistics on the GSM8K dataset. Specifically, we stratify the classifier’s output probabilities, where higher values indicate greater importance to the final answer, into equally spaced bins and compute the average value of the intrinsic indicators within each bin. As shown in Figure 6, we observe a strong positive correlation: the average KL divergence rises consistently as the classifier assigns higher criticality scores. This implies that influential tokens typically coincide with regions of high distributional disagreement between the target and draft models. Consequently, the token-level influence that AutoJudge learns to approximate via costly supervision appears to be inherently encoded in the divergence between the models’ probability distributions.

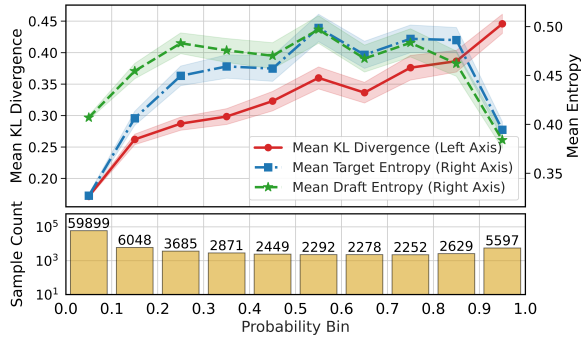


Figure 6: Correlation between supervised criticality scores and intrinsic model statistics on GSM8K. Samples are stratified into 10 bins based on AutoJudge scores (x-axis). **Top:** The mean KL divergence (red line) shows a clear upward trend as criticality increases, whereas model entropy (dashed lines) shows no significant correlation. **Bottom:** The sample count distribution across probability bins (i.e., AutoJudge scores).

**Key Insight.** The observed correlation suggests an intrinsic connection between criticality and the model’s predictive uncertainty. Rather than requiring additional supervision, this relationship indicates that the existing probabilistic behavior of the models conveys informative cues for identifying critical tokens. Thus, the KL divergence serves as a training-free proxy for locating logic pivots. To further enhance robustness in practice, we employ a lightweight confidence mask: when the target model exhibits high certainty (i.e., the top-1 probability exceeds 0.9), the system defaults to standard speculative sampling to ensure precision, reserving the KL-based relaxation specifically for tokens where the model faces genuine uncertainty.

## 4.2 Theoretical Perspective

This section characterizes the theoretical alignment between KL-based thresholding and the AutoJudge linear classifier. We first summarize the main intuition in the takeaway below, and then formalize it in Theorem 4.1.

**Takeaway.** AutoJudge and KL-based thresholding rely on the same signal, namely how much the target model shifts the draft model’s relative token preferences, expressed via  $\Delta_{ij}(x)$  (defined in Eq. (3)). AutoJudge learns a *linear* decision rule over these shifts, while the KL divergence provides a *quadratic* aggregation of their overall magnitude. As a result, KL-based thresholding can act as a proxy for the same “logic pivots” without dataset mining.

To facilitate the following analysis, we first introduce some necessary notation. Let  $x = [h_t; h_d]$  be the concatenated feature vector, where  $h_t, h_d \in \mathbb{R}^d$  denote the hidden states of the target and draft models. The corresponding logits are given by  $z_t = W_t h_t + b_t$  and  $z_d = W_d h_d + b_d$ , yielding the output distributions  $P_t = \text{softmax}(z_t)$  and  $P_d = \text{softmax}(z_d)$ . The theoretical link between the KL measure and the trained classifier is the pairwise logit-gap difference  $\Delta_{ij}(x)$ , defined for any vocabulary indices  $(i, j)$  as

$$\Delta_{ij}(x) := (z_t(i) - z_t(j)) - (z_d(i) - z_d(j)), \quad (3)$$

a fundamental linear primitive that captures the shift in relative preference between tokens when transitioning from the draft to the target model.

**Theorem 4.1** (Structural Correspondence of KL and Linear Classifiers). *The empirical alignment between KL-based thresholding and the trained linear classification stems from their shared dependence on the primitives  $\Delta_{ij}(x)$ : (i) **KL Divergence as Quadratic Aggregation:** Under a second-order expansion, the KL divergence acts as a weighted quadratic sum of the primitives  $\Delta_{ij}(x)$ . In this view, KL-based thresholding defines a quadratic decision boundary in the primitive space, sensing the total magnitude of distribution shifts. (ii) **Trained Classifier as Linear Partitioning:** The linear classifier identifies critical tokens by defining a linear decision surface in the space of primitives  $\Delta_{ij}(x)$ , partitioning the feature space based on whether token mismatches lead to a deviation in output quality.*

*Proof.* See Appendix A.1 for details.  $\square$

Theorem 4.1 attributes the stable agreement between both methods to their shared foundation in the linear primitives  $\Delta_{ij}(x)$ , differing only in whether they employ quadratic or linear geometry to distinguish token criticality.

## 5 Experiment

### 5.1 Experimental Setup and Baselines

**Benchmarks and Models.** We adopt standard speculative decoding protocols (Garipov et al., 2025) across four benchmarks: GSM8K (Cobbe et al., 2021) and MATH-500-Hard (Hendrycks et al., 2021) for mathematics, LiveCodeBench (Jain et al., 2025) for coding, and MMLU-Pro (Wang et al., 2024) for comprehensive

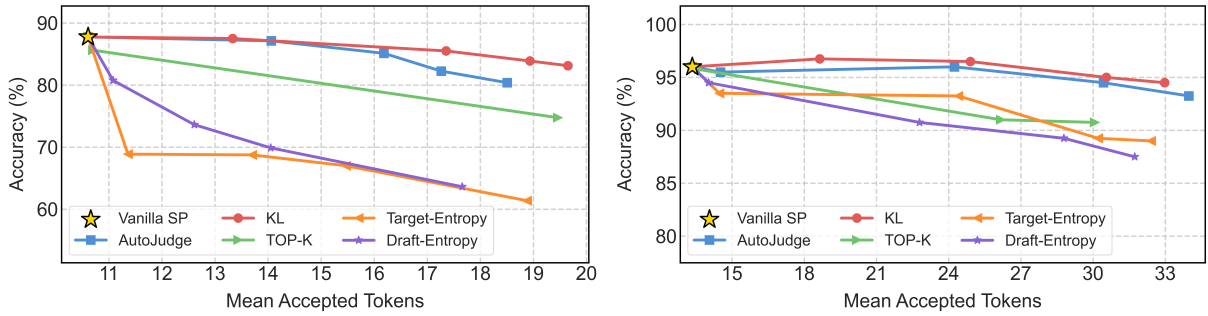


Figure 7: Accuracy and MAT on GSM8K for Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct (left) and Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct (right).

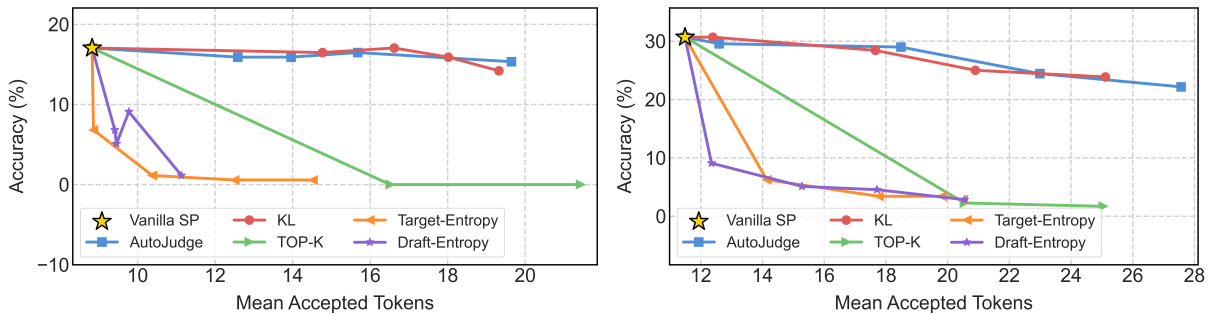


Figure 8: Accuracy and MAT on LiveCodeBench for Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct (left) and Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct (right).

337 knowledge. Our primary model pairs are  
 338 Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct and  
 339 Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct  
 340 (Grattafiori et al., 2024). Additionally, we  
 341 evaluate the Qwen3-0.6B/Qwen3-8B pair (Yang  
 342 et al., 2025) to investigate performance on  
 343 reasoning-specialized models.

344 **Baselines and Metrics.** We compare our method  
 345 against five baselines: (i) Vanilla SP (Chen et al.,  
 346 2023) (standard speculative decoding); (ii) Top-  
 347 K (Bachmann et al., 2025), which relaxes veri-  
 348 fication based on token ranking; (iii) AutoJudge  
 349 (Garipov et al., 2025), a training-based classi-  
 350 fier approach; and two entropy-based methods,  
 351 (iv) Target-Entropy and (v) Draft-Entropy (Wang  
 352 et al., 2025). Evaluation metrics include Mean  
 353 Accepted Tokens (MAT) for efficiency, bench-  
 354 mark-specific scores for quality, and end-to-end wall-  
 355 clock speedup relative to Vanilla SP implemented  
 356 in vLLM (Kwon et al., 2023). Further details are  
 357 in Appendix A.2.

## 358 5.2 Main Results

359 **Superior Efficiency-Accuracy Trade-off.** We  
 360 first compare methods on GSM8K and Live-  
 361 CodeBench (Figures 7-8). The training-free KL

362 thresholding exhibits highly consistent trends with  
 363 the training-based AutoJudge, often outperforming  
 364 it slightly. Specifically, on the Llama-1B/8B pair  
 365 for GSM8K, KL thresholding improves MAT by  
 366 **63.5%** (10.61→17.35) over Vanilla SP with only a  
 367  $\sim 2\%$  accuracy drop. Similarly, on the 8B/70B pair,  
 368 MAT increases by **128.8%** (13.35→30.55) with  
 369 a negligible  $\sim 1\%$  accuracy decrease. In contrast,  
 370 baselines like Top-K and entropy-based strategies  
 371 suffer from a much sharper accuracy degradation  
 372 as MAT increases. A similar MAT-accuracy trade-  
 373 off is also observed on LiveCodeBench, where  
 374 code generation requires strict precision. KL  
 375 thresholding improves MAT by **104.0%** (1B/8B,  
 376 8.83→18.01) and **53.7%** (8B/70B, 11.49→17.66)  
 377 with only minimal performance loss. Its perfor-  
 378 mance is comparable to AutoJudge, and is even  
 379 slightly better in some settings. In comparison,  
 380 Top-K and entropy-based strategies often fail to  
 381 preserve code correctness, frequently resulting in  
 382 non-executable programs. Overall, the consistently  
 383 similar trends observed between KL thresholding  
 384 and AutoJudge suggest that KL thresholding can  
 385 serve as a strong and effective proxy for the verifi-  
 386 cation signal, which aligns well with our motivation  
 387 outlined in Section 4 to leverage model-intrinsic  
 388 signals for judge decoding.

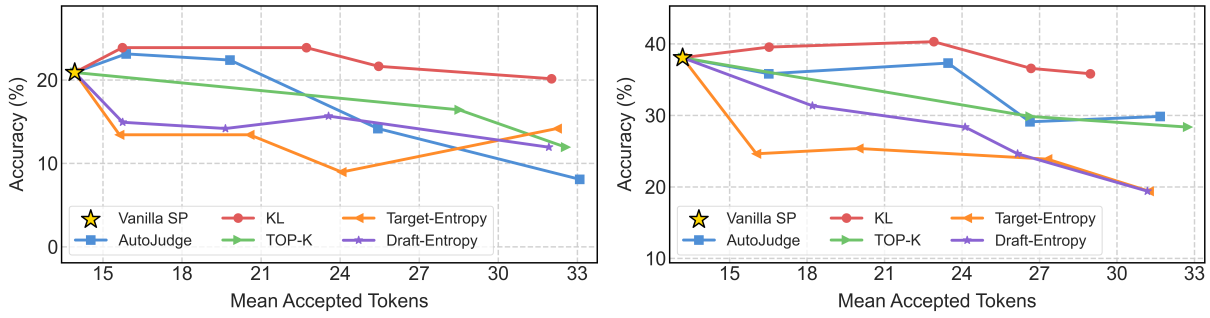


Figure 9: Accuracy and MAT on MATH-500-Hard for Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct (left) and Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct (right).

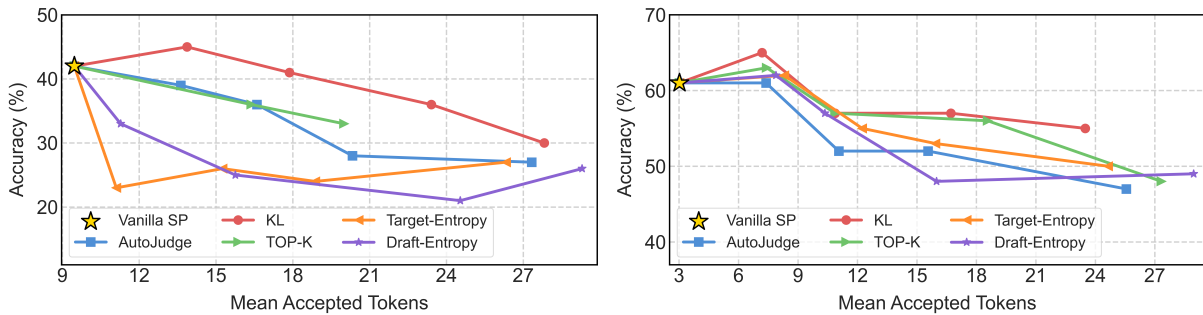


Figure 10: Accuracy and MAT on MMLU-Pro for Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct (left) and Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct (right).

**Robustness to Domain Shift.** To evaluate generalization, we test on MATH-500-Hard and MMLU-Pro benchmarks (Figures 9-10) using the AutoJudge classifier trained only on GSM8K, as domain-specific supervision is often unavailable. The results indicate that KL thresholding demonstrates superior robustness compared to AutoJudge. Specifically, on MATH-500-Hard, KL thresholding improves MAT from 13.93 to 32.02 (1B/8B) with a negligible  $<1\%$  performance drop, and also increases MAT from 13.18 to 26.67 (8B/70B) with about a 1.5% performance drop. Conversely, the GSM8K-trained AutoJudge classifier becomes brittle under this domain shift, showing clear accuracy degradation comparable to the weaker Top-K baseline. This observation confirms that simple, training-free distributional statistics serve as a more reliable proxy for verification signals than classifiers trained on limited domains.

**Real-world Speedup in vLLM.** We integrate KL thresholding into vLLM framework to benchmark real-world end-to-end latency (as shown in Table 1). Consistent with our previous MAT-Accuracy findings, KL thresholding effectively matches the speed-quality trade-off of the computationally expensive AutoJudge. It delivers a  $1.3\times$

speedup over the Vanilla SP baseline while simultaneously keeping accuracy within a relatively small 1%–2% margin. This result reinforces that costly critical-token mining is unnecessary for achieving practical inference acceleration in demanding production environments.

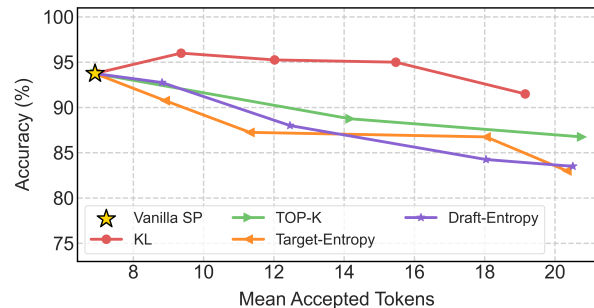


Figure 11: Accuracy and MAT on GSM8K for Qwen3-0.6B/Qwen3-8B.

**Efficiency in Long-Chain Reasoning.** AutoJudge’s inherent reliance on critical-token mining is particularly computationally prohibitive for reasoning models, where complex multi-step deliberation produces long trajectories with frequent draft-target divergence. We empirically evaluate KL thresholding on the Qwen3-0.6B/Qwen3-8B

Table 1: Inference deployment results with vLLM on GSM8K for Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct (left) and Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct (right), tested on  $8 \times$  V100 GPUs. Speed is measured in tokens/s, and *Speedup* is reported relative to standard speculative decoding (i.e., Vanilla SP).

Method	Metric	Llama-3.2-1B-Instruct & Llama-3.1-8B-Instruct				
AutoJudge	Threshold	0.01	0.07	0.09	0.14	0.22
	Acc	82.50%	80.29%	79.39%	78.40%	75.59%
	Speed	39.11	45.06	47.12	48.87	53.70
	Speedup	1.02 $\times$	1.17 $\times$	1.23 $\times$	1.27 $\times$	1.40 $\times$
KL	Threshold	0.10	0.30	0.40	0.50	0.90
	Acc	82.77%	80.96%	81.07%	79.46%	77.13%
	Speed	39.58	45.35	48.35	50.96	61.45
	Speedup	1.03 $\times$	1.18 $\times$	1.26 $\times$	1.33 $\times$	1.60 $\times$
Vanilla SP	Acc	82.50%				
	Speed	38.40				

Method	Metric	Llama-3.1-8B-Instruct & Llama-3.1-70B-Instruct				
AutoJudge	Thr	0.04	0.07	0.09	0.14	0.22
	Acc	92.77%	92.54%	92.09%	91.08%	89.96%
	Speed	27.34	29.55	30.98	32.57	35.07
	Speedup	1.13 $\times$	1.22 $\times$	1.28 $\times$	1.35 $\times$	1.45 $\times$
KL	Thr	0.20	0.40	0.50	0.90	1.40
	Acc	92.69%	91.94%	91.88%	91.28%	90.27%
	Speed	27.21	30.10	31.49	33.21	35.10
	Speedup	1.12 $\times$	1.24 $\times$	1.30 $\times$	1.37 $\times$	1.45 $\times$
Vanilla SP	Acc	93.00%				
	Speed	24.19				

pair (Figure 11). Remarkably, KL thresholding achieves a **2.77 $\times$  increase in MAT** (6.91 $\rightarrow$ 19.15) relative to Vanilla SP with only a marginal  $\sim 2\%$  performance drop. This evidence indicates that the training-free KL thresholding effectively handles the complex divergence patterns of reasoning models without the need for intricate and expensive supervision mining.

## 6 Related Work

**Speculative Decoding.** To accelerate LLM inference, speculative decoding employs a “draft-then-verify” principle (Chen et al., 2023; Leviathan et al., 2023), where a lightweight draft model rapidly generates candidate tokens that are subsequently verified efficiently in parallel by the larger target model. To further enhance drafting efficiency and avoid separate draft models, *training-based* approaches have been proposed to modify the target model’s architecture. Prominent examples include Medusa (Cai et al., 2024), which augments the model with multiple decoding heads, and the EAGLE series (Li et al., 2024b,a, 2025), which accelerates drafting via feature-level autoregression and dynamic tree structures. Conversely, *training-free* methods eliminate the need for additional parameters by exploiting the inherent redundancy in text. These strategies rely on retrieval or pattern matching, such as the N-gram based Lookahead decoding (Fu et al., 2024; Zhao et al., 2024), and retrieval-assisted generation like REST (He et al., 2024), Token Recycling (Luo et al., 2025) and Suffix Decoding (Oliaro et al., 2024; Hu et al., 2025).

**Judge Decoding.** Standard speculative decoding suffers from a verification criterion that rejects semantically equivalent but lexically distinct tokens. To address this, Judge Decoding (Bachmann et al., 2025) relaxes exact matching by employing a judge

model to evaluate the semantic validity of draft tokens, drawing inspiration from the LLM-as-a-judge framework (Gu et al., 2024; Bavaresco et al., 2025). Subsequent research focuses on automating the identification of “critical tokens”, namely divergences that genuinely harm performance. AutoJudge (Garipov et al., 2025) automates data collection by pinpointing critical tokens that degrade model accuracy. SelfJudge (Yoon et al., 2025) further refines this detection by calculating a semantic preservation score over the subsequent  $N$  tokens to determine criticality. Recent works introduce more granular verification based on utility and reasoning paths. Pivot-Aware Speculative Decoding (Ziashahabi et al., 2025) assesses token criticality by estimating the probability of generating correct answers via multiple sampled future paths. Similarly, R2R (Fu et al., 2025) distinguishes benign variations from logical errors by generating short continuations from both the draft and target tokens, employing a verifier to check for significant semantic divergence between the two paths.

## 7 Conclusion

This work bridges the gap between learned judge decoding and model-intrinsic statistics. We demonstrate that the costly “criticality” signal used in prior work is largely redundant to the distributional disagreement between models. By replacing supervised classifiers with a simple, training-free KL threshold, we achieve comparable or superior efficiency. Furthermore, our theoretical analysis unifies these seemingly distinct approaches, proving that KL screening and linear classifiers operate on the same underlying logit primitives. Our experiments validate that efficient judge decoding does not require complex supervision, offering a streamlined path for future deployment.

## 502 Limitations

503 We acknowledge some limitations in our work.  
504 First, due to computational constraints, we have  
505 not yet evaluated our approach on extremely large-  
506 scale models such as Llama-3.1-405B-Instruct,  
507 which would be essential for comprehensively vali-  
508 dating scalability and real-world applicability. Sec-  
509 ond, the KL divergence computation introduces ad-  
510 ditional computational overhead during inference.  
511 We have not implemented highly optimized cus-  
512 tom Triton kernels to optimize memory access and  
513 enable operator fusion, which could significantly  
514 reduce this cost. As a result, our current imple-  
515 mentation may bottleneck the potential speedups  
516 achievable in modern high-performance inference  
517 frameworks like vLLM.

## 518 Ethical Considerations

519 The datasets utilized in this work are derived solely  
520 from publicly accessible benchmarks, ensuring that  
521 no sensitive or private information is included.  
522 Moreover, all LLMs and baseline methods em-  
523 ployed in our experiments are also available to  
524 the public. We have taken care to acknowledge  
525 the original authors by properly citing their work.  
526 Regarding the preparation of this manuscript, we  
527 acknowledge the use of LLMs solely for language  
528 polishing and generating minor graphical icons for  
529 the framework diagram. They did not contribute  
530 to the research design, data analysis, or scientific  
531 claims presented in this paper. All ideas, analyses,  
532 and conclusions remain the sole and original work  
533 of the authors.

## 534 References

535 Amos Azaria and Tom Mitchell. 2023. The internal  
536 state of an LLM knows when it’s lying. In *Proc. Conf.*  
537 *Empirical Methods in Natural Language Processing*.  
538 Gregor Bachmann, Sotiris Anagnostidis, Albert  
539 Pumarola, Markos Georgopoulos, Artsiom  
540 Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali  
541 Thabet, and Jonas Kohler. 2025. Judge Decoding:  
542 Faster speculative sampling requires going beyond  
543 model alignment. In *Proc. Int. Conf. Learning*  
544 *Representations*.  
545 Anna Bavaresco, Raffaella Bernardi, Leonardo Berto-  
546 lazzi, Desmond Elliott, Raquel Fernández, Albert  
547 Gatt, Esam Ghaleb, Mario Giulianelli, Michael  
548 Hanna, Alexander Koller, Andre Martins, Philipp  
549 Mondorf, Vera Neplenbroek, Sandro Pezzelle, Bar-  
550 bara Plank, David Schlangen, Alessandro Suglia,  
551 Aditya K Surikuchi, Ece Takmaz, and Alberto

Testoni. 2025. LLMs instead of human judges? a  
552 large scale empirical study across 20 nlp evaluation  
553 tasks. In *ACL*. 554  
555 Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu  
556 Peng, Jason D. Lee, Deming Chen, and Tri Dao.  
557 2024. Medusa: Simple LLM inference accelera-  
558 tion framework with multiple decoding heads. In  
559 *arXiv:2401.10774*.  
560 Charlie Chen, Sebastian Borgeaud, Geoffrey Irv-  
561 ing, Jean-Baptiste Lespiau, Laurent Sifre, and  
562 John Jumper. 2023. Accelerating large language  
563 model decoding with speculative sampling. In  
564 *arXiv:2302.01318*.  
565 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian,  
566 Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias  
567 Plappert, Jerry Tworek, Jacob Hilton, Reiichiro  
568 Nakano, Christopher Hesse, and John Schulman.  
569 2021. Training verifiers to solve math word prob-  
570 lems. In *arXiv:2110.14168*.  
571 DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang,  
572 Junxiao Song, Ruoyu Zhang, et al. 2025. Deepseek-  
573 r1: Incentivizing reasoning capability in LLMs via  
574 reinforcement learning. In *arXiv:2501.12948*.  
575 Tianyu Fu, Yi Ge, Yichen You, Enshu Liu, Zhihang  
576 Yuan, Guohao Dai, Shengen Yan, Huazhong Yang,  
577 and Yu Wang. 2025. R2R: efficiently navigating  
578 divergent reasoning paths with small-large model  
579 token routing. In *arXiv:2505.21600*.  
580 Yichao Fu, Peter Bailis, Ion Stoica, and Hao  
581 Zhang. 2024. Break the sequential dependency  
582 of LLM inference using lookahead decoding. In  
583 *arXiv:2402.02057*.  
584 Roman Garipov, Fedor Velikonitvsev, Ivan Ermakov,  
585 Ruslan Svirschevski, Vage Egiazarian, and Max  
586 Ryabinin. 2025. Autojudge: Judge decoding without  
587 manual annotation. In *Advances in Neural Informa-*  
588 *tion Processing Systems*.  
589 Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri,  
590 Abhinav Pandey, et al. 2024. The llama 3 herd of  
591 models. In *arXiv:2407.21783*.  
592 Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan,  
593 Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen,  
594 Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun  
595 Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and  
596 Jian Guo. 2024. A survey on LLM-as-a-Judge. In  
597 *arXiv:2411.15594*.  
598 Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and  
599 Di He. 2024. REST: retrieval-based speculative de-  
600 coding. In *NAACL*.  
601 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul  
602 Arora, Steven Basart, Eric Tang, Dawn Song, and  
603 Jacob Steinhardt. 2021. Measuring mathematical  
604 problem solving with the math dataset. In *Advances*  
605 *in Neural Information Processing Systems*.

606	Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang,	Reiner Pope, Sholto Douglas, Aakanksha Chowdhery,	660
607	Cuiping Li, Hong Chen, and Jing Zhang. 2025. SAM	Jacob Devlin, James Bradbury, Jonathan Heek, Kefan	661
608	Decoding: Speculative decoding via suffix automa-	Xiao, Shivani Agrawal, and Jeff Dean. 2023. Effi-	662
609	ton. In <i>ACL</i> .	ciently scaling transformer inference. In <i>Proceedings</i>	663
		<i>of Machine Learning and Systems</i> .	664
610	Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang	Giovanni Servedio, Alessandro De Bellis, Dario Di	665
611	Li, and Torsten Hoeffler. 2021. Data movement is all	Palma, Vito Walter Anelli, and Tommaso Di Noia.	666
612	you need: A case study on optimizing transformers.	2025. Are the hidden states hiding something? Test-	667
613	In <i>Proceedings of Machine Learning and Systems</i> .	ing the limits of factuality-encoding capabilities in	668
		LLMs. In <i>ACL</i> .	669
614	Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia	Shengyin Sun, Yiming Li, Xing Li, Yingzhao Lian,	670
615	Yan, Tianjun Zhang, Sida Wang, Armando Solar-	Weizhe Lin, Hui-Ling Zhen, Zhiyuan Yang, Chen	671
616	Lezama, Koushik Sen, and Ion Stoica. 2025. Live-	Chen, Xianzhi Yu, Mingxuan Yuan, and Chen Ma.	672
617	codebench: Holistic and contamination free evalua-	2025. Scaling up, speeding up: A benchmark of spec-	673
618	tion of large language models for code. In <i>Proc. Int.</i>	ulative decoding for efficient LLM test-time scaling.	674
619	<i>Conf. Learning Representations</i> .	In <i>arXiv:2509.04474</i> .	675
620	Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying	Jikai Wang, Zhenxu Tian, Juntao Li, Qingrong Xia,	676
621	Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gon-	Xinyu Duan, Zhefeng Wang, Baoxing Huai, and Min	677
622	zalez, Hao Zhang, and Ion Stoica. 2023. Efficient	Zhang. 2025. Alignment-augmented speculative de-	678
623	memory management for large language model serv-	coding with alignment sampling and conditional ver-	679
624	ing with pagedattention. In <i>Proceedings of the 29th</i>	ification. In <i>Proc. Conf. Empirical Methods in Natu-</i>	680
625	<i>Symposium on Operating Systems Principles</i> .	<i>ral Language Processing</i> .	681
626	Yaniv Leviathan, Matan Kalman, and Yossi Matias.	Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni,	682
627	2023. Fast inference from transformers via specula-	Abhranil Chandra, Shiguang Guo, Weiming Ren,	683
628	tive decoding. In <i>Proc. Int. Conf. Machine Learning</i> .	Aaran Arulraj, Xuan He, Ziyang Jiang, Tianle Li, Max	684
629	Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang	Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue,	685
630	Zhang. 2024a. Eagle-2: Faster inference of language	and Wenhui Chen. 2024. MMLU-Pro: A more robust	686
631	models with dynamic draft trees. In <i>Proc. Conf. Em-</i>	and challenging multi-task language understanding	687
632	<i>pirical Methods in Natural Language Processing</i> .	benchmark. In <i>Advances in Neural Information Pro-</i>	688
		<i>cessing Systems</i> .	689
633	Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang	Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang,	690
634	Zhang. 2024b. Eagle: Speculative sampling requires	Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhi-	691
635	rethinking feature uncertainty. In <i>Proc. Int. Conf.</i>	fang Sui. 2024. Unlocking efficiency in large lan-	692
636	<i>Machine Learning</i> .	guage model inference: A comprehensive survey of	693
		speculative decoding. In <i>ACL</i> .	694
637	Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang	An Yang, Anfeng Li, Baosong Yang, Beichen Zhang,	695
638	Zhang. 2025. Eagle-3: Scaling up inference accelera-	Binyuan Hui, et al. 2025. Qwen3 technical report. In	696
639	tion of large language models via training-time test.	<i>arXiv:2505.09388</i> .	697
640	In <i>arXiv:2503.01840</i> .	Kanghoon Yoon, Minsub Kim, Sungjae Lee, Joonhyung	698
641	Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming	Lee, Sunghyeon Woo, Yeonjun In, Se Jung Kwon,	699
642	Zhang, Xuanyu Zhang, Qing Yang, and Dongliang	Chanyoung Park, and Dongsoo Lee. 2025. Selfjudge:	700
643	Xu. 2025. Turning trash into treasure: Accelerat-	Faster speculative decoding via self-supervised judge	701
644	ing inference of large language models with token	verification. In <i>arXiv:2510.02329</i> .	702
645	recycling. In <i>ACL</i> .	Yao Zhao, Zhitian Xie, Chen Liang, Chenyi Zhuang,	703
646	Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad	and Jinjie Gu. 2024. Lookahead: An inference ac-	704
647	Saqib, Saeed Anwar, Muhammad Usman, Naveed	celeration framework for large language model with	705
648	Akhtar, Nick Barnes, and Ajmal Mian. 2025. A com-	lossless generation accuracy. In <i>Proc. ACM Conf.</i>	706
649	prehensive overview of large language models. In	<i>Knowledge Discovery and Data Mining</i> .	707
650	<i>ACM Transactions on Intelligent Systems and Tech-</i>	Amir Ziashahabi, Yavuz Faruk Bakman, Duygu Nur	708
651	<i>nology</i> , volume 16, pages 1–72. Association for Com-	Yaldiz, Mostafa El-Khamy, Sai Praneeth Karim-	709
652	puting Machinery.	reddy, and Salman Avestimehr. 2025. Reject only	710
653	Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Au-	critical tokens: Pivot-aware speculative decoding. In	711
654	rick Qiao. 2024. SuffixDecoding: Extreme specu-	<i>arXiv:2511.00351</i> .	712
655	lative decoding for emerging ai applications. In		
656	<i>arXiv:2411.04975</i> .		
657	OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam		
658	Richardson, Ahmed El-Kishky, et al. 2024. Openai		
659	o1 system card. In <i>arXiv:2412.16720</i> .		

## A Appendix

### A.1 Proof of Theorem 4.1

To facilitate the derivation, we first review the notation and setup. Let the target and draft logits be  $z_t := W_t h_t + b_t$  and  $z_d := W_d h_d + b_d$  respectively, with corresponding probability distributions  $P_t := \text{softmax}(z_t)$  and  $P_d := \text{softmax}(z_d)$ . We define the concatenated hidden representation  $x := [h_t; h_d] \in \mathbb{R}^{2d}$  and the logit difference vector  $\delta := z_t - z_d$ . Consequently,  $\delta$  is affine in  $x$  and is given by  $\delta = Mx + c$ , where  $M := [W_t, -W_d]$  and  $c := b_t - b_d$ . For any vocabulary indices  $i, j \in \{1, \dots, V\}$ , the pairwise logit-gap difference is defined as  $\Delta_{ij}(x) := (z_t(i) - z_t(j)) - (z_d(i) - z_d(j))$ , which yields the  $\Delta_{ij}(x) = \delta_i - \delta_j$ . This linear primitive  $\Delta_{ij}(x)$  serves as the basis for our analysis.

**Proof Sketch of Theorem 4.1.** We ground the analysis in the affine primitives  $\Delta_{ij}(x)$ . By expanding the KL divergence via the Fisher information metric (§A.1.1), we decompose the divergence into a weighted quadratic aggregation of these primitives (§A.1.2). We subsequently demonstrate that discrete ranking inconsistencies necessitate boundary-crossing deviations (§A.1.3), thereby enforcing a quantitative lower bound on the divergence (§A.1.4). Finally, we prove that the linear classifier’s decision surface forms a linear superposition of this same primitive basis (§A.1.5), confirming that both mechanisms regulate the same logit-space deviations.  $\square$

#### A.1.1 KL as a Bregman divergence and Fisher second-order approximation

We define the log-sum-exp potential  $A(z) := \log \sum_{i=1}^V e^{z_i}$ , which satisfies  $\nabla A(z) = p$  and  $\nabla^2 A(z) = \text{diag}(p) - pp^\top$  for  $p = \text{softmax}(z)$ .

**Lemma A.1** (KL as a Bregman divergence). *Let  $p = \text{softmax}(z)$  and  $q = \text{softmax}(z')$ . Then  $D_{\text{KL}}(p||q) = A(z') - A(z) - \nabla A(z)^\top (z' - z)$ .*

*Proof.* By the log-softmax identities  $\log p_i = z_i - A(z)$  and  $\log q_i = z'_i - A(z')$ , we have:

$$\begin{aligned} D_{\text{KL}}(p||q) &= \sum_i p_i (\log p_i - \log q_i) \\ &= \sum_i p_i [(z_i - A(z)) - (z'_i - A(z'))] \\ &= (A(z') - A(z)) \sum_i p_i - \sum_i p_i (z'_i - z_i) \\ &= A(z') - A(z) - \nabla A(z)^\top (z' - z), \quad (4) \end{aligned}$$

where the last step follows from  $\sum_i p_i = 1$  and the fact that  $\nabla A(z) = p$ .  $\square$

**Lemma A.2** (Second-order expansion around  $z_d$ ). *Let  $P_d = \text{softmax}(z_d)$ ,  $P_t = \text{softmax}(z_t)$  and write  $z_t = z_d + \delta$ . Conditioned on accepted prefixes in speculative sampling, the two models are close so that  $\|\delta\|$  is small (large discrepancies are rejected and do not contribute). Then  $D_{\text{KL}}(P_t||P_d) = \frac{1}{2} \delta^\top F(P_d) \delta + O(\|\delta\|^3)$ , where  $F(P_d) = \nabla^2 A(z_d) = \text{diag}(P_d) - P_d P_d^\top$ .*

*Proof.* Applying Lemma A.1 with  $p = P_t$  and  $q = P_d$ , and noting  $z_d - z_t = -\delta$ , we have the identity  $D_{\text{KL}}(P_t||P_d) = A(z_d) - A(z_t) + P_t^\top \delta$ . We expand  $A(z_t)$  to the second order as  $A(z_t) = A(z_d) + P_d^\top \delta + \frac{1}{2} \delta^\top F(P_d) \delta + O(\|\delta\|^3)$ . Similarly, expanding  $P_t$  to the first order gives  $P_t = P_d + F(P_d) \delta + O(\|\delta\|^2)$ , which implies  $P_t^\top \delta = P_d^\top \delta + \delta^\top F(P_d) \delta + O(\|\delta\|^3)$ . Substituting these expressions into the KL identity leads to the cancellation of linear terms, yielding  $D_{\text{KL}}(P_t||P_d) = \frac{1}{2} \delta^\top F(P_d) \delta + O(\|\delta\|^3)$ .  $\square$

**Corollary A.3** (KL’s quadratic form controlled by a linear map of the concatenated state). *Recall that  $\delta = z_t - z_d$  and  $\delta = Mx + c$  with  $x = [h_t; h_d]$ . Combining the second-order expansion in Lemma A.2 with  $\delta = Mx + c$ , we obtain*

$$D_{\text{KL}}(P_t||P_d) \approx \frac{1}{2} (Mx + c)^\top F(P_d) (Mx + c). \quad (5)$$

#### A.1.2 Pairwise form: KL as a Fisher-weighted sum of primitives

**Lemma A.4** (Pairwise decomposition of the Fisher quadratic form). *Let  $p \in \mathbb{R}^V$  satisfy  $p_i \geq 0$  and  $\sum_{i=1}^V p_i = 1$ , and let  $F(p) = \text{diag}(p) - pp^\top$ . Then for any  $\delta \in \mathbb{R}^V$ ,*

$$\frac{1}{2} \delta^\top F(p) \delta = \frac{1}{4} \sum_{i=1}^V \sum_{j=1}^V p_i p_j (\delta_i - \delta_j)^2. \quad (6)$$

*Proof.* Expand the left-hand side using  $F(p) = \text{diag}(p) - pp^\top$ , we have

$$\begin{aligned} \delta^\top F(p) \delta &= \delta^\top \text{diag}(p) \delta - \delta^\top (pp^\top) \delta \\ &= \sum_{i=1}^V p_i \delta_i^2 - (p^\top \delta)^2 \quad (7) \end{aligned}$$

$$= \sum_{i=1}^V p_i \delta_i^2 - \left( \sum_{i=1}^V p_i \delta_i \right)^2. \quad (8)$$

Expand the right-side pairwise sum, we have

$$\begin{aligned}
& \sum_{i=1}^V \sum_{j=1}^V p_i p_j (\delta_i - \delta_j)^2 = \sum_{i,j} p_i p_j (\delta_i^2 + \delta_j^2 - 2\delta_i \delta_j) \\
& = \sum_{i,j} p_i p_j \delta_i^2 + \sum_{i,j} p_i p_j \delta_j^2 - 2 \sum_{i,j} p_i p_j \delta_i \delta_j \\
& = 2 \sum_i p_i \delta_i^2 - 2 \left( \sum_i p_i \delta_i \right)^2, \quad (9)
\end{aligned}$$

where the last equality follows from Eq. (8). Dividing both sides by 4 yields Eq. (6).  $\square$

**Corollary A.5** (KL as quadratic aggregation of  $\Delta_{ij}$ ). *Combining Lemma A.2 and Lemma A.4 with  $p = P_d$  and  $\Delta_{ij}(x) = \delta_i - \delta_j$ , we have  $D_{\text{KL}}(P_t \| P_d) \approx \sum_{i=1}^V \sum_{j=1}^V \alpha_{ij} \Delta_{ij}(x)^2$ , where  $\alpha_{ij} := \frac{1}{4} P_d(i) P_d(j)$ .*

### A.1.3 Top-K inconsistency and boundary-crossing primitive

The discrete supervision signal used for classifier training is defined through the consistency between the target and draft models' top-K prediction sets. Since LLMs assign most of the probability mass to a small subset of tokens, agreement of the top-K sets corresponds to similar generation behavior, whereas disagreement serves as an indicator of divergent outputs. A top- $k$  inconsistency implies the existence of a pair of logits whose pairwise difference crosses the ranking boundaries, yielding a lower bound on the primitive  $\Delta_{ij}(x)$ . Let  $T_k(x)$  and  $D_k(x)$  denote the index sets of the largest  $k$  logits under  $z_t$  and  $z_d$ , respectively. Let  $z_t^{(k)}$  and  $z_d^{(k)}$  denote the  $k$ -th largest logit values of  $z_t$  and  $z_d$ . Denote the top- $k$  margins by  $\gamma_t^{(k)} = z_t^{(k)} - z_t^{(k+1)}$  and  $\gamma_d^{(k)} = z_d^{(k)} - z_d^{(k+1)}$ .

**Proposition A.6** (Top- $k$  inconsistency implies a boundary-crossing pairwise gap). *If  $T_k(x) \neq D_k(x)$ , then there exist  $i \in T_k(x) \setminus D_k(x)$  and  $j \in D_k(x) \setminus T_k(x)$  such that  $\Delta_{ij}(x) \geq \gamma_t^{(k)} + \gamma_d^{(k)}$ .*

*Proof.* Since  $T_k(x) \neq D_k(x)$ , both  $T_k(x) \setminus D_k(x)$  and  $D_k(x) \setminus T_k(x)$  are non-empty. Select any  $i \in T_k(x) \setminus D_k(x)$  and  $j \in D_k(x) \setminus T_k(x)$ . For  $z_t$ ,  $i \in T_k(x)$  implies  $z_t(i) \geq z_t^{(k)}$ , and  $j \notin T_k(x)$  implies  $z_t(j) \leq z_t^{(k+1)}$ , we have

$$z_t(i) - z_t(j) \geq z_t^{(k)} - z_t^{(k+1)} = \gamma_t^{(k)}. \quad (10)$$

For  $z_d$ ,  $j \in D_k(x)$  implies  $z_d(j) \geq z_d^{(k)}$ , and  $i \notin D_k(x)$  implies  $z_d(i) \leq z_d^{(k+1)}$ , we have

$$z_d(i) - z_d(j) \leq z_d^{(k+1)} - z_d^{(k)} = -\gamma_d^{(k)}. \quad (11)$$

Combining the above inequalities Eq. (10) and Eq. (11) gives  $\Delta_{ij}(x) \geq \gamma_t^{(k)} + \gamma_d^{(k)}$ .  $\square$

Top- $k$  inconsistency therefore ensures the existence of a boundary-crossing primitive  $(i, j)$  whose logit-difference gap exceeds the sum of the target and draft margins. This provides the link between the discrete supervision condition and a continuous deviation in the primitive space, which is used in Appendix A.1.4 to establish a lower bound on the KL divergence.

### A.1.4 From boundary-crossing primitives to a KL lower bound

Proposition A.6 established that a top- $k$  mismatch implies the existence of a boundary-crossing primitive  $(i, j)$  whose pairwise logit-difference satisfies  $|\Delta_{ij}(x)| \geq \gamma_t^{(k)} + \gamma_d^{(k)}$ . Meanwhile, Corollary A.5 expresses the KL as a Fisher-weighted quadratic aggregation over all such primitives. Combining these two facts shows that the presence of a boundary-crossing primitive induces a quantitative lower bound on the overall KL divergence.

**Proposition A.7** (Boundary-crossing primitive implies a KL lower bound). *If for some index pair  $(i, j)$  we have  $|\Delta_{ij}(x)| \geq \gamma_t^{(k)} + \gamma_d^{(k)}$ , then based on the quadratic expansion in Lemma A.2,  $D_{\text{KL}}(P_t \| P_d) \gtrsim \alpha_{ij} (\gamma_t^{(k)} + \gamma_d^{(k)})^2$ , where  $\alpha_{ij} = \frac{1}{4} P_d(i) P_d(j)$ .*

*Proof.* From Corollary A.5, dropping the non-negative terms for pairs other than  $(i, j)$  yields  $D_{\text{KL}}(P_t \| P_d) \approx \sum_{u,v} \alpha_{uv} \Delta_{uv}(x)^2 \geq \alpha_{ij} \Delta_{ij}(x)^2 \geq \alpha_{ij} (\gamma_t^{(k)} + \gamma_d^{(k)})^2$ .  $\square$

This yields a lower bound on the KL divergence, showing that any top- $k$  inconsistency enforces a separation between the two predictive distributions.

### A.1.5 Why a linear classifier trained on "importance" aligns with KL screening

The subsequent lemma writes  $\Delta_{ij}(x)$  explicitly as an affine function of  $x$ .

**Lemma A.8** (Affine form of the primitives). *For any  $i, j \in \{1, \dots, V\}$  there exist  $a_{ij} \in \mathbb{R}^{2d}$  and  $\kappa_{ij} \in \mathbb{R}$  such that  $\Delta_{ij}(x) = a_{ij}^\top x + \kappa_{ij}$ .*

*Proof.* Let  $e_i \in \mathbb{R}^V$  denote the  $i$ -th standard basis vector. By definition,  $z_t = W_t h_t + b_t$  and  $z_d = W_d h_d + b_d$ , hence  $z_t(i) = e_i^\top (W_t h_t + b_t)$  and  $z_d(i) = e_i^\top (W_d h_d + b_d)$ . Therefore  $z_t(i) - z_t(j) = (e_i - e_j)^\top (W_t h_t + b_t)$  and  $z_d(i) - z_d(j) = (e_i - e_j)^\top (W_d h_d + b_d)$ . Subtracting the two identities gives  $\Delta_{ij}(x) = (e_i - e_j)^\top W_t h_t - (e_i - e_j)^\top W_d h_d + (e_i - e_j)^\top (b_t - b_d)$ . Define  $a_{ij} := [W_t^\top (e_i - e_j); -W_d^\top (e_i - e_j)]$  and  $\kappa_{ij} := (e_i - e_j)^\top (b_t - b_d)$ . Using  $x = [h_t; h_d]$  yields  $\Delta_{ij}(x) = a_{ij}^\top x + \kappa_{ij}$ .  $\square$

**Proposition A.9** (Representation of the classifier in the primitive basis). *Let  $\text{Cls}(x) = \sigma(w^\top x + b)$  be a linear classifier trained on the concatenated representation  $x$ . In LLMs, the vocabulary size  $V$  is typically much larger than the hidden dimension  $d$  ( $V \gg d$ ). Consequently, the set of primitive direction vectors  $\{a_{ij} \mid i, j = 1, \dots, V\}$  forms an overcomplete frame that effectively spans the feature space  $\mathbb{R}^{2d}$ . The classifier’s weight vector  $w$  can therefore be expressed as a linear combination of primitives  $w = \sum_{i=1}^V \sum_{j=1}^V \beta_{ij} a_{ij}$  for a set of coefficients  $\{\beta_{ij}\}$ . Accordingly, the classifier is equivalent to  $\text{Cls}(x) = \sigma\left(\sum_{i=1}^V \sum_{j=1}^V \beta_{ij} \Delta_{ij}(x) + b'\right)$ , for an adjusted bias term  $b'$ . This representation indicates that the decision surface can be well characterized by a linear combination of the primitives  $\Delta_{ij}(x)$ .*

*Proof.* Because the set  $\{a_{ij} \mid i, j = 1, \dots, V\}$  effectively spans the feature space  $\mathbb{R}^{2d}$ , the classifier weight vector  $w$  admits the expansion  $w = \sum_{i=1}^V \sum_{j=1}^V \beta_{ij} a_{ij}$ . Substituting the affine form of each primitive from Lemma A.8, we obtain

$$\begin{aligned} w^\top x &= \left( \sum_{i=1}^V \sum_{j=1}^V \beta_{ij} a_{ij} \right)^\top x = \sum_{i=1}^V \sum_{j=1}^V \beta_{ij} (a_{ij}^\top x) \\ &= \sum_{i=1}^V \sum_{j=1}^V \beta_{ij} (\Delta_{ij}(x) - \kappa_{ij}). \end{aligned} \quad (12)$$

By defining the bias  $b' := b - \sum_{i=1}^V \sum_{j=1}^V \beta_{ij} \kappa_{ij}$ , the classifier can be rewritten in terms of  $\Delta_{ij}(x)$  as  $w^\top x + b = \sum_{i=1}^V \sum_{j=1}^V \beta_{ij} \Delta_{ij}(x) + b'$ , which yields  $\text{Cls}(x) = \sigma\left(\sum_{i=1}^V \sum_{j=1}^V \beta_{ij} \Delta_{ij}(x) + b'\right)$ .  $\square$

Corollary A.5 provides the second-order primitive expansion of the KL divergence. Together with

Proposition A.9, this yields the following formal juxtaposition in the primitive coordinate system

$$\begin{cases} D_{\text{KL}}(P_t \| P_d) \approx \sum_{i=1}^V \sum_{j=1}^V \frac{1}{4} P_d(i) P_d(j) \Delta_{ij}(x)^2, \\ \text{Cls}(x) \equiv \sigma \left( \sum_{i=1}^V \sum_{j=1}^V \beta_{ij} \Delta_{ij}(x) + b' \right), \end{cases}$$

The first line corresponds to a weighted quadratic aggregation of the primitives  $\Delta_{ij}(x)$ , whereas the second line represents a linear aggregation over the same primitive basis, demonstrating that the classifier operates directly on the fundamental components of the KL divergence. Consequently, the structural correspondence asserted in Theorem 4.1 is formally established.

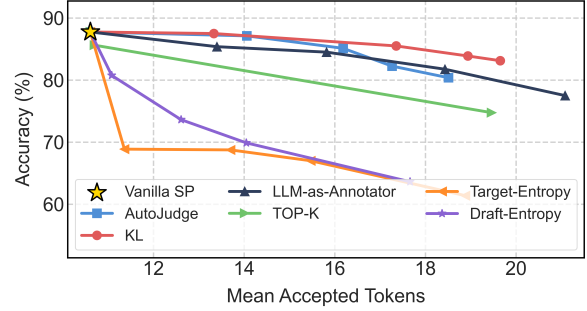


Figure 12: Accuracy and MAT on GSM8K for Llama-3.2-1B-Instruct and Llama-3.1-8B-Instruct (with Qwen3-Max as the annotator).

## A.2 Additional Experiment Details

We conduct experiments on four benchmarks: GSM8K and MATH-500-Hard for math reasoning, MMLU-Pro for broad-coverage knowledge and reasoning, and LiveCodeBench for code generation. GSM8K consists of grade-school math word problems that require multi-step numerical reasoning. We additionally evaluate on MATH-500-Hard, which we construct by selecting all Level-5 problems from MATH-500, to introduce a math distribution that differs from GSM8K and is substantially more challenging. MMLU-Pro is a more difficult variant of MMLU spanning a wide range of subjects. LiveCodeBench focuses on algorithmic code generation with execution-based evaluation. To manage end-to-end decoding cost, we run evaluation on held-out subsets. For GSM8K, we evaluate on 400 examples from the test split, while AutoJudge training uses the full GSM8K training split.

## Prompt for Critical-Token Annotation

### [System Prompt]

You are an expert annotator.

### [User Prompt]

You are an expert annotator. Your task is to identify the KEY REASONING ERROR tokens that cause the wrong answer to be incorrect.

#### INPUT:

Question: [to be inserted]  
Correct Answer: [to be inserted]  
Wrong Answer: [to be inserted]

#### INSTRUCTIONS:

1. Identify WHERE the logical/reasoning ERROR occurs (not just textual differences).
2. Mark ONLY the specific tokens that introduce the error.
3. There may be MULTIPLE error locations - mark each separately.
4. Ignore stylistic differences - focus on semantic/logical mistakes.
5. An error is a token that causes incorrect reasoning, wrong calculation, or false conclusion.

#### OUTPUT FORMAT (STRICT):

ANNOTATED\_ANSWER: [exact wrong answer text with markers inserted]

#### Use markers:

- [ERROR\_START] before the first token of each error
- [ERROR\_END] after the last token of each error
- Keep ALL original text and spacing
- Multiple errors = multiple [ERROR\_START]...[ERROR\_END] pairs

#### EXAMPLE 1:

Question: What is 5 + 3?  
Correct: 5 + 3 = 8  
Wrong: 5 + 3 = 9  
Output: ANNOTATED\_ANSWER: 5 + 3 = [ERROR\_START]9[ERROR\_END]

#### EXAMPLE 2:

Question: If I hang 5 shirts outside and it takes them 5 hours to dry, how long would it take to dry 30 shirts?  
Correct: It would still take 5 hours to dry 30 shirts, as drying is parallel.  
Wrong: It would take 30 hours to dry 30 shirts, as each shirt needs an additional hour to dry.  
Output: ANNOTATED\_ANSWER: It would take [ERROR\_START]30 hours to dry 30 shirts, as each shirt needs an additional hour to dry.[ERROR\_END]

#### EXAMPLE 3:

Question: What is the capital of France and Germany?  
Correct: Paris and Berlin.  
Wrong: Paris is the capital of France and Munich is the capital of Germany.  
Output: ANNOTATED\_ANSWER: Paris is the capital of France and [ERROR\_START]Munich[ERROR\_END] is the capital of Germany.

Now process the input and output ONLY the ANNOTATED\_ANSWER with error markers.

Figure 13: Prompt for critical-token annotation, using Qwen3-Max as the LLM annotator.

903 For MMLU-Pro, we evaluate on 100 examples. For  
904 LiveCodeBench, we partition the dataset into five  
905 folds, use four folds to train the AutoJudge classi-  
906 fier, and reserve the remaining fold as the shared  
907 test set for all methods. All experiments are run  
908 on NVIDIA A6000, L40, and V100 GPUs. For

fair and consistent throughput reporting, all vLLM  
speed measurements are conducted on 8×V100.

### A.3 Potential of LLM Annotation

Given the limitations of labor-intensive manual an-  
notation and the high computational overhead of

909

910

911

912

913

### GSM8K Example (Question ID: 5)

**[Question]**

James writes a 3-page letter to 2 different friends twice a week. How many pages does he write a year?

**[Correct Answer]**

To find the number of pages James writes a year, we need to calculate the number of pages he writes per week and then multiply that by the number of weeks in a year.

James writes a 3-page letter to 2 different friends twice a week. So, he writes  $2 \times 2 = 4$  letters per week.

Since there are 3 pages in each letter, James writes  $4 \times 3 = 12$  pages per week.

There are 52 weeks in a year. So, James writes  $12 \times 52 = 624$  pages per year.

The final answer is 624.

**[Annotated Wrong Answer]**

To find out how many pages James writes a year, we need to calculate the number of pages he writes per week and then multiply it by the number of weeks in a year.

James writes a 3-page letter to 2 different friends twice a week. This means he writes **[ERROR\_START]** $2 \times 3 = 6$  pages per week. **[ERROR\_END]**

Since there are 52 weeks in a year, James writes  $6 \times 52 = 312$  pages per year.

The final answer is 312.

Figure 14: Example of Qwen3-Max annotation on the GSM8K (Question ID: 5), with **[ERROR\_START]** and **[ERROR\_END]** marking critical tokens in the draft model’s erroneous solution.

914 heuristic mining, a natural alternative is to lever- 936  
915 age an LLM as a surrogate annotator to provide 937  
916 supervision for judge training. In this appendix, we 938  
917 explore the feasibility of using Qwen3-Max<sup>1</sup> as an 939  
918 automatic labeler to identify logic-pivoting tokens 940  
919 in erroneous generations. We construct an anno- 941  
920 tation set from the GSM8K training split by first 942  
921 filtering for instances where the target model pro- 943  
922 duces a correct final answer while the draft model 944  
923 produces an incorrect one. From this subset, we 945  
924 then sample 2,000 examples. For each problem, we 946  
925 provide Qwen3-Max with the problem statement, 947  
926 the draft model’s erroneous output, and the target 948  
927 model’s verified correct answer, prompting it to 949  
928 highlight the specific words or tokens in the draft 950  
929 output that lead to the incorrect final answer (i.e., 951  
930 error-triggering or logic-pivoting tokens). The de- 952  
931 tailed prompt is shown in Figure 13. We then use 953  
932 these token-level annotations to train a lightweight 954  
933 linear classifier following the same training method- 955  
934 ology as AutoJudge. In total, the annotation cost is 956  
935 approximately 2M tokens (about \$10). 957

The results in Figure 12 indicate that the judge 936  
trained on Qwen3-Max-labeled data consistently 937  
underperforms the judge trained with AutoJudge- 938  
mined labels in the low-performance-loss regime 939  
(upper-left region of the curve). In other words, 940  
although LLM annotation is substantially cheaper 941  
than counterfactual rollout-based mining, its su- 942  
pervision does not yield a more accurate token 943  
accept/reject rule when decoding requires high pre- 944  
cision (i.e., minimal quality degradation). This gap 945  
suggests a key limitation of LLM-based annota- 946  
tion in our setting. This issue is also mentioned 947  
in Bachmann et al. (2025): using LLMs for this 948  
purpose proved to be too imprecise. Qwen3-Max 949  
may highlight tokens that look suspicious from a 950  
general semantic perspective, but these tokens are 951  
not always the ones that flip the final outcome of 952  
the generation. As a result, the labels can be in- 953  
formative but less well aligned with what judge 954  
decoding needs, especially in the high-precision 955  
regime where a small amount of label noise can no- 956  
ticeably worsen the quality–speed trade-off. Please 957  
refer to Figures 14-15 for annotation examples. 958

<sup>1</sup><https://qwen.ai/blog?id=qwen3-max>

### GSM8K Example (Question ID: 68)

#### [Question]

Jesse and Mia are competing in a week long race. They have one week to run 30 miles. On the first three days Jesse averages  $(2/3)$  of a mile. On day four she runs 10 miles. Mia averages 3 miles a day over the first 4 days. What is the average of their average that they have to run over the final three days?

#### [Correct Answer]

To find the average of their average that they have to run over the final three days, we need to first find their average for the entire race and then subtract the average they have to run over the first four days.

Jesse's average for the first three days is  $(2/3)$  miles per day. So, over the first three days, she runs a total of  $(2/3) * 3 = 2$  miles.

On day four, Jesse runs 10 miles. So, her total distance for the first four days is  $2 + 10 = 12$  miles.

Mia's average for the first four days is 3 miles per day. So, over the first four days, she runs a total of  $3 * 4 = 12$  miles.

The total distance they have to run over the entire week is 30 miles. So, the average distance they have to run over the final three days is  $30 - 12 = 18$  miles.

The final answer is 6.

#### [Annotated Wrong Answer]

To find the average distance Jesse and Mia need to run over the final three days, we first need to calculate their total distances for the first four days.

Jesse's total distance for the first four days:

- Day 1:  $(2/3)$  miles
- Day 2:  $(2/3)$  miles
- Day 3:  $(2/3)$  miles
- Day 4: 10 miles

Total distance for the first four days =  $(2/3) + (2/3) + (2/3) + 10 =$  **[ERROR\_START]**8 + 10 = 18 miles**[ERROR\_END]**

Mia's total distance for the first four days:

- Day 1: 3 miles
- Day 2: 3 miles
- Day 3: 3 miles
- Day 4: **[ERROR\_START]**10 miles**[ERROR\_END]**

Total distance for the first four days =  $3 + 3 + 3 + 10 =$  **[ERROR\_START]**19 miles**[ERROR\_END]**

Now, we need to find the average distance each of them needs to run over the final 3 days.

Jesse's average distance for the final three days:

Total distance for the first four days = **[ERROR\_START]**18 miles**[ERROR\_END]**

Number of days = 3

Mean distance = Total distance / Number of days = **[ERROR\_START]**18 / 3 = 6 miles**[ERROR\_END]**

Mia's average distance for the final three days:

Total distance for the first four days = **[ERROR\_START]**19 miles**[ERROR\_END]**

Number of days = 3

Average distance = Total distance / Number of days = **[ERROR\_START]**19 / 3 = 6.33 miles**[ERROR\_END]**

The final answer is:  $\boxed{6.33}$

Figure 15: Example of Qwen3-Max annotation on the GSM8K (Question ID: 68), with **[ERROR\_START]** and **[ERROR\_END]** marking critical tokens in the draft model's erroneous solution.