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 distribution. 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.

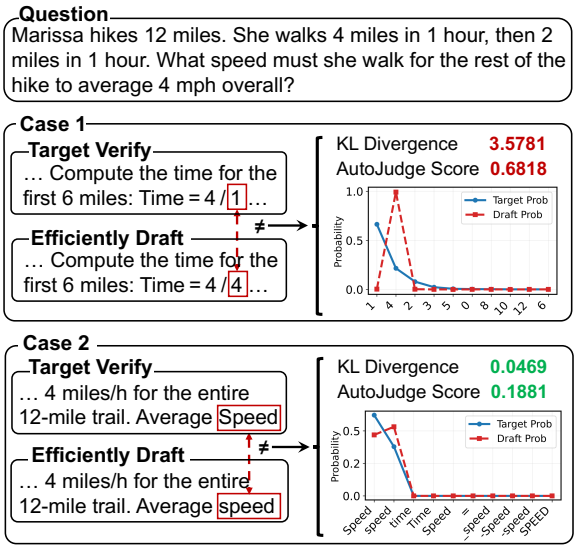

Figure 1: Relationship between AutoJudge score and token-level KL divergence. Higher AutoJudge scores indicate more critical tokens, which coincide with larger KL divergence and stronger target–draft disagreement (Case 1: inconsistent key numbers). Conversely, low scores correspond to small KL divergence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

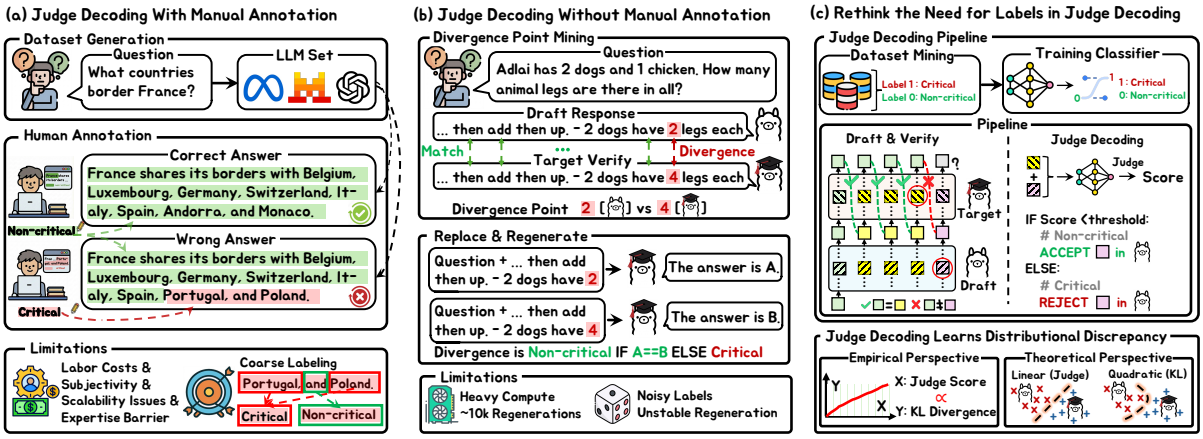

Figure 2: Judge decoding as learning distributional discrepancy. (a) Manual-annotation pipeline. (b) Divergence-point mining without manual labels; (c) We show that a linear judge's score is empirically correlated with and theoretically connected to distributional divergence (e.g., KL divergence).

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

3 The Bottlenecks of Supervision in Judge Decoding

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.

3.1 Judge Decoding With Manual Annotation

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:

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

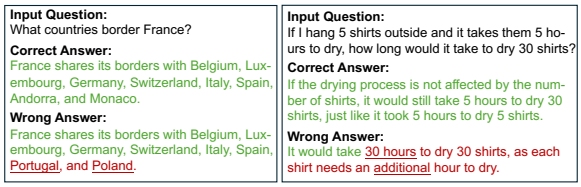

Figure 3: Illustration of overextended labeling boundaries in manual annotation, adapted from Bachmann et al. (2025). Key: Non-critical Tokens, Critical Tokens, Logic-pivoting Tokens. The span-based labeling (red) obscures the true logic-pivoting tokens (red underline), creating noisy supervision signals.

3.2 Judge Decoding Without Manual Annotation

To mitigate human dependency, Garipov et al. (2025) proposed AutoJudge, which mines critical tokens by checking if a token substitution alters the final answer. While this approach automates data collection, it introduces new computational and statistical challenges:

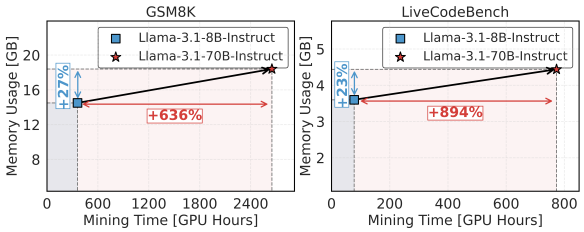

Figure 4: Efficiency analysis of AutoJudge dataset mining. The substantial growth in GPU hours and memory footprint for larger models highlights a significant scalability bottleneck.

1. **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.
2. **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*.

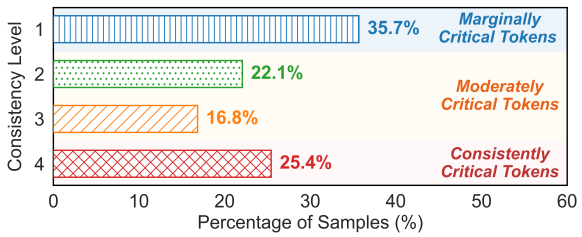

Figure 5: Consistency analysis of critical tokens. The low percentage of consistently critical tokens (Level 4) indicates that heuristic mining is heavily influenced by generation randomness.

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 alternative, 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:

1. **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:

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

   where higher entropy indicates a more dispersed probability distribution over the vocabulary $\mathcal{V}$.
2. **Disagreement (KL Divergence)** captures the structural deviation between the draft and target reasoning paths via:

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

   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.

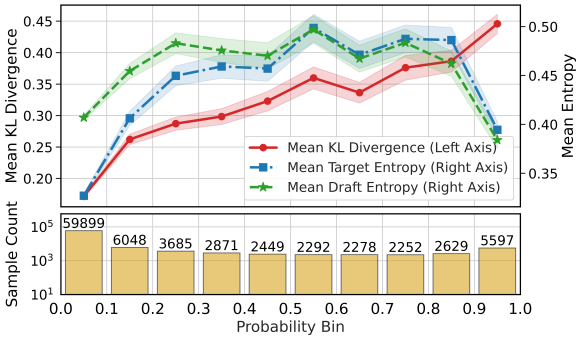

Figure 6: Correlation between supervised criticality scores and intrinsic model statistics on GSM8K. Samples are stratified into 10 bins based on AutoJudge scores (x-axis). **Top:** The mean KL divergence (red line) shows a clear upward trend as criticality increases, whereas model entropy (dashed lines) 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 predictive uncertainty, indicating that the models' existing probabilistic behavior already conveys informative cues for identifying critical tokens without additional supervision. Thus, KL divergence serves as a training-free proxy for locating logic pivots. To further enhance robustness, we employ a lightweight confidence mask: when the target model's top-1 probability exceeds 0.9, the system defaults to standard speculative sampling, reserving KL-based relaxation for tokens with genuine uncertainty. We retain this fallback as a principled, zero-cost safeguard; a detailed analysis is deferred to Appendix A.6.

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 $P_t = \mathrm{softmax}(z_t)$ and $P_d = \mathrm{softmax}(z_d)$. 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. □

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

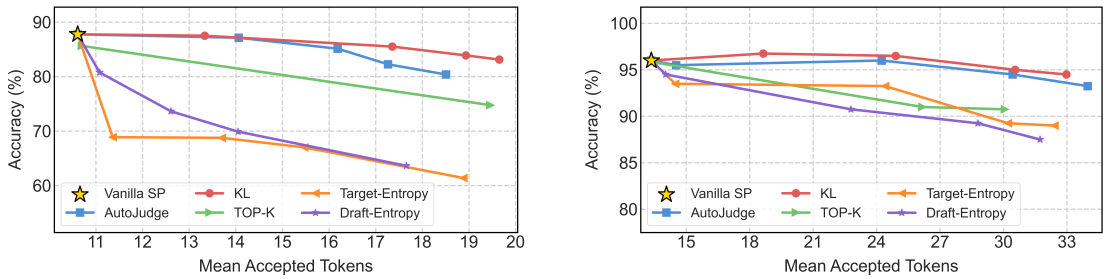

Figure 7: Accuracy and MAT on GSM8K for Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct (left) and Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct (right).

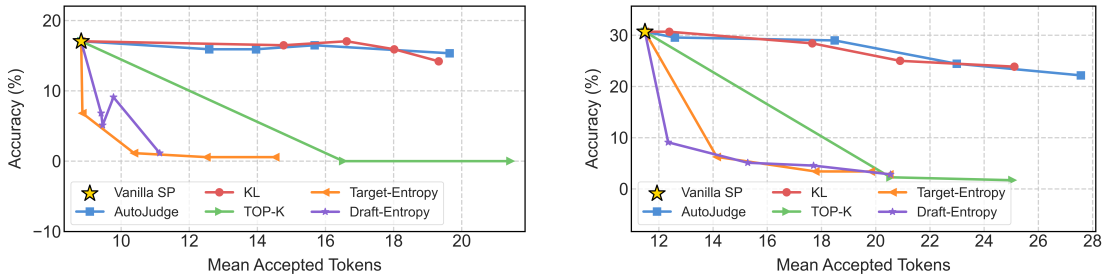

Figure 8: Accuracy and MAT on LiveCodeBench for Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct (left) and Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct (right).

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

Baselines and Metrics. We compare our method against: (i) Vanilla SP (Chen et al., 2023) (standard speculative decoding); (ii) Top-K (Bachmann et al., 2025), which relaxes verification based on token ranking; (iii) AutoJudge (Garipov et al., 2025), a training-based classifier approach; and two entropy-based methods, (iv) Target-Entropy and (v) Draft-Entropy (Wang et al., 2025). Evaluation metrics include Mean Accepted Tokens (MAT) for efficiency, benchmark-specific scores for quality, and end-to-end wall-clock speedup relative to Vanilla SP implemented in vLLM (Kwon et al., 2023). Further details are in Appendix A.2.

5.2 Main Results

Superior Efficiency-Accuracy Trade-off. We first compare methods on GSM8K and LiveCodeBench (Figures 7-8). The training-free KL thresholding exhibits highly consistent trends with the training-based AutoJudge, often outperforming it slightly. Specifically, on the Llama-1B/8B pair for GSM8K, KL thresholding improves MAT by **63.5%** (10.61→17.35) over Vanilla SP with only a ∼2% accuracy drop. Similarly, on the 8B/70B pair, MAT increases by **128.8%** (13.35→30.55) with a negligible ∼1% accuracy decrease. In contrast, baselines like Top-K and entropy-based strategies suffer from a much sharper accuracy degradation as MAT increases. A similar MAT–accuracy trade-off is also observed on LiveCodeBench, where code generation requires strict precision. KL thresholding improves MAT by **104.0%** (1B/8B, 8.83→18.01) and **53.7%** (8B/70B, 11.49→17.66) with only minimal performance loss. Its performance is comparable to AutoJudge, and is even slightly better in some settings. In comparison, Top-K and entropy-based strategies often fail to preserve code correctness, frequently resulting in non-executable programs. The consistently similar trends between KL thresholding and AutoJudge confirm that KL divergence serves as an effective proxy for learned verification, aligning with our motivation (Section 4) to replace trained judges with model-intrinsic signals.

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. Specif-

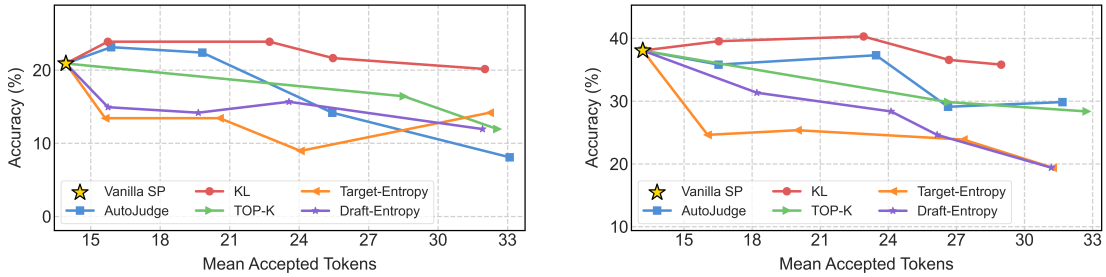

Figure 9: Accuracy and MAT on MATH-500-Hard for Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct (left) and Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct (right).

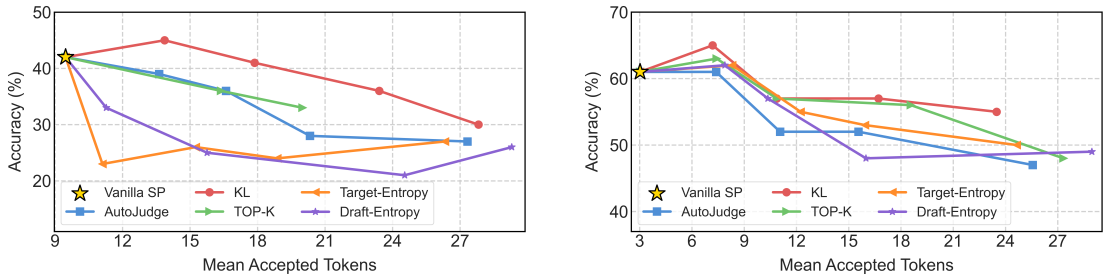

Figure 10: Accuracy and MAT on MMLU-Pro for Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct (left) and Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct (right).

ically, 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 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× speedup** over the Vanilla SP baseline while simultaneously keeping accuracy within a relatively small 1%–2% margin. This reinforces that costly critical-token mining is unnecessary for practical inference acceleration.

Efficiency in Long-Chain Reasoning. AutoJudge's reliance on critical-token mining is particularly prohibitive for reasoning models, whose long trajectories exhibit frequent draft-target divergence. We empirically evaluate KL thresholding on the Qwen3-0.6B/Qwen3-8B pair (Figure 11).

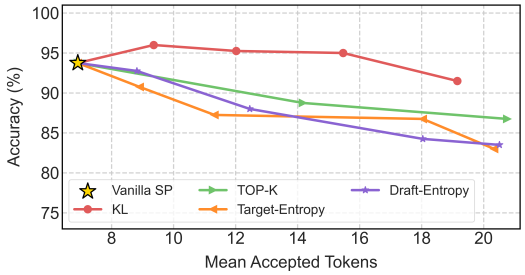

Figure 11: Accuracy and MAT on GSM8K for Qwen3-0.6B/Qwen3-8B.

Remarkably, KL thresholding achieves a **2.77× increase in MAT** (6.91→19.15) relative to Vanilla SP with only a marginal ∼2% performance drop. This evidence indicates that the KL thresholding effectively handles the complex divergence patterns of reasoning models without the need for intricate and expensive supervision mining.

Top-K Vocabulary Truncation. A natural question is whether restricting KL computation to only the Top-K tokens can further improve efficiency. Perhaps counter-intuitively, Top-K truncation does not yield wall-clock speedups, as irregular memory access from selection and index-gathering offsets the theoretical complexity reduction (see Appendix A.5). More critically, Top-K truncation degrades acceptance quality in the low-threshold regime. As shown in Table 2, at threshold 0.3, Top-20 and Top-200 incur **1.50%** and **1.12%** accu-

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×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×	1.17×	1.23×	1.27×	1.40×
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×	1.18×	1.26×	1.33×	1.60×
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×	1.22×	1.28×	1.35×	1.45×
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×	1.24×	1.30×	1.37×	1.45×
Vanilla SP	Acc	93.00%				
	Speed	24.19				

Table 2: Output quality and MAT under full-vocab and Top-K KL thresholding. Evaluated on GSM8K with Llama-3.2-1B-Instruct / Llama-3.1-8B-Instruct.

Threshold	**Full-vocab KL**		**Top-20**		**Top-200**	
	Acc	MAT	Acc	MAT	ACC	MAT
Vanilla SP	87.75%	10.61	87.75%	10.61	87.75%	10.61
0.3	87.50%	13.33	86.00% (↓1.50%)	13.45	86.38% (↓1.12%)	13.35
0.6	85.50%	17.35	85.62%	17.40	86.05%	16.72
0.7	83.88%	18.94	84.00%	18.72	84.30%	18.90
0.8	83.12%	19.65	83.62%	19.93	83.49%	19.91

racy drops respectively relative to full-vocab KL, because a low threshold places the decision boundary in a high-density region of the KL distribution where truncation-induced estimation error flips a significant number of accept/reject decisions.

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) and the EAGLE series (Li et al., 2024b,a, 2025). 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 (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 draft and target tokens and checking for semantic divergence.

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.

Limitations

We acknowledge some limitations in our work. First, due to computational constraints, we have not yet evaluated our approach on extremely large-scale models such as Llama-3.1-405B-Instruct, which would be essential for comprehensively validating scalability and real-world applicability. Second, the KL divergence computation introduces additional computational overhead during inference. We have not implemented highly optimized custom Triton kernels to optimize memory access and enable operator fusion, which could significantly reduce this cost. As a result, our current implementation may bottleneck the potential speedups achievable in modern high-performance inference frameworks like vLLM.

Ethical Considerations

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

References

Amos Azaria and Tom Mitchell. 2023. The internal state of an LLM knows when it's lying. In *Proc. Conf. Empirical Methods in Natural Language Processing*.

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. 2025. Judge Decoding: Faster speculative sampling requires going beyond model alignment. In *Proc. Int. Conf. Learning Representations*.

Anna Bavaresco, Raffaella Bernardi, Leonardo Bertolazzi, Desmond Elliott, Raquel Fernández, Albert Gatt, Esam Ghaleb, Mario Giulianelli, Michael Hanna, Alexander Koller, Andre Martins, Philipp Mondorf, Vera Neplenbroek, Sandro Pezzelle, Barbara Plank, David Schlangen, Alessandro Suglia, Aditya K Surikuchi, Ece Takmaz, and Alberto Testoni. 2025. LLMs instead of human judges? a large scale empirical study across 20 nlp evaluation tasks. In *ACL*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *arXiv:2401.10774*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. In *arXiv:2302.01318*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. In *arXiv:2110.14168*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning. In *arXiv:2501.12948*.

Tianyu Fu, Yi Ge, Yichen You, Enshu Liu, Zhihang Yuan, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. 2025. R2R: efficiently navigating divergent reasoning paths with small-large model token routing. In *arXiv:2505.21600*.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of LLM inference using lookahead decoding. In *arXiv:2402.02057*.

Roman Garipov, Fedor Velikonivtsev, Ivan Ermakov, Ruslan Svirschevski, Vage Egiazarian, and Max Ryabinin. 2025. Autojudge: Judge decoding without manual annotation. In *Advances in Neural Information Processing Systems*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. 2024. The llama 3 herd of models. In *arXiv:2407.21783*.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2024. A survey on LLM-as-a-Judge. In *arXiv:2411.15594*.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: retrieval-based speculative decoding. In *NAACL*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Advances in Neural Information Processing Systems*.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. 2025. SAM Decoding: Speculative decoding via suffix automaton. In *ACL*.

Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. 2021. Data movement is all you need: A case study on optimizing transformers. In *Proceedings of Machine Learning and Systems*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *Proc. Int. Conf. Learning Representations*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proc. Int. Conf. Machine Learning*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proc. Conf. Empirical Methods in Natural Language Processing*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. Eagle: Speculative sampling requires rethinking feature uncertainty. In *Proc. Int. Conf. Machine Learning*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. Eagle-3: Scaling up inference acceleration of large language models via training-time test. In *arXiv:2503.01840*.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. 2025. Turning trash into treasure: Accelerating inference of large language models with token recycling. In *ACL*.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2025. A comprehensive overview of large language models. In *ACM Transactions on Intelligent Systems and Technology*, volume 16, pages 1–72. Association for Computing Machinery.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. 2024. SuffixDecoding: Extreme speculative decoding for emerging ai applications. In *arXiv:2411.04975*.

OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, et al. 2024. Openai o1 system card. In *arXiv:2412.16720*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems*.

Giovanni Servedio, Alessandro De Bellis, Dario Di Palma, Vito Walter Anelli, and Tommaso Di Noia. 2025. Are the hidden states hiding something? Testing the limits of factuality-encoding capabilities in LLMs. In *ACL*.

Shengyin Sun, Yiming Li, Xing Li, Yingzhao Lian, Weizhe Lin, Hui-Ling Zhen, Zhiyuan Yang, Chen Chen, Xianzhi Yu, Mingxuan Yuan, and Chen Ma. 2025. Scaling up, speeding up: A benchmark of speculative decoding for efficient LLM test-time scaling. In *arXiv:2509.04474*.

Jikai Wang, Zhenxu Tian, Juntao Li, Qingrong Xia, Xinyu Duan, Zhefeng Wang, Baoxing Huai, and Min Zhang. 2025. Alignment-augmented speculative decoding with alignment sampling and conditional verification. In *Proc. Conf. Empirical Methods in Natural Language Processing*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *ACL*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. 2025. Qwen3 technical report. In *arXiv:2505.09388*.

Kanghoon Yoon, Minsub Kim, Sungjae Lee, Joonhyung Lee, Sunghyeon Woo, Yeonjun In, Se Jung Kwon, Chanyoung Park, and Dongsoo Lee. 2025. Selfjudge: Faster speculative decoding via self-supervised judge verification. In *arXiv:2510.02329*.

Yao Zhao, Zhitian Xie, Chen Liang, Chenyi Zhuang, and Jinjie Gu. 2024. Lookahead: An inference acceleration framework for large language model with lossless generation accuracy. In *Proc. ACM Conf. Knowledge Discovery and Data Mining*.

Amir Ziashahabi, Yavuz Faruk Bakman, Duygu Nur Yaldiz, Mostafa El-Khamy, Sai Praneeth Karimireddy, and Salman Avestimehr. 2025. Reject only critical tokens: Pivot-aware speculative decoding. In *arXiv:2511.00351*.

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 $P_t := \mathrm{softmax}(z_t)$ and $P_d := \mathrm{softmax}(z_d)$. 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, δ 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 $i, j \in \{1, \ldots, V\}$, 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. □

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 $\nabla^2 A(z) = \mathrm{diag}(p) - pp^\top$ for $p = \mathrm{softmax}(z)$.

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

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

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

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

Lemma A.2 (Second-order expansion around z_d)**.** *Let $P_d = \mathrm{softmax}(z_d)$, $P_t = \mathrm{softmax}(z_t)$ 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 $D_{\mathrm{KL}}(P_t\|P_d) = \frac{1}{2}\,\delta^\top F(P_d)\,\delta + O(\|\delta\|^3)$, where $F(P_d) = \nabla^2 A(z_d) = \mathrm{diag}(P_d) - P_d P_d^\top$.*

Proof. Applying Lemma A.1 with $p = P_t$ and $q = P_d$, and noting $z_d - z_t = -\delta$, we have the identity $D_{\mathrm{KL}}(P_t\|P_d) = A(z_d) - A(z_t) + P_t^\top \delta$. 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 $D_{\mathrm{KL}}(P_t\|P_d) = \frac{1}{2}\delta^\top F(P_d)\delta + O(\|\delta\|^3)$. □

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*

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

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 $F(p) = \mathrm{diag}(p) - pp^\top$. Then for any $\delta \in \mathbb{R}^V$,*

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

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

$$\begin{aligned} \delta^\top F(p)\,\delta &= \delta^\top \mathrm{diag}(p)\,\delta - \delta^\top (pp^\top)\delta \\ &= \sum_{i=1}^{V} p_i \delta_i^2 - \left(p^\top \delta\right)^2 \qquad (7) \\ &= \sum_{i=1}^{V} p_i \delta_i^2 - \Big(\sum_{i=1}^{V} p_i \delta_i\Big)^2. \qquad (8) \end{aligned}$$

Expand the right-side pairwise sum, we have

$$\sum_{i=1}^{V}\sum_{j=1}^{V} p_i p_j (\delta_i - \delta_j)^2 = \sum_{i,j} p_i p_j (\delta_i^2 + \delta_j^2 - 2\delta_i\delta_j)$$
$$= \sum_{i,j} p_i p_j \delta_i^2 + \sum_{i,j} p_i p_j \delta_j^2 - 2\sum_{i,j} p_i p_j \delta_i \delta_j$$
$$= 2\sum_{i} p_i \delta_i^2 - 2\Big(\sum_{i} p_i \delta_i\Big)^2, \tag{9}$$

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

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

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

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

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

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

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, $D_{\mathrm{KL}}(P_t \| P_d) \gtrsim \alpha_{ij}\big(\gamma_t^{(k)} + \gamma_d^{(k)}\big)^2$, 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 $D_{\mathrm{KL}}(P_t \| P_d) \approx \sum_{u,v} \alpha_{uv} \Delta_{uv}(x)^2 \geq \alpha_{ij} \Delta_{ij}(x)^2 \geq \alpha_{ij}(\gamma_t^{(k)} + \gamma_d^{(k)})^2$. □

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 $i, j \in \{1, \ldots, V\}$ 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 $a_{ij} := \left[W_t^\top (e_i - e_j);\ -W_d^\top (e_i - e_j) \right]$ and $\kappa_{ij} := (e_i - e_j)^\top (b_t - b_d)$. Using $x = [h_t; h_d]$ yields $\Delta_{ij}(x) = a_{ij}^\top x + \kappa_{ij}$. □

Proposition A.9 (Representation of the classifier in the primitive basis)**.** *Let* $\mathrm{Cls}(x) = \sigma(w^\top x + b)$ *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* $\{ a_{ij} \mid i, j = 1, \ldots, V \}$ *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* $\mathrm{Cls}(x) = \sigma\Big(\sum_{i=1}^{V} \sum_{j=1}^{V} \beta_{ij}\, \Delta_{ij}(x) + b' \Big)$*, 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 $\{ a_{ij} \mid i, j = 1, \ldots, V \}$ 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

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

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 $\mathrm{Cls}(x) = \sigma\Big(\sum_{i=1}^{V} \sum_{j=1}^{V} \beta_{ij}\, \Delta_{ij}(x) + b' \Big)$. □

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

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

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.

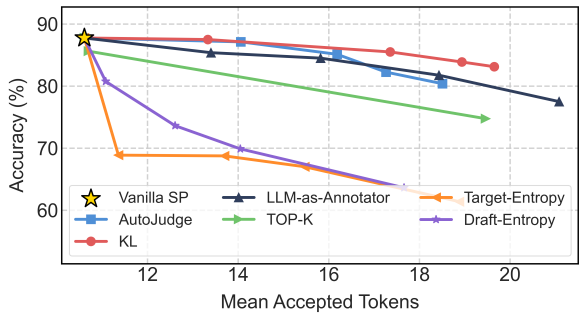

Figure 12: Accuracy and MAT on GSM8K for Llama-3.2-1B-Instruct and Llama-3.1-8B-Instruct (with Qwen3-Max as the annotator).

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.

For MMLU-Pro, we evaluate on 100 examples. For LiveCodeBench, we partition the dataset into five folds, use four folds to train the AutoJudge classifier, and reserve the remaining fold as the shared test set for all methods. All experiments are run 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 annotation and the high computational overhead of

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 x 2 = 4
    letters per week.

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

    There are 52 weeks in a year. So, James writes 12 x 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 [ER
    ROR_START]2 * 3 = 6 pages per week.[ERROR_END]

    Since there are 52 weeks in a year, James writes 6 * 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.

heuristic mining, a natural alternative is to leverage an LLM as a surrogate annotator to provide supervision for judge training. In this appendix, we explore the feasibility of using Qwen3-Max[1] as an automatic labeler to identify logic-pivoting tokens in erroneous generations. We construct an annotation set from the GSM8K training split by first filtering for instances where the target model produces a correct final answer while the draft model produces an incorrect one. From this subset, we then sample 2,000 examples. For each problem, we provide Qwen3-Max with the problem statement, the draft model's erroneous output, and the target model's verified correct answer, prompting it to highlight the specific words or tokens in the draft output that lead to the incorrect final answer (i.e., error-triggering or logic-pivoting tokens). The detailed prompt is shown in Figure 13. We then use these token-level annotations to train a lightweight linear classifier following the same training methodology as AutoJudge. In total, the annotation cost is approximately 2M tokens (about $10).

[1] https://qwen.ai/blog?id=qwen3-max

The results in Figure 12 indicate that the judge trained on Qwen3-Max-labeled data consistently underperforms the judge trained with AutoJudge-mined labels in the low-performance-loss regime (upper-left region of the curve). In other words, although LLM annotation is substantially cheaper than counterfactual rollout-based mining, its supervision does not yield a more accurate token accept/reject rule when decoding requires high precision (i.e., minimal quality degradation). This gap suggests a key limitation of LLM-based annotation in our setting. This issue is also mentioned in Bachmann et al. (2025): using LLMs for this purpose proved to be too imprecise. Qwen3-Max may highlight tokens that look suspicious from a general semantic perspective, but these tokens are not always the ones that flip the final outcome of the generation. As a result, the labels can be informative but less well aligned with what judge decoding needs, especially in the high-precision regime where a small amount of label noise can noticeably worsen the quality–speed trade-off. Please refer to Figures 14-15 for annotation examples.

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 + 1 0 = 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.

A.4 Detailed Decoding Algorithm

As discussed in Section 3, both supervision paradigms for judge decoding bottleneck on the cost and reliability of token-level labels. Our key observation is that the KL divergence $D_{\mathrm{KL}}(P_T \| P_D)$ already captures the semantic discrepancy a trained judge attempts to learn, yielding an annotation-free acceptance criterion with no rollouts and no additional parameters. The resulting procedure is detailed in Algorithm 1.

Algorithm 1 Annotation-Free Judge Decoding

Require: Prompt $\mathbf{x}$, target model M_T, draft model M_D, draft length γ, KL threshold τ, confidence threshold τ_c

Ensure: Generated sequence $\mathbf{x}$

repeat
  `// Stage 1: Draft`
  M_D drafts γ tokens $d_1, \ldots, d_\gamma$, storing draft logits $\mathbf{z}_D^{(i)}$ at each position i.
  `// Stage 2: Verify`
  M_T verifies $d_1, \ldots, d_\gamma$ in one forward pass, yielding logits $\mathbf{z}_T^{(i)}$ and greedy predictions $t_i = \arg\max \mathbf{z}_T^{(i)}$ for $i = 1, \ldots, \gamma$.
  `// Stage 3: Accept / Reject`
  for $i = 1$ **to** γ **do**
    $P_T \leftarrow \mathrm{softmax}\left(\mathbf{z}_T^{(i)}\right)$
    $P_D \leftarrow \mathrm{softmax}\left(\mathbf{z}_D^{(i)}\right)$
    $\mathrm{KL}_i \leftarrow D_{\mathrm{KL}}(P_T \| P_D)$
    if $\max(P_T) > \tau_c$ **then**
      $\mathit{accept}_i \leftarrow (d_i = t_i)$
    else
      $\mathit{accept}_i \leftarrow (d_i = t_i) \ \vee \ (\mathrm{KL}_i < \tau)$
    end if
    if not accept_i **then**
      Replace d_i with t_i; truncate after position i; **break**
    end if
  end for
  `// Stage 4: Bonus token`
  if all γ tokens accepted **then**
    Append one additional token from M_T
  end if
until EOS emitted **or** max length reached
return $\mathbf{x}$

A.5 Analysis of Top-K Vocabulary Truncation

Computational Comparison. A Top-K variant reduces the element-wise KL computation from $O(V)$ to $O(K)$, which is theoretically appealing. However, it introduces two additional operations that full-vocab KL does not require: (1) a Top-K selection step involving branch-intensive, irregular memory access, and (2) a gather-by-index step that breaks coalesced memory patterns. Table 3 summarizes the operational differences.

The efficiency advantage of Top-K lies solely in Step 3, but realizing this advantage in practice requires fusing Steps 1–3 into a single kernel to mitigate the irregular memory access introduced by Steps 1 and 2. With standard PyTorch implementations, these irregular operations negate the theoretical benefit. We profile the wall-clock latency on Llama-3.1-8B-Instruct ($V = 128{,}256$) for various K values in Figure 16. Notably, latency remains nearly constant across all K values (0.125–0.130 ms), confirming that the bottleneck lies in the Top-K selection and gather operations rather than the KL summation itself. In contrast, full-vocab KL consists entirely of streaming element-wise operations that are naturally fused-kernel-friendly, achieving 1.8× lower latency than the fastest Top-K variant without any custom kernel optimization.

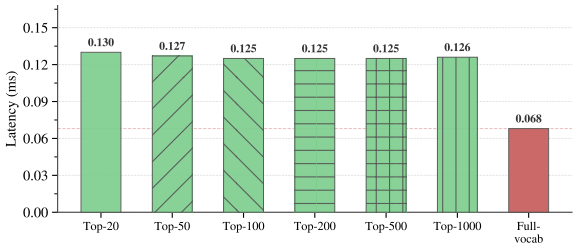

Figure 16: Wall-clock latency (ms) of KL computation under different Top-K configurations. Full-vocab KL is 1.8× faster than the fastest Top-K variant due to its fully streaming memory access pattern.

Theoretical Compatibility We verify that the structural correspondence established in Theorem 4.1 is exactly preserved under Top-K truncation. The core primitive $\Delta_{ij}(x) = (x_i - x_j)$ is defined on logit *differences*, hence invariant to vocabulary restriction and renormalization. Concretely, let $\tilde{P}_t$ and $\tilde{P}_d$ denote the renormalized target and draft distributions over a Top-K subset S_K. Since Lemma A.4 is an algebraic identity valid for any probability simplex on any finite index set, Applying Corollary A.5 to $\tilde{P}_t$ and $\tilde{P}_d$ on S_K yields $D_{\mathrm{KL}}(\tilde{P}_t \| \tilde{P}_d) \approx \sum_{i,j \in S_K} \tilde{\alpha}_{ij} \, \Delta_{ij}(x)^2$, with $\tilde{\alpha}_{ij} = \frac{1}{4} \tilde{P}_d(i) \, \tilde{P}_d(j)$. Combined with Proposition A.9, the core claim (*quadratic aggregation*

Table 3: Operational comparison between full-vocab KL and Top-K KL. Full-vocab KL avoids the irregular memory access introduced by Steps 1–2 and consists entirely of streaming, fused-kernel-friendly operations.

Step	**Full-vocab KL**	**Top-K KL**
1: Top-K selection	Not needed	$O(V)$, branch-intensive, irregular access
2: Gather by index	Not needed	$O(K)$, indirect indexing, breaks coalescing
3: Log-softmax + KL	$O(V)$, streaming, fusible	$O(K)$, streaming, fusible

for KL vs. linear aggregation for the classifier, both over the same primitives) holds exactly. Truncation merely narrows the summation range from V^2 to $|S_K|^2$ without altering the algebraic structure of the correspondence.

A.6 Analysis of the Top-1 Confidence Fallback

We provide a detailed analysis of the Top-1 confidence fallback mechanism, examining both its activation frequency and its interaction with supervised verification methods. All experiments are conducted on GSM8K using the Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct pair.

Activation Frequency. Table 4 reports the trigger statistics of the Top-1 confidence fallback across different KL thresholds. Even at the most aggressive threshold (i.e., 0.8), the fallback activates on fewer than 5% of token-level verification decisions (fewer than 0.4 times per query on average), confirming that KL divergence serves as the dominant verification signal throughout decoding.

Table 4: Trigger frequency and performance of the Top-1 confidence fallback across KL thresholds. "Trigger (%)" denotes the percentage of token-level decisions where the fallback is activated; "Trigger / Query" denotes the average number of activations per query.

		Vanilla SP	0.3	0.6	0.7	0.8
w/ fallback	Acc (%)	87.75	87.50	85.50	83.88	83.12
	MAT	10.61	13.33	17.35	18.94	19.65
	Trigger (%)	N/A	0.0	1.1	2.7	4.9
	Trigger/Query	N/A	0	0.11	0.22	0.37
w/o	Acc (%)	87.75	87.60	86.62	85.25	84.38
	MAT	10.61	13.45	17.21	18.75	19.80

We highlight three observations from Table 4:

- **The fallback is triggered very infrequently.** Over 95% of all rejection events are governed by the KL-based criterion, indicating that the confidence mask operates in a narrow regime.
- **The fallback has negligible impact on overall performance.** Both configurations (*w/* and *w/o*) yield comparable accuracy and MAT across all thresholds. This is expected by design: when the target confidence is high, the draft model overwhelmingly agrees with the target on the Top-1 token, so KL divergence is naturally low and the draft would already be accepted. The fallback only intervenes when both conditions, high target confidence and draft-target disagreement, co-occur, which is inherently rare.
- **The fallback serves as a principled, zero-cost safeguard.** It adds no learnable parameters, introduces negligible computational overhead (a single scalar comparison), and provides a conservative fallback for potential edge cases. Its design philosophy is analogous to a "safety valve": rarely activated under normal operation, but available as an additional layer of robustness.

Interaction with Supervised Judges. To verify that the performance advantage of our method cannot be attributed to the Top-1 fallback alone, we further combine it with AutoJudge and report results in Table 5. As shown in Table 5, incorporating the Top-1 fallback into AutoJudge produces negligible differences across all thresholds. The fallback targets a rare regime where high target confidence and draft-target disagreement co-occur, so its effect is negligible regardless of the underlying verification mechanism. This confirms that the gains of our approach are attributable to the KL-based verification, which captures the full distributional divergence between draft and target in a training-free manner, rather than compressing it into a binary accept/reject decision.

Table 5: AutoJudge performance with and without Top-1 fallback under different KL thresholds.

Setting	**Metric**	**Vanilla SP**	**0.003**	**0.008**	**0.02**	**0.03**
w/o fallback	Acc	87.75%	87.12%	85.12%	82.25%	80.38%
	MAT	10.61	14.06	16.18	17.26	18.51
w/ fallback	Acc	87.75%	86.60%	84.12%	82.25%	80.75%
	MAT	10.61	14.05	15.99	17.87	18.63

A.7 Step-Level KL Divergence Extension

We investigate whether KL divergence can be applied at the *step level*, i.e., aggregating token-level KL values within each reasoning step and making accept/reject decisions per step. Specifically, we segment the draft window into reasoning steps delimited by newline characters, compute the mean KL divergence within each step, and reject all tokens in a step if its aggregated KL exceeds the threshold. We evaluate on GSM8K using Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct. Results are shown in Figure 17.

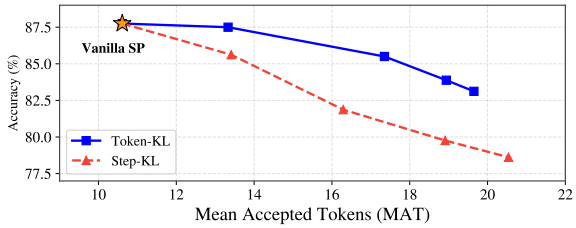

Figure 17: Accuracy–speedup trade-off of Token-KL vs. Step-KL on GSM8K (Llama-3.2-1B-Instruct / Llama-3.1-8B-Instruct).

At comparable MAT levels, Step-KL consistently underperforms Token-KL by approximately 2–5% in accuracy (e.g., 85.62% vs. 87.50% at MAT≈13; 78.62% vs. 83.12% at MAT≈20). We attribute this degradation to three factors: *(1) Signal dilution under aggregation*: the high KL signal from a few critical "logic-pivoting" tokens is diluted by non-critical tokens, causing entire steps containing errors to be incorrectly accepted; *(2) Collateral rejection of correct tokens*: when a step exceeds the threshold due to one or two high-KL tokens, *all* tokens in that step are rejected, wasting useful draft tokens and inflating re-drafting iterations; *(3) Variable and unreliable step granularity*: reasoning steps vary widely in length, making a single threshold difficult to calibrate robustly. These results confirm that token-level granularity is the appropriate design choice for our framework.

A.8 Quality-Constrained Speedups

We integrate our method into vLLM and evaluate quality-constrained speedups under various accuracy drop budgets δ. The speedup is measured relative to vanilla speculative decoding (Vanilla SP), which itself already achieves roughly $2\times$ acceleration over standard autoregressive (AR) generation in vLLM. The absolute speedup over AR decoding is therefore considerably larger. Note that our current implementation is a straightforward integration without system-level optimization, and we believe there remains room for further engineering-level improvement. Results for two draft-target model pairs are presented in Table 6 and Table 7, respectively. As shown, our method achieves up to $1.60\times$ speedup over vanilla speculative decoding with the Llama-3.2-1B / Llama-3.1-8B pair under a $\delta \leq 5\%$ accuracy drop budget, and up to $1.45\times$ with the Llama-3.1-8B / Llama-3.1-70B pair under $\delta \leq 3.0\%$.

Table 6: Quality-constrained speedups on vLLM with Llama-3.2-1B-Instruct/Llama-3.1-8B-Instruct.

δ (Acc Drop)	Acc (%)	Speed (tokens/s)	Speedup
Vanilla SP	82.56	29.45	1.00×
$\leq 0.1\%$	82.77	39.58	1.03×
$\leq 0.5\%$	82.00	41.91	1.09×
$\leq 1.5\%$	81.07	48.35	1.26×
$\leq 2.5\%$	79.46	50.96	1.33×
$\leq 5\%$	77.13	61.45	1.60×

Table 7: Quality-constrained speedups on vLLM with Llama-3.1-8B-Instruct/Llama-3.1-70B-Instruct.

δ (Acc Drop)	Acc (%)	Speed (tokens/s)	Speedup
Vanilla SP	93.00	24.19	1.00×
$\leq 0.5\%$	92.69	27.21	1.12×
$\leq 1.0\%$	92.02	31.77	1.31×
$\leq 1.5\%$	91.41	32.78	1.36×
$\leq 2.5\%$	90.36	34.29	1.42×
$\leq 3.0\%$	90.27	35.10	1.45×

A.9 Discussion on Consensus vs. Correctness

One may argue that when both the draft and target models share the same erroneous reasoning, token-level KL consistency cannot lead to correct results. We clarify this concern below.

- **Speculative decoding targets distributional fidelity, not correctness.** The objective of speculative decoding is to approximate the target model's output distribution, not to surpass it. Low KL divergence signals that the draft faithfully reproduces the target's behavior. Accepting consensus tokens is therefore the *intended behavior*.
- **Trained judges are equally unable to detect shared errors.** Supervision for critical token detection, whether mined automatically (as in AutoJudge (Garipov et al., 2025)) or annotated manually (Bachmann et al., 2025), is de-

rived exclusively from draft-target divergence points. When both models produce identical output, no divergence is detected and no token is flagged as critical. This limitation is inherent to the draft-then-verify paradigm itself.

- **Correcting the target model's own errors is an orthogonal direction.** Addressing such errors requires fundamentally different techniques (e.g., tool-augmented reasoning or reward models), which fall outside the scope of speculative decoding.