Reject Only *Critical* Tokens: Pivot-Aware Speculative Decoding

Anonymous ACL submission

Abstract

Speculative Decoding (SD) ensures that the output matches the target model's distribution exactly. However, we argue that this distribution matching requirement is too stringent and results in unnecessarily low acceptance rates - limiting potential speedups. Instead, we advocate a *reformulation* of the decoding objective: the proposed decoding strategy should match the *expected utility* i.e., the task-specific performance, of the target model. This perspective also aligns better with real-world use cases of LLMs, where utility (e.g., code correctness, factual accuracy) is often more important than sampling distribution. Based on this reformulation, we propose a novel decoding strategy: **Pivot-Aware Speculative Decoding**, which rejects only those tokens that would lead to a utility drop in the final output. We refer to these critical tokens as *Pivot tokens*. We propose a method for labeling tokens as pivotal or non-pivotal and train a lightweight classifier to detect them. This method can be viewed as a relaxed version of standard SD, which offers much higher acceptance while preserving utility. We evaluate our method across various datasets, demonstrating that we can achieve up to $2.5\times$ speedup with comparable utility.

1 Introduction

While LLMs demonstrate impressive performance in many domains (OpenAI, 2023), they also come with a major drawback: slow text generation. LLMs typically follow an auto-regressive structure, generating one token at a time in a sequential manner. This sequential nature significantly slows down generation, especially as model size increases. *Speculative Decoding (SD)* (Leviathan et al., 2023; Chen et al., 2023) addresses this bottleneck by leveraging a small, fast *draft model* alongside the large *target model*. Specifically, the draft model first generates a sequence of N tokens. Then, the target model verifies these N tokens in parallel by computing the acceptance probability as the ratio of the sampling probabilities of N tokens assigned by the target and draft models. This parallel verification allows the target model to skip sequential generation for accepted tokens, which leads to substantial speed-ups, which is roughly proportional to the fraction of tokens accepted. Importantly, acceptance based on sampling probability ratios allows SD to preserve the original sampling distribution of the target model, meaning that SD's output matches that of the target model.

Question: An agricultural cooperative must ship 6500 kg of potatoes. During transport by truck, 150 kg are damaged and therefore cannot be sold. The potatoes are distributed in 50 kg bags, each bag being sold for $72. What will the sale of the potatoes bring?

Model Output:
To find the sale of the potatoes:
1. Calculate the total weight of potatoes that can be sold:
 $ 6500 - 150 = 6350 $ kg.
2. Calculate the number of bags:
 $ 6350 ÷ 50 = 127 $ bags.
3. Calculate the total revenue from selling the potatoes:
 $ 127 × 72 = 9~~2~~144 $.

Final Answer: 9144

Figure 1: To match the target model's distribution, SD rejects many tokens that a draft model proposes (shown in blue). Most of these rejections are unnecessary, and fixing a single token (~~2~~ → 1) is enough to recover the correct answer.

Although SD guarantees an equivalent sampling distribution to the target model, its speed-up is often limited by the low acceptance rate of draft tokens imposed by this strict distribution-matching requirement. Recent works attempt to mitigate this low-acceptance rate limitation through various strategies, such as aligning the draft model's sampling distribution more closely with the target model (Zhou et al., 2024), or employing heuristics like rejecting only hallucinated, low-quality tokens (Bachmann et al., 2025).

In contrast to existing approaches, we advocate *reformulating SD* to prioritize what matters most

in the practical use of LLMs: their downstream performance, or in other words, their *utility*. In most real-world use cases of LLMs, the exact probability distribution is often irrelevant compared to ensuring that the model's outputs serve the end task effectively, which is to achieve high utility. Motivated by this insight, we modify the objective of SD such that, instead of requiring the proposed decoding to match the target model's sampling distribution, we aim for the proposed decoding to match the **expected utility** of the target model's outputs.

Following this reformulation, we propose **Pivot-Aware Speculative Decoding**, a decoding strategy that *rejects only the tokens whose generation would lead to a utility drop in the final output of the target model.* We name these critical tokens as **pivot** tokens as shown in Figure 1 and train a lightweight classifier to identify them effectively. This approach enables significant speed-ups without sacrificing utility, arguably the most important metric in many real-world tasks. Since utility is definable across a wide range of tasks, our method results in a generic and extensible decoding algorithm.

2 Methodology: PAD (Pivot-Aware Speculative Decoding)

Preliminaries. A language model is a probabilistic model $p(\mathbf{x})$ over token sequences $\mathbf{x} = (x_1, \ldots, x_k) \in \mathcal{V}^k$, typically factorized autoregressively: $p(\mathbf{x}) = \prod_{t=1}^{k} p(x_t \mid \mathbf{x}_{<t})$, where $\mathbf{x}_{<t}$ is the prefix up to $t-1$. Given a context $\mathbf{x}_c$, the model generates output $\mathbf{x}_o \sim p(\cdot \mid \mathbf{x}_c)$ sequentially, which leads to slow generation especially for larger models. Speculative Decoding (SD) accelerates sampling from a target model p_{target} using a smaller draft model p_{draft}. Starting from t-th token, the draft proposes a block of γ token, $\mathbf{x}_{t:t+\gamma-1} \sim p_{\text{draft}}(\cdot \mid \mathbf{x}_{<t})$, which are then evaluated in parallel by p_{target}. Each token x_{t+i}, where $0 \leq i < \gamma$, is accepted with probability $\min\left(1, \frac{p_{\text{target}}(x_{t+i}|\mathbf{x}_{<t+i})}{p_{\text{draft}}(x_{t+i}|\mathbf{x}_{<t+i})}\right)$. If all draft tokens are accepted, an additional token $x_{t+\gamma}$ is sampled from the target model; if x_{t+i} is rejected, a replacement token is resampled from $p'(x \mid \mathbf{x}_{<t+i}) = \text{norm}\big(\max\big(0,\, p_{\text{target}} - p_{\text{draft}}\big)\big)$. This ensures sampling distribution equals p_{target} (Leviathan et al., 2023; Chen et al., 2023). SD's speedup is approximated as $\frac{\eta\gamma+1}{c\gamma+1}$ where $\eta = \frac{\text{\# accepted draft tokens}}{\text{\# generated draft tokens}}$ is the draft acceptance ratio and c is cost coefficient computed as be the ratio between the time for a single run of draft model and target model.

Objective: Matching Target Utility. To improve draft token acceptance and generation speed, we propose matching the *utility* of the target model rather than its sampling distribution. Utility, defined as task performance given a context or query, can be continuous, but we adopt a binary definition for simplicity.

Definition 1 (Utility)**.** Let x denote a query or context, and y be the language model's output. Given an evaluation function $\text{Eval}(y, x)$ and a user-specified threshold τ, the utility function $u(y, x)$ is defined as: $u(y, x) = \begin{cases} 1, & \text{if Eval}(y, x) \geq \tau, \\ 0, & \text{otherwise.} \end{cases}$ i.e. $u(y, x) = 1$ if the model's output achieves the desired evaluation score; otherwise, $u(y, x) = 0$.

Given the definition of utility, we now aim to ensure that the proposed decoding strategy, denoted by $\hat{p}$, achieves an *expected utility* which closely matches that of the target model p_{target}. Let $U(p, x) = \mathbb{E}_{\mathbf{y}\sim p(\cdot|\mathbf{x})}\left[u(\mathbf{y}, \mathbf{x})\right]$ denote the expected utility of p for context x.

Definition 2 (ϵ-Utility preserving decoding)**.** Suppose we are given a task or set of tasks, with inputs $\mathbf{x}_c$ drawn from a dataset distribution $\mathcal{D}$. Then we say that $\hat{p}$ is ϵ-utility preserving if:

$$\mathbb{E}_{\mathbf{x}_c\sim\mathcal{D}}\left[U(\hat{p}, x_c)\right] \geq \mathbb{E}_{\mathbf{x}_c\sim\mathcal{D}}\left[U(p_{\text{target}}, x_c)\right] - \epsilon \tag{1}$$

where $\epsilon \geq 0$ is a user-specified small tolerance value.

It is trivial to verify that p_{target} is utility preserving for any ϵ. For this reason, utility preservation is a *relaxation* of the original SD objective (which enforces exact distributional equivalence).

Pivot-Aware Speculative Decoding. Our goal is to design a new decoding strategy p_{PAD} that is utility preserving as in Definition 2. To this end, we propose an approach based on rejecting only those tokens that *would lead to a utility drop in the final output of the target model, assuming the remainder of the generation is completed by the target model itself.* We refer to such "utility-changer" tokens as **pivot tokens**. Formally, we define a pivot token as follows:

Definition 3 (Pivot Token)**.** Suppose we are given a context $\mathbf{x}_c$, $\mathbf{y}_{<t}$ is the tokens generated before position t, and $\tilde{y}_t$ is a candidate token. $\tilde{y}_t$ is a pivot token at time t if

$$U(p_{\text{target}}, (\mathbf{x}_c, \mathbf{y}_{<t}, \tilde{y}_t)) \leq U(p_{\text{target}}, (\mathbf{x}_c, \mathbf{y}_{<t})) - \epsilon \tag{2}$$

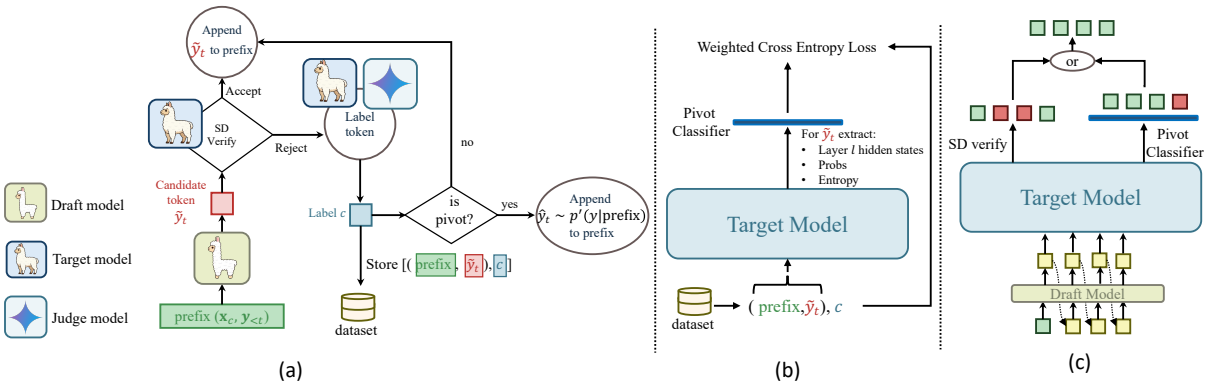

Figure 2: Pivot-Aware Speculative Decoding. (a) Dataset generation: label SD-rejected draft tokens via target-model rollouts with an LLM-as-judge check; (b) Training: fit a pivot classifier on target-side features (layer-ℓ hidden states, logits, entropy); (c) Inference: tokens are accepted if standard SD accepts them or the classifier predicts non-pivot.

That is, $\tilde{y}_t$ is a pivot token if conditioning on the tth token being $\tilde{y}_t$ results in a loss of expected utility when sampling completions (rollouts) from the target distribution. In other words, pivot tokens pivot the generation trajectory toward lower utility outputs. Note that whether a particular token is a pivot token depends on the query $\mathbf{x}_c$, the output so far $\mathbf{y}_{<t}$, the target distribution p_{target}, and also the utility function u.

Given a binary classifier $f_{\text{pivot}}(y_t, \mathbf{y}_{<t}, \mathbf{x}_c)$ indicating whether $\tilde{y}_t$ is a pivot token, our decoding p_{PAD} follows SD with a modified rule: if $f_{\text{pivot}}(\tilde{y}_t, \mathbf{y}_{<t}, \mathbf{x}) < \sigma$ directly accept $\tilde{y}_t$ otherwise follow standard SD. This modification relaxes SD's acceptance criteria by always accepting tokens deemed not pivotal ($f_{\text{pivot}}(\tilde{y}_t, \mathbf{y}_{<t}, \mathbf{x}) < \sigma$), leading to a higher token acceptance rate.

Lemma 1 (Rejecting only pivot tokens preserves utility)**.** If f_{pivot} has 100% recall on pivot tokens (i.e., it never labels a pivot as non-pivot), then p_{PAD} satisfies Definition 2 with $\epsilon = 0$.

The formal proof is provided in Appendix A.1. Note that a trivial 100% recall pivot-token classifier can be obtained by labeling all draft tokens as pivots, which reduces to standard SD with many rejections. Classifier quality dictates the trade-off between utility preservation and speed-up: better classifiers reject fewer non-pivots, increasing efficiency while maintaining utility.

2.1 Pivot Classifier: Data and Training

Direct labeling via Definition 3 is intractable because it requires expectations over continuations per token. We use a rollout-based approximation with safeguards to control noise and cost. We spend labeling budget only on tokens that *standard SD would reject*. For each $\mathbf{x} \sim \mathcal{D}$ and position t, draw $\tilde{y}_t \sim p_{\text{draft}}(\cdot \mid \mathbf{x}, \mathbf{y}_{<t})$. If token is accepted under SD, skip; otherwise, attempt to label $\tilde{y}_t$.

Rollout estimate with tolerance. For a candidate $\tilde{y}_t$, estimate $\widehat{U}(\mathbf{x}, \mathbf{y}_{<t})$ and $\widehat{U}(\mathbf{x}, \mathbf{y}_{<t}, \tilde{y}_t)$, each computed from N independent rollouts from p_{target} with binary utility u. To mitigate sampling noise due to false positives, we introduce a tolerance $\alpha \in [0, 1]$ and provisionally label pivot if $\widehat{U}(\mathbf{x}, \mathbf{y}_{<t}, \tilde{y}_t) < \alpha \widehat{U}(\mathbf{x}, \mathbf{y}_{<t})$, and non-pivot otherwise.

LLM-as-judge sanity check. Some harmful tokens can appear benign under u due to self-correction or brittle shortcuts, such as reward hacking, leading to false negative samples. Among rollouts contributing to $\widehat{U}(\mathbf{x}, \mathbf{y}_{<t}, \tilde{y}_t)$ with $u = 1$, select the *median-length* positive rollout and prompt an LLM-as-judge to assess reasoning soundness (no leaps/contradictions). If judged unsound, flip the provisional label to *pivot*. This single check is a cheap guard against mislabeled non-pivots.

Training. For each labeled example, we extract target-side features available at verification time (hidden state for a chosen layer l, candidate token probability and entropy) and train f_{pivot} as a binary classifier with a recall-focused objective. The runtime overhead is negligible for a small MLP forward pass only when SD would otherwise reject.

Inference. At inference, PAD runs standard SD. When SD would reject at position t, query f_{pivot}. If it predicts *non-pivot*, accept the draft token; if it predicts *pivot*, fall back to the target token as in SD. To further ensure robustness, we also reject any token whose target distribution probability

Setting	Metric	GSM8K	AIME24	MBPP
Target	Acc.	$94_{\pm 0.6}$	$73_{\pm 4.5}$	$70_{\pm 1.9}$
	Tok/s	71	71	71
SD	η (%)	$45.3_{\pm 0.2}$	$47.2_{\pm 0.7}$	$41.8_{\pm 0.4}$
	Speed-up	1.57	1.69	1.46
PAD ($\sigma = 0.7$)	Acc.	$93_{\pm 1.2}$	$57_{\pm 7.8}$	$64.7_{\pm 1.7}$
	η (%)	$77.2_{\pm 0.8}$	$78.8_{\pm 0.5}$	$69.1_{\pm 0.7}$
	Speed-up	2.46	2.51	2.25
PAD ($\sigma = 0.5$)	Acc.	$93.4_{\pm 0.9}$	$61.6_{\pm 5.3}$	$68.6_{\pm 2.3}$
	η (%)	$70.8_{\pm 0.9}$	$71.6_{\pm 0.6}$	$61.7_{\pm 0.2}$
	Speed-up	2.29	2.33	2.00
PAD ($\sigma = 0.3$)	Acc.	$93.7_{\pm 1.1}$	$69.6_{\pm 4.2}$	$68.3_{\pm 4.8}$
	η (%)	$58.2_{\pm 0.2}$	$58.3_{\pm 0.6}$	$50.2_{\pm 0.3}$
	Speed-up	1.95	1.95	1.71
Draft	Acc.	$74.2_{\pm 1.5}$	$12.5_{\pm 3.4}$	$51.1_{\pm 1.3}$
	Tok/s	280	280	280

Table 1: Evaluation results. Acc.: accuracy; Tok/s: tokens per second; SD: speculative decoding.

falls below 0.0001. Refer to Appendix B for more details.

3 Experiments

Datasets: We evaluate our approach on three tasks spanning different domains. For mathematical reasoning, we use GSM8K (Cobbe et al., 2021) and AIME24 (HuggingFaceH4, 2024). For code generation, we use MBPP (Mostly Basic Python Problems) (Austin et al., 2021). For GSM8K and MBPP, we randomly sample 200 test prompts.

Generation setting. We implement our method on top of the high-performance `gpt-fast` library (Meta PyTorch Team, 2024). Experiments use the Qwen3 family (Yang et al., 2025): `Qwen3-8B` as the target and `Qwen3-0.6B` as the draft. We enable thinking and use decoding parameters temperature $= 0.6$, top_p $= 0.95$, top_k $= 20$. The maximum context length is 32,000 tokens and the speculative length is $\gamma = 10$. Experiments are executed on 8×A100 GPUs.

Metrics: For each task, we report accuracy (pass@1) and throughput (tokens per second). For speculative decoding–based methods (PAD and SD), we additionally report the draft acceptance ratio η and speed-up over target model. Metrics are averaged over 8 completions.

Results Table 1 presents the evaluation results. By adjusting the threshold σ, PAD provides a controllable trade-off between accuracy and efficiency. On the simpler tasks (GSM8K and MBPP), using larger σ values achieves greater speed-ups (up to $2.46\times$ versus $1.57\times$ for SD) while maintaining competitive accuracy. For the more challenging AIME24 task, maintaining high accuracy requires smaller σ values, which reduces the achievable speed-up relative to SD ($1.95\times$ versus $1.69\times$).

4 Related Works

EAGLE (Li et al., 2024) and Medusa (Cai et al., 2024) introduce alternative drafting strategies by leveraging additional decoding heads and specialized draft models, while incorporating verification in parallel with tree attention. Another line of work focuses on designing faster draft models through heuristics (Chen et al., 2024; He et al., 2024; Zhao et al., 2024) or adaptive draft lengths (Liu et al., 2025). Others aim to improve draft quality to increase acceptance rates, for example by aligning the draft distribution with the target (Zhou et al., 2024) or exploiting features of the target model (Du et al., 2024). These methods are orthogonal to our work, which instead emphasizes the verification phase. Closer to our approach, some methods relax strict distribution matching to improve acceptance. Kim et al. (2023) leverages token-level uncertainty, while (Bachmann et al., 2025) trains a classifier to accept or reject tokens. However, both rely on heuristic criteria rather than a principled utility-based formulation. Moreover, Bachmann et al. (2025) depends on hand-crafted datasets. By contrast, we reformulate decoding as a utility-based objective, yielding a fully self-supervised method grounded in task performance rather than manually designed criteria. More recently, other approaches (Liao et al., 2025; Fu et al., 2025; Pan et al., 2025) have proposed combining draft and target model outputs at the step level. However, these methods depend on an auxiliary reward model or process for scoring, which introduces additional overhead.

5 Conclusion

In this work, we proposed a novel reformulation of Speculative Decoding that focuses on preserving the utility of the target model rather than strictly matching its sampling distribution. Specifically, we introduced Pivot-Aware Speculative Decoding, a decoding strategy that rejects only the pivotal tokens, referred to as *pivot tokens*, which are likely to lead to a utility drop in the final output. To enable this, we trained a lightweight classifier to detect pivot tokens. Our method achieves a substantial speedup, up to $2.51\times$, while maintaining the performance and utility of the target model.

6 Limitations

This work introduces a general framework that serves as an alternative to speculative decoding. Its primary limitation is that maintaining the accuracy–efficiency trade-off hinges on training high-quality pivot classifiers. Several design choices remain open for exploration, including the optimal classifier architecture (e.g., multilayer perceptron vs. sequential models like RNNs) and the most effective feature sets for classifier inputs. These directions warrant systematic investigation to engineer more accurate and efficient classifiers. Additionally, while the classifier's computational cost is significantly lower than that of large language models, it nonetheless introduces some computational overhead relative to speculative decoding.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas K Kohler. 2025. Judge decoding: Faster speculative sampling requires going beyond model alignment. In *The Thirteenth International Conference on Learning Representations*.

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 *Forty-first International Conference on Machine Learning*.

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

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Kevin Chang, and Jie Huang. 2024. Cascade speculative drafting for even faster LLM inference. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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. *arXiv preprint arXiv:2110.14168*.

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and Yang You. 2024. Glide with a caPE: A low-hassle method to accelerate speculative decoding. In *Forty-first International Conference on Machine Learning*.

Yichao Fu, Rui Ge, Zelei Shao, Zhijie Deng, and Hao Zhang. 2025. Scaling speculative decoding with lookahead reasoning. *arXiv preprint arXiv:2506.19830*.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico. Association for Computational Linguistics.

HuggingFaceH4. 2024. Aime 2024. Hugging Face Datasets.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative decoding with big little decoder. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning*.

Baohao Liao, Yuhui Xu, Hanze Dong, Junnan Li, Christof Monz, Silvio Savarese, Doyen Sahoo, and Caiming Xiong. 2025. Reward-guided speculative decoding for efficient llm reasoning. *arXiv preprint arXiv:2501.19324*.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. 2025. PEARL: Parallel speculative decoding with adaptive draft length. In *The Thirteenth International Conference on Learning Representations*.

Meta PyTorch Team. 2024. gpt-fast: Pure pytorch gpt inference. `https://github.com/meta-pytorch/gpt-fast`. High-performance inference library.

OpenAI. 2023. GPT-4 Technical Report. *Preprint*, arXiv:2303.08774.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. 2025. Specreason: Fast and accurate inference-time compute via speculative reasoning. *arXiv preprint arXiv:2504.07891*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others.

2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Weilin Zhao, Yuxiang Huang, Xu Han, Wang Xu, Chaojun Xiao, Xinrong Zhang, Yewei Fang, Kaihuo Zhang, Zhiyuan Liu, and Maosong Sun. 2024. Ouroboros: Generating longer drafts phrase by phrase for faster speculative decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13378–13393, Miami, Florida, USA. Association for Computational Linguistics.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. Distillspec: Improving speculative decoding via knowledge distillation. In *The Twelfth International Conference on Learning Representations*.

A Technical Appendices and Supplementary Material

Technical appendices with additional results, figures, graphs and proofs may be submitted with the paper submission before the full submission deadline (see above), or as a separate PDF in the ZIP file below before the supplementary material deadline. There is no page limit for the technical appendices.

A.1 Proof of Lemma 1

Proof of Lemma 1. Let $f_{\text{pivot}}(x_{ot}, \mathbf{x}_{ot<}, \mathbf{x}_c)$ be a binary classifier with 100% recall, such that it correctly identifies all pivot tokens for any prefix $\mathbf{x}_{ot<}$ and context $\mathbf{x}_c$. We aim to show that, for any $\mathbf{x}_c$, the utility of the sampled output with any random seed from the target model, $u(p^*_{\text{target}}(\cdot \mid \mathbf{x}_c), \mathbf{x}_c)$, is equal to the utility of the sampled output with any random seed from the utility-aware decoding strategy, $u(p^*_{ad}(\cdot \mid \mathbf{x}_c), \mathbf{x}_c)$. This would naturally imply that the objective in Equation 1 is satisfied with $\epsilon = 0$.

We prove this by induction.

Define a completion function $C(\mathbf{x}_{\text{gen}}, \mathbf{x}_c)$ which, given a partially generated token sequence $\mathbf{x}_{\text{gen}}$ and context $\mathbf{x}_c$, returns the full sequence by completing $\mathbf{x}_{\text{gen}}$ using p_{target}, unless $\mathbf{x}_{\text{gen}}$ is already a completed sequence, in which case it returns $\mathbf{x}_{\text{gen}}$ directly.

Our goal is to show that for any generation prefix $\mathbf{x}_{ot}$ produced by p_{ad},

$$u(C(\mathbf{x}_{ot}, \mathbf{x}_c), \mathbf{x}_c) = u(p^*_{\text{target}}(\cdot \mid \mathbf{x}_c), \mathbf{x}_c), \forall t.$$

If this holds at every token generation step, it follows that the final utility of utility-aware decoding also matches that of the target model:

$$u(p^*_{ad}(\cdot \mid \mathbf{x}_c), \mathbf{x}_c) = u(p^*_{\text{target}}(\cdot \mid \mathbf{x}_c), \mathbf{x}_c),$$

since the full generation of p_{ad} corresponds to the final step.

Base Case: When no tokens have yet been generated, i.e., $\mathbf{x}_{ot} = ()$, we have:

$$u(C((), \mathbf{x}_c), \mathbf{x}_c) = u(p^*_{\text{target}}(\cdot \mid \mathbf{x}_c), \mathbf{x}_c),$$

by the definition of the completion function C.

Inductive Step: Assume that for some prefix $\mathbf{x}_{ot}$ up to step t, the condition holds:

$$u(C(\mathbf{x}_{ot}, \mathbf{x}_c), \mathbf{x}_c) = u(p^*_{\text{target}}(\cdot \mid \mathbf{x}_c), \mathbf{x}_c).$$

Now consider the generation of the next token x_{ot+1}. If x_{ot+1} is accepted by p_{ad}, it must not be an pivot token. By the definition of pivot tokens, adding x_{ot+1} does not change the final utility of $u(C(\mathbf{x}_{ot}, \mathbf{x}_c), \mathbf{x}_c)$, which is equal to $u(p^*_{\text{target},\mathbf{x}_c}(\cdot \mid \mathbf{x}_c), \mathbf{x}_c)$ by the inductive assumption. If the token is rejected, it is replaced by the greedy choice from p_{target}, which does not change the utility due to the inductive assumption and the definition of the completion function C. Specifically, we have:

$$\begin{aligned} u\big(p^*_{\text{target}}(\cdot \mid \mathbf{x}_c), \mathbf{x}_c\big) &= u\big(C(\mathbf{x}_{ot}, \mathbf{x}_c), \mathbf{x}_c\big) \\ &= u\big(C((x_{ot+1}, \mathbf{x}_{ot}), \mathbf{x}_c), \mathbf{x}_c\big). \end{aligned} \tag{3}$$

where x_{ot+1} is the greedy next token selected by p_{target}. Therefore:

$$u(C(\mathbf{x}_{ot+1}, \mathbf{x}_c)) = u(p^*_{\text{target}}(\cdot \mid \mathbf{x}_c)).$$

As this proof is for any sampling with any random seed and by induction, utility-aware decoding produces an output with the same utility as the target model for all $\mathbf{x}_c$, satisfying Equation 1 with $\epsilon = 0$.

□

B Dataset Generation and Pivot Classifier

Dataset generation. Algorithm 1 details the data collection procedure described in Section 2.1. We use the GSM8K training set (Cobbe et al., 2021) and generate 32619 labeled samples, with 3657 tokens labeled as *pivot* and 28962 as *non-pivot*. The target model is `Qwen3-8b` and the draft model is `Qwen3-0.6b`. We disable "thinking" to keep rollout costs manageable. Also, α is set to 0.8. For the soundness check, we employ `Gemini-Flash-Light` as the LLM-as-judge with medium reasoning effort[1]. The prompt used for the soundness check is provided below (Prompt B.1).

[1] Gemini API: `Gemini-Flash-Light`

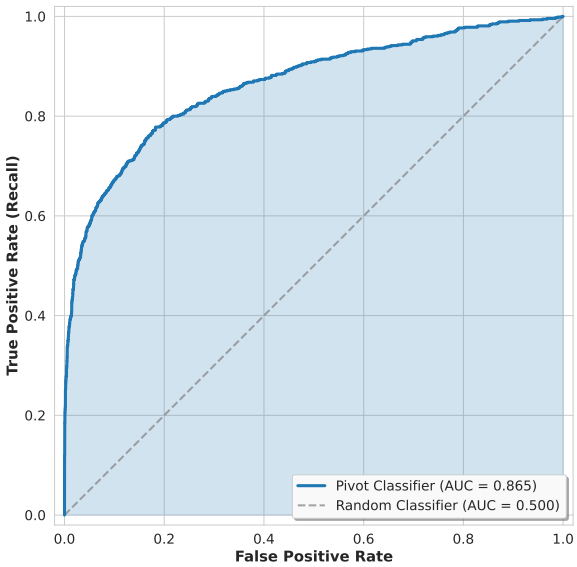

Figure 3: ROC curve for the pivot classifier on the held-out test set.

Pivot classifier. We predict whether a draft token is a pivot using an MLP over the target hidden state at layer l ($\mathbf{h}_l$), token entropy H, and the target-model probability of the candidate token p. Let $\mathbf{s} = [H, p]$ and define the concatenation operator $\oplus$ (i.e., $a \oplus b$ stacks vectors a and b). With ReLU $\phi(\cdot)$, and writing $\mathrm{Linear}(x) = Wx + b$, the model is:

$$
\begin{aligned}
\mathbf{u} &= \phi\big(\mathrm{Linear}_h(\mathbf{h}_l)\big), \\
\mathbf{v} &= \phi\big(\mathrm{Linear}_s(\mathbf{s})\big), \\
\mathbf{c} &= \mathbf{u} \oplus \mathbf{v}, \\
\boldsymbol{z} &= \mathrm{Linear}_{\text{out}}\Big(\phi\big(\mathrm{Linear}_{\text{fuse}}(\mathbf{c})\big)\Big), \quad \boldsymbol{z} \in \mathbb{R}^2, \\
\hat{\boldsymbol{y}} &= \mathrm{softmax}(\boldsymbol{z}),
\end{aligned}
$$

where $\mathrm{Linear}_{\text{fuse}}$ is a standard linear layer applied to the concatenated vector $\mathbf{c}$, and $\mathrm{Linear}_{\text{out}}$ maps the fused features to two logits (pivot / non-pivot). To address class imbalance, we train with weighted cross-entropy, split the data 80/20 into train/test, and select the checkpoint with the lowest validation loss. The ROC curve, obtained by sweeping the decision threshold σ, yields AUC $= 0.865$, well above the 0.5 random baseline, indicating robust separation of pivot and non-pivot tokens across thresholds (Fig. 3). At deployment, σ can be tuned to trade off acceptance rate against target-model fallbacks to meet a given latency/utility budget.

Algorithm 1: Pivot Classifier Data Collection

Input: $\mathcal{D}$ (prompts x), target p_t, draft p_d, SD_ACCEPT, SD_RESAMPLE, binary utility u, rollouts N, tolerance α, judge JudgeIsSound, steps T

Output: $\mathcal{S} = \{(\text{text}, \ell)\}$ with $\ell \in \{\texttt{pivot}, \texttt{non-pivot}\}$ (features extracted later)

```
func Rollouts(x, y_{<t}):
    sample N completions y^{(n)} ~ p_t(· | x, y_{<t});
    return {(y^{(n)}, u(y^{(n)}, x), |y^{(n)}|)}_{n=1}^{N}
func SelectRep(P):
    // P = {(y, u=1, ℓ)}
    sort P by sequence length ℓ; return middle element (median length)
S ← ∅;
for x ~ D do
    initialize prefix y_{<1} ← ∅;
    for t = 1 to T do
        // 1) Draft proposes next token
        sample ỹ_t ~ p_d(· | x, y_{<t});
        // 2) If SD would accept, append and move on (not a candidate)
        if SD_ACCEPT(x, y_{<t}, ỹ_t, p_t) then
            y_{<t+1} ← (y_{<t}, ỹ_t); continue
        // 3) Otherwise, evaluate MC utilities for base vs. candidate
        B ←Rollouts(x, y_{<t});   Û_base ← (1/|B|) Σ_{(y,u,*)∈B} u;
        C ←Rollouts(x, (y_{<t}, ỹ_t));   Û_cand ← (1/|C|) Σ_{(y,u,*)∈C} u;
        // 4) Label as pivot / non-pivot (with judge check for false wins)
        if Û_cand < α Û_base then
            ℓ ← pivot
        else
            ℓ ← non-pivot;
            P ← {(y, u, ℓ) : (y, u, ℓ) ∈ C, u = 1};
            if P ≠ ∅ then
                y^+ ←SelectRep(P);
                if ¬ JudgeIsSound(concat(x, y^+)) then
                    ℓ ← pivot
        // 5) Persist labeled sample for training (only when SD rejected)
        text ← concat(x, (y_{<t}, ỹ_t));
        append (text, ℓ) to S;
        // 6) Update prefix for next step per spec
        if ℓ = non-pivot then
            y_{<t+1} ← (y_{<t}, ỹ_t) ;                    // accepted for next step
        else
            // pivotal: fall back to SD rejection path (target-side resampling)
            y_t ← SD_RESAMPLE(x, y_{<t}, p_t);
            y_{<t+1} ← (y_{<t}, y_t)
return S ;                          // (run offline feature extraction next)
```

Prompt B.1: Reasoning Soundness Check

```
You are an expert evaluator specializing in identifying flawed reasoning in problem-solving
    narratives. Your task is to analyze a given response and determine if it contains logical
    errors, even if the final answer is correct.

Read the entire 'Response to Analyze' from start to finish. A response is considered to have
    flawed reasoning if you identify any of the following patterns:

- **Incorrect Intermediate Steps:** It contains calculation errors or incorrect logical steps
    that are later implicitly corrected or ignored to reach the correct final answer.
- **Logical Fallacies:** The reasoning contains leaps of logic or doesn't coherently lead to the
    conclusion, but the result coincidentally matches the correct answer.
- **Self-Correction:** The response states an incorrect piece of information or result and then
    corrects itself. Sometimes (not always) there are explicit phrases like "Wait, that's not
    right," "But the question asks for...," or "Let's try again" before the correct path is
    found.
- **Unnecessary or Redundant Steps:** It includes steps or calculations that are irrelevant and
    do not contribute to the final answer, indicating a confused or inefficient thought process.
- **Other Reasoning Flaws:** Any other type of faulty logic that raises questions about the
    soundness of the process.

---

### **Examples**

**EXAMPLE 1: FLAWED REASONING (Self-Correction)**

* **Question:** A bat and a ball cost \$1.10 in total. The bat costs \$1.00 more than the ball.
    How much does the ball cost?
* **Response to Analyze:** The bat costs \$1.00, so the ball must cost \$0.10. Wait, that's not
    right because the difference is only \$0.90. Let's try again. If the ball is B, the bat is B
     + 1. So B + (B + 1) = 1.10, which means 2B = 0.10. The ball costs \$0.05.
* **Your JSON Output:**
   ```json
   {{
     "analysis": "The response initially presents an incorrect answer (\$0.10) but then
         immediately identifies the error and uses a correct algebraic method to perform a self-
         correction, arriving at the right answer.",
     "decision": true
   }}
   ```

EXAMPLE 2: SOUND REASONING (No Flaws)

* **Question:** A bakery has 5 boxes of donuts, with 12 donuts in each box. They sell 3 boxes.
    How many donuts are left?
* **Response to Analyze:** First, determine the number of boxes remaining, which is 5 - 3 = 2
    boxes. Then, calculate the total donuts in the remaining boxes: 2 boxes * 12 donuts/box = 24
     donuts.
* **Your JSON Output:**
   ```json
   {{
     "analysis": "The reasoning is sound, direct, and efficient. It correctly calculates the
         remaining boxes before finding the total number of donuts left. There are no logical
         flaws.",
     "decision": false
   }}
   ```

EXAMPLE 3: FLAWED REASONING (Unnecessary Steps)

* **Question:** A recipe requires 2 cups of flour to make 12 cookies. You want to make 36 cookies
    . How much flour do you need?
* **Response to Analyze:** To make 36 cookies, which is 3 times 12, you'll need 3 times the flour
    . So, 2 cups * 3 = 6 cups. The oven should be preheated to 350F. The total flour needed is 6
     cups.
* **Your JSON Output:**
```

```
    ```json
    {{
      "analysis": "The core calculation is correct, but the response introduces an unnecessary
           and irrelevant piece of information ('The oven should be preheated to 350F') that does
           not contribute to solving the problem.",
      "decision": true
    }}
    ```

Input Data for Analysis

1. Question:
{question}

2. Golden Answer (for your reference):
{ground_truth}

3. Response to Analyze:
{response}

Your Task

After carefully reviewing the 'Response to Analyze', provide your evaluation in a strict JSON
     format, following the structure shown in the examples above. The JSON object must contain
     exactly two keys: `"analysis"` and `"decision"`.

Your JSON Output:
```