LOOPSPEC: PIPELINED SELF-SPECULATIVE DECODING FOR LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers achieve strong performance with compact parameter sizes by repeatedly applying a shared stack of Transformer blocks across recurrent depths. However, they incur higher decoding latency than standard Transformer models of comparable parameter size because shared weights are accessed at every recurrent depth. To improve decoding efficiency, self-speculative decoding is particularly well suited to Looped Transformers, as their intermediate recurrent states can directly provide draft predictions without an auxiliary draft model. We therefore propose LOOPSPEC, a training-free self-speculative decoding framework tailored for Looped Transformers. LOOPSPEC extracts draft tokens from early recurrent states and operates in a pipelined manner, overlapping draft generation of future tokens with target verification of the current token. To improve draft accuracy without excessive computational overhead, we introduce a selective second proposal from a deeper recurrent depth while ensuring lossless decoding under both greedy and sampling regimes. Furthermore, we derive the optimal proposal depths in closed form and show that the theoretical predictions closely match empirical measurements. Across reasoning and coding benchmarks, LOOPSPEC achieves up to 6.83× inference speedup across diverse Looped Transformers.

1 INTRODUCTION

Looped Transformers (Geiping et al., 2025; Zhu et al., 2025; McLeish et al., 2026; Nanbeige Lab et al., 2026) have recently emerged as an efficient architectural paradigm that iteratively applies shared Transformer blocks across recurrent depths, drastically reducing the total parameter count. For instance, a model of four blocks applied 8 times achieves the effective depth of a 32-layer model while storing only four layers of parameters. By expanding computational depth through this recurrence, Looped Transformers match or outperform standard Transformers of comparable parameter count on reasoning tasks. However, executing shared Transformer blocks across multiple recurrent depths repeatedly fetches identical parameters within each token generation step, leaving inference memory-bandwidth bound thus increasing decoding latency (Hooper et al., 2023).

While speculative decoding with a separately trained draft model (Leviathan et al., 2023; Chen et al., 2023; Li et al., 2025; Chen et al., 2026) has emerged as a promising approach for lossless inference acceleration, it introduces additional training costs, potential distribution mismatch between draft and target models, and extra memory. Self-speculative decoding (Zhang et al., 2024; Elhoushi et al., 2024; Cha et al., 2026) avoids these limitations by using intermediate computations of the target model itself to generate draft tokens, eliminating the need for a separate draft model. Looped Transformers are naturally suited to this paradigm, since the intermediate representation in each recurrent depth forms a sequence of increasingly refined proposals toward the final prediction.

To this end, we propose LOOPSPEC, a training-free self-speculative pipelining framework specifically designed for Looped Transformers. The key idea is to exploit the token predictions from the intermediate recurrent depths. Rather than waiting for all recurrent steps to finish, LOOPSPEC drafts a token from an early recurrent state and immediately starts computing the subsequent token conditioned on this draft (Figure 1(b)). The original computation and this speculative continuation use the same Transformer blocks, so they can be processed together in one batch even at different recurrent depths. Once the original computation reaches its final depth, it verifies the early draft. If the draft is accepted, some recurrent steps for the subsequent token have already been completed, so fewer

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

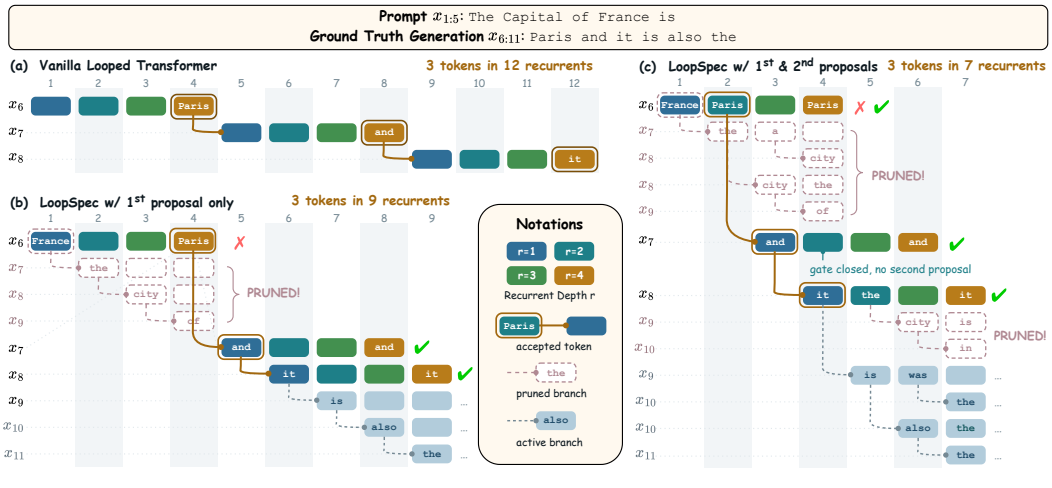


Figure 1: LOOPSPEC pipelines decoding over $R = 4$ recurrent depth. (a) A vanilla Looped Transformer finishes all R recurrences of one token before starting the next. (b) LOOPSPEC with a first proposal only at $d_1 = 1$: an early-depth draft starts the next token immediately, but a rejection stalls the pipeline (Section 4.2). (c) LOOPSPEC with first proposal at $d_1 = 1$ and a residual second proposal with gating mechanism at $d_2 = 2$: the second proposal creates a fallback branch, causing fewer stalls in the pipeline (Section 4.3).

steps are needed before it can reach the final depth. Otherwise, we prune the speculative draft and restart from the verified prefix.

Since deeper recurrent computations progressively refine the model's prediction, LOOPSPEC uses a deeper recurrent state to produce a second proposal and starts an alternative continuation before verification. Accepting this second proposal preserves its progress when the first proposal is rejected (Figure 1(c)). The resulting computation forms a pipeline in which the target computation and multiple speculative continuations advance in parallel, substantially increasing hardware utilization while preserving the exact target-model distribution. Figure 1 provides an overview of the complete LOOPSPEC pipeline.

Our primary contributions can be summarized as follows:

- We propose LOOPSPEC, a training-free self-speculative pipelining framework that accelerates Looped Transformers without external drafters or structural modifications. To improve draft acceptance while limiting computational overhead, LOOPSPEC introduces (i) a *residual* second proposal from a deeper recurrent state to recover from first proposal rejection, and (ii) a *gating mechanism* that creates the fallback speculative continuation only when the deeper state disagrees with the first draft (Sections 4.2 and 4.3).
- We prove that the restart probability depends only on the second proposal depth, not the first. Using the empirically observed power-law decay of the intermediate-to-target TV distance, we derive the optimal proposal depths in closed form matching the measured optima (Section 5).
- We empirically demonstrate up to **6.83**× lossless speedup across seven checkpoints from two Looped Transformer families through an SGLang (Zheng et al., 2024) implementation with branch-wise KV management and CUDA Graph optimization, outperforming the training-based DFlash (Chen et al., 2026) by up to 1.7× (Section 6 and B).

2 RELATED WORK

Speculative Decoding. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) is a lossless inference acceleration technique that uses a lightweight drafting mechanism to generate multiple candidate tokens, which are then verified in parallel by the target model. Drafting mechanisms typically employ either a smaller pretrained model from the same architecture family (Leviathan et al., 2023) or a specially trained external drafter. Modern state-of-the-art drafters, such as EAGLE-3 (Li

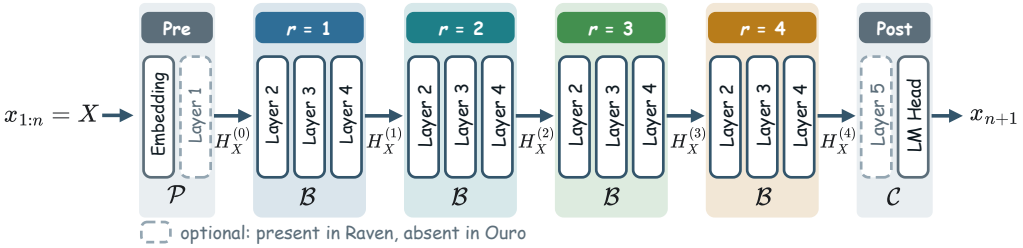


Figure 2: Overview of a Looped Transformer with $R = 4$ recurrent steps. In `Ouro` models (Zhu et al., 2025), the pre-layers $\mathcal{P}$ and post-layers $\mathcal{C}$ contain only the Embedding layer and LM Head, respectively, whereas in `Raven` models (McLeish et al., 2026) both $\mathcal{P}$ and $\mathcal{C}$ contain several Transformer layers.

et al., 2025) and DFlash (Chen et al., 2026), achieve high acceptance rates by conditioning proposals on intermediate target representations. Self-speculative decoding instead generates draft tokens using the target model itself without a separate external drafter. Draft & Verify (Zhang et al., 2024) obtains drafts by skipping intermediate layers, and LayerSkip (Elhoushi et al., 2024) utilizes early exits. Building on intermediate-layer predictions, PPD (Yang et al., 2024) forms a speculative decoding pipeline that launches the computation of subsequent tokens on additional compute resources while the current token continues through the remaining layers.

Looped Transformers. Looped Transformers (Geiping et al., 2025; Zhu et al., 2025; McLeish et al., 2026; Park et al., 2026; Nanbeige Lab et al., 2026) repeatedly apply shared Transformer blocks across recurrent depths, increasing effective depth without proportionally increasing the number of parameters. While they achieve higher task accuracy than standard Transformers of comparable size, their recurrent computation incurs substantially greater memory traffic during inference, limiting decoding throughput.

Recent work reduces recurrent inference cost by modifying the model architecture or adapting the number of recurrent steps. LT2 (Deng et al., 2026) substitutes full attention with linear or sparse attention variants and distills hybrid checkpoints, while Think-at-Hard (Fu et al., 2026) incorporates a routing controller with depth-aware adapters to adjust recurrent passes dynamically. Rather than altering attention or adding routing heads, SPEED (Hooper et al., 2023) explores speculative pipelining across cyclically shared decoder groups. However, SPEED requires a custom training process and only supports greedy decoding.

3 Preliminaries

3.1 Looped Transformers

For a sequence prefix $X = x_{1:n}$ of length n, the Looped Transformer computation can be abstracted into (1) pre-layers $\mathcal{P}$, (2) recurrent-layers $\mathcal{B}$, and (3) post-layers $\mathcal{C}$:

$$H_X^{(0)} = \mathcal{P}(X), \quad H_X^{(r)} = \mathcal{B}\left(H_X^{(r-1)}\right), \quad p_R(\cdot \mid X) = \mathcal{C}\left(H_X^{(R)}\right). \tag{1}$$

Here $H_X^{(0)}$ and $H_X^{(r)}$ denote hidden states before recurrence and at recurrent depth $r \in \{1, \cdots, R\}$, respectively; r indexes recurrent depth, and R denotes the total recurrent depth of the model. $x_{n+1} \sim p_R(\cdot \mid X)$ is the target next-token probability distribution. The pre-layers $\mathcal{P}$ include the token embedding, and the post-layers $\mathcal{C}$ incorporate the LM head as well as temperature scaling, optional top-k/top-p filtering, and softmax normalization to output a probability distribution over the vocabulary $\mathcal{V}$.

More generally, applying the post-layers at any recurrent depth r yields an intermediate next-token distribution $p_r(\cdot \mid X) = \mathcal{C}(H_X^{(r)})$, of which the target distribution p_R is the special case $r = R$. Existing Looped Transformers instantiate this abstraction in different ways. `Ouro` (Zhu et al., 2025) places all Transformer blocks inside $\mathcal{B}$, whereas `Raven` (McLeish et al., 2026) places only a subset of intermediate blocks in $\mathcal{B}$ and the remaining ones in $\mathcal{P}$ and $\mathcal{C}$ (Figure 2).

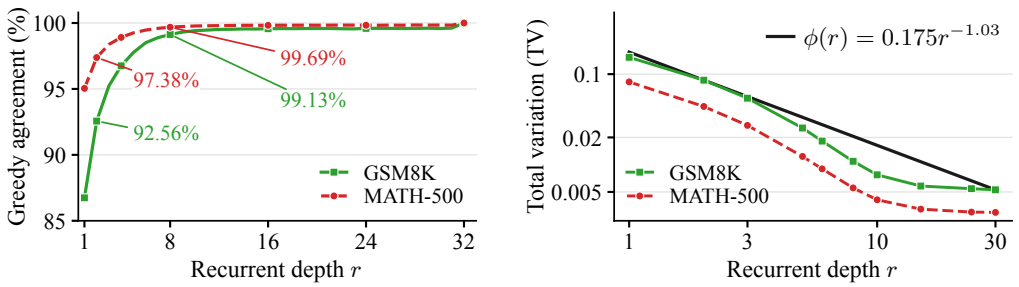


Figure 3: Comparison between intermediate proposals at depth r and the target distribution at depth $R = 32$ for `Raven-Llama-3.2`. (a) Greedy top-1 agreement on GSM8K and MATH-500. (b) Sampling ($T = 1.0$, top-$p = 0.7$) total variation (TV) distance to the target on the same benchmarks. TV is obtained on 10 depths over GSM8K and MATH-500, averaged per token position on top-p distribution. $\phi(r)$ is the empirical power-law upper envelope for the TV distance, constructed as described in Section 4.1.

3.2 Rejection Sampling and Speculative Decoding

For any two distinct probability distributions μ and ν over a vocabulary $\mathcal{V}$, we define their element-wise positive difference as $[\mu - \nu]_+(v) \coloneqq \max(\mu(v) - \nu(v), 0)$ for each $v \in \mathcal{V}$, and the residual distribution operator $(\mu \ominus \nu)$ as:

$$(\mu \ominus \nu)(v) \coloneqq \frac{[\mu - \nu]_+(v)}{\sum_{u \in \mathcal{V}} [\mu - \nu]_+(u)}. \tag{2}$$

The core idea of speculative decoding (Leviathan et al., 2023; Chen et al., 2023) is to draft fast and then verify via rejection sampling. We state the rule for a single token position, which is the form LoopSpec builds on.

Formally, let X be the current prefix, and let $p(\cdot \mid X)$ and $q(\cdot \mid X)$ denote the target and the proposal distribution of the next token, respectively. A candidate $\widetilde{x}$ is drawn from the proposal distribution $q(\cdot \mid X)$ and accepted with probability $\min\left\{1, \frac{p(\widetilde{x}|X)}{q(\widetilde{x}|X)}\right\}$. If the candidate $\widetilde{x}$ is rejected, the next token x is instead resampled from the residual distribution $x \sim (p(\cdot \mid X) \ominus q(\cdot \mid X))$. Leviathan et al. (2023) show that the token produced by rejection sampling is distributed exactly as a token sampled from the target p, so the procedure is lossless. Crucially, losslessness holds for *any* valid proposal distribution q.

4 LoopSpec: Pipelined Self-Speculative Decoding for Looped Transformers

We present LoopSpec, a training-free self-speculative decoding framework for Looped Transformers. We first show that early recurrent states can provide effective draft proposals (Section 4.1). Building on this observation, we introduce a pipeline with a single proposal depth $d_1 < R$, where each draft starts computation for the next token before final-depth verification (Section 4.2). As decoding is memory-bandwidth bound, batching does not introduce noticeable overhead. We then extend the pipeline with a second proposal depth d_2, using the residual distribution and the gating mechanism to further improve the proposal quality and acceptance (Section 4.3).

4.1 Observation: Early Recurrent States are Effective Drafters

In Looped Transformers, proposals derived from intermediate recurrent states can tightly approximate the target distribution. In Figure 3, we observe this behavior in the `Raven-Llama-3.2` model (McLeish et al., 2026). For instance, on GSM8K (Cobbe et al., 2021), the depth-1 greedy agreement exceeds 86% and surpasses 99% by depth 8, with a similar trend on MATH-500

(Hendrycks et al., 2021; Lewkowycz et al., 2022; Kydlicek et al., 2025; Lightman et al., 2024). We further estimate the empirical total variation (TV) distance between the intermediate readout and the target distributions under the same setting, and observe that it decreases sharply as recurrence depth r increases. To characterize this, we bound the empirical TV distance with a power-law upper envelope $\phi(r) = \beta r^{-\alpha}$. Specifically, we choose parameters $\alpha, \beta > 0$ to minimize the maximum ratio between $\phi(r)$ and the empirical TV distance over all depths r. This power-law upper envelope allows us to analyze the optimal depth configuration studied in Section 5.

These observations motivate using early recurrent states to draft future tokens, while continuing recurrence to verify the drafts at depth R. Deeper intermediate states can further serve as fallback proposals when earlier predictions diverge from the target.

4.2 Pipelined Self-Speculative Decoding with a Single Proposal

We first describe the single-proposal pipeline in Figure 1(b), where each token is drafted at only one intermediate depth d_1 where $d_1 < R$, and verified at the final depth R. Unlike vanilla decoding, which completes all R recurrences before moving to the next token (Figure 1(a)), this schedule overlaps computation across token positions (Figure 1(b)). We call the recurrent computation for one prefix a *branch*, represented as a horizontal sequence of recurrent states in the figure. A draft starts a child branch, while its parent continues toward verification. Multiple branches can remain active at different recurrent depths and crucially they are advanced by a *single batched call* to the shared block.

For a branch b, we denote its prefix by $X^{(b)}$ and use the hidden states $H^{(r)}_{X^{(b)}}$ and readouts $p_r(\cdot \mid X^{(b)})$ from Section 3.1. We write p_r when the prefix is clear. After prefilling the prompt and committing the first decoded token, LoopSpec initializes a branch on the resulting prefix at depth 0 and repeats the following steps:

1. **Batched Recurrence.** All active branches advance by one recurrent depth through a single batched call to $\mathcal{B}$. In Figure 1(b), each column represents one such step across the active branches.
2. **Early Drafting.** When branch b reaches d_1, it reads out $q_1 := p_{d_1} = \mathcal{C}(H^{(d_1)}_{X^{(b)}})$ and samples $\widetilde{x}^{(1)}_b \sim q_1$. A child branch is then initialized at depth 0 with the extended prefix $X^{(b)} \,\|\, \widetilde{x}^{(1)}_b$. For example, with $d_1 = 1$ in Figure 1(b), the draft "France" at timestep 1 starts a child at timestep 2 before "France" is verified at timestep 4.
3. **Verification and Continuation.** When the branch reaches R, it verifies $\widetilde{x}^{(1)}_b$ against p_R using rejection sampling as Section 3.2. If accepted, the draft is committed and decoding continues from its child, which has already completed $R - d_1$ recurrent steps. In Figure 1(b), accepting "and" at timestep 8 preserves its child branch that started at timestep 6. If rejected, a replacement $x_b \sim p_R \ominus q_1$ is committed, all speculative descendants of b are pruned, and a new branch starts on $X^{(b)} \,\|\, x_b$ at depth 0. The "France" draft at timestep 1 in Figure 1(b) illustrates this case: depth-R commits "Paris", which differs from the draft "France", so we prune the descendants of "France" and restart computation for the next token.

With sustained draft acceptance, tokens are committed every d_1 recurrent steps rather than every R steps. However, each rejection discards the speculative progress and requires another R recurrent steps before the next commitment. This motivates a more accurate second proposal that can provide a fallback draft when the first draft is rejected.

4.3 Residual Second Proposal with Gating Mechanism

We now extend the single-proposal pipeline in Section 4.2 by adding a second proposal depth d_2, with $d_1 < d_2 < R$. At depth d_2, a branch can draft an alternative token for the same position, starting a fallback child alongside the first child. Accepting this fallback preserves its progress instead of restarting the pipeline. We next describe how to construct the fallback proposal, when to create it, and how to verify the two candidates.

Residual Second Proposal. The second proposal is used only after the first proposal is rejected, so the required target distribution is $p_R \ominus q_1$. We therefore draft from an estimate of this residual

target. When branch b reaches d_2, it obtains a deeper readout $\widetilde{q}_2 \coloneqq p_{d_2} = \mathcal{C}(H^{(d_2)}_{X^{(b)}})$ and forms

$$q_2 \coloneqq \widetilde{q}_2 \ominus q_1 = p_{d_2} \ominus p_{d_1}, \qquad \widetilde{x}_b^{(2)} \sim q_2. \tag{3}$$

Here we use $\widetilde{q}_2$ as an estimator for p_R and sample the second proposal from the residual distribution q_2. A second child then starts from depth 0 with the prefix $X^{(b)} \,\|\, \widetilde{x}_b^{(2)}$. The two proposals represent alternative branches for the same token position, as illustrated by the "France" and "Paris" drafts at timesteps 1 and 2 in the top row of Figure 1(c). Our ablation study in Section 6.3 shows that the residual second proposal q_2 consistently improves the decoding speedup over the non-residual second proposal $\widetilde{q}_2$ across various benchmarks.

Gating Mechanism. Creating a fallback at every branch would rapidly increase the number of active branches, since each child can itself issue two proposals. Specifically, if every active branch forks at both d_1 and d_2, the number of new branches s_k spawned in the k-th interval of d_1 recurrent steps obeys the delayed recurrence $s_k = s_{k-1} + s_{k-d_2/d_1}$, where $s_k = 1$ for $0 \le k < d_2/d_1$[1]. For $R = 32$, $d_1 = 2$, and $d_2 = 8$, this gives maximum branch count $B = \sum_{k=0}^{R/d_1-1} s_k = 249$. To limit this growth of B, LOOPSPEC opens the gate of second proposal and creates a second proposal branch only if

$$\widetilde{q}_2(\widetilde{x}_b^{(1)}) < q_1(\widetilde{x}_b^{(1)}). \tag{4}$$

Intuitively, a fallback branch is created when the deeper recurrent state withdraws confidence from the primary candidate. Under greedy decoding, the rule reduces to $\arg\max \widetilde{q}_2 \neq \widetilde{x}_b^{(1)}$. In Figure 1(c), the gate stays closed for the "and" branch where no second proposal is triggered at timestep 4. Section 6.3 evaluates the benefit of the gating mechanism. Moreover, Section C examines the gate opening frequency and its effect on the effective batch size.

Cascade Verification. When a branch b reaches depth R, it produces the exact target distribution p_R and a cascade verification attempts to accept the primary candidate, then the fallback candidate, and restarts the pipeline only if neither candidate is accepted.

1. **First Proposal.** Accept $\widetilde{x}_b^{(1)}$ against p_R with probability $\min\left\{1, p_R(\widetilde{x}_b^{(1)})/q_1(\widetilde{x}_b^{(1)})\right\}$. If accepted, commit it and keep its descendants, and prune the second child's subtree if present. In Figure 1(c), at timestep 7, accepting the first draft "it" from timestep 4 preserves its continuation while pruning the second draft "the" from timestep 5 and its subtree.
2. **Second Proposal.** If the first proposal is rejected and a second proposal exists, verify $\widetilde{x}_b^{(2)}$ against $\rho_1 \coloneqq p_R \ominus q_1$ with probability $\min\left\{1, \rho_1(\widetilde{x}_b^{(2)})/q_2(\widetilde{x}_b^{(2)})\right\}$. If accepted, commit it and keep its descendants, and prune the first child's subtree. This child has already completed $R - d_2$ recurrent steps, so the next commitment requires only d_2 more steps instead of R. This is the rejection of "France" and acceptance of the second draft "Paris" in the top 5 rows of Figure 1(c).
3. **Residual Resampling.** If both proposals are rejected, sample $x_b \sim \rho_1 \ominus q_2$. If the first is rejected and the gate was closed, sample $x_b \sim \rho_1$ instead. Commit x_b, prune the speculative descendants of b, and start a fresh branch on $X^{(b)} \,\|\, x_b$ at depth 0.

Under greedy decoding, this cascade checks whether $\widetilde{x}_b^{(1)}$ matches $\arg\max p_R$, falling back to $\widetilde{x}_b^{(2)}$ if available, and committing $\arg\max p_R$ otherwise. The algorithm is provided in Section H.

5 THEORETICAL ANALYSIS

In this section, we provide theoretical analysis of LOOPSPEC from two perspectives. First, we show that the restart probability depends only on the approximation error of the second proposal and is independent of that of the first proposal. Second, we use the power-law decay of approximation error across recurrent depths, as observed in Figure 3(b), to derive the optimal proposal depths that minimize the expected number of recurrent steps per generated token. The optimal proposal depths derived from our analysis closely match the empirically optimal depth configurations.

[1]Throughout the paper, we consider that d_2 is divisible by d_1.

Theorem 1 (Rejection Rate of LOOPSPEC). *Fix proposal depths $1 \le d_1 < d_2 < R$ and let ε_2 be the total variation distance between p_{d_2} and p_R. Under the residual second proposal $q_2 = p_{d_2} \ominus p_{d_1}$ and cascade verification, the probability that both candidates are rejected at a given position is at most ε_2, independently of d_1. Moreover, with the gating mechanism, the bound is at most $2\varepsilon_2$.*

Proofs of all theorems are provided in Section G. Theorem 1 implies that the residual proposal $p_{d_2} \ominus p_{d_1}$ achieves the same rejection bound ε_2 as a proposal drawn directly from p_{d_2}, while avoiding any additional dependence on d_1. Thus, introducing the residual proposal does not worsen the rejection bound compared with using p_{d_2} directly. With the gating mechanism described in Section 4.3, the bound increases only by a constant factor, from ε_2 to $2\varepsilon_2$.

This result allows us to characterize the cost of a rejection using only the approximation quality at the second proposal depth d_2. Hence, we can focus on how the approximation error decreases with recurrent depth to derive the proposal depths that minimize the expected decoding cost. Motivated by the observations in Figure 3(b), we assume a non-increasing power-law envelope $\phi(r) = \beta r^{-\alpha}$ that upper bounds the expected total variation distance between the intermediate readout distribution p_r and the target distribution p_R.

Theorem 2 (Proposal Depth Selection). *Let X be a random prefix drawn from a fixed distribution over prefixes. Assume that there exist constants $\alpha, \beta > 0$ such that*

$$\mathbb{E}_X \left[\mathrm{TV}\left(p_r(\cdot \mid X), p_R(\cdot \mid X)\right) \right] \le \beta r^{-\alpha}. \tag{5}$$

for every $r = 1, ..., R$ and the total depth $R > \frac{1}{2\alpha} \max\{(\alpha\beta)^{1/\alpha}, (\alpha\beta)^{-\alpha-1}, (\alpha\beta)^{1/\alpha}(2\alpha)^{\frac{\alpha^2+\alpha+1}{\alpha^2}}\}$. Then, the optimal proposal depths to minimize expected number of recurrent steps to commit a single token are

$$d_1^\star = (\alpha\beta)^{\frac{\alpha+1}{\alpha^2+\alpha+1}} (2\alpha R)^{\frac{1}{\alpha^2+\alpha+1}}, \qquad d_2^\star = (\alpha\beta)^{\frac{\alpha}{\alpha^2+\alpha+1}} (2\alpha R)^{\frac{\alpha+1}{\alpha^2+\alpha+1}}. \tag{6}$$

As shown in Figure 3(b), we obtain an empirical upper bound with $\beta = 0.175$ and $\alpha = 1.03$ for `Raven-Llama-3.2` model with $R = 32$. Substituting these values into Equation (6) gives us $(d_1^\star, d_2^\star) = (1.26, 8.85)$. In Section 6, we explore various (d_1, d_2) configurations and find that $(d_1, d_2) = (2, 8)$ performs best for the same model. This demonstrates that our theoretical analysis closely captures the empirically optimal depth configuration.

6 EXPERIMENTS

Models and Evaluations. We evaluate LOOPSPEC across two representative Looped Transformer families: the `Ouro` family with $R = 4$ (Zhu et al., 2025) and the `Raven` family with $R = 32$ (McLeish et al., 2026). Evaluations cover mathematical reasoning, general reasoning, and code generation: GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021; Lewkowycz et al., 2022; Kydlicek et al., 2025; Lightman et al., 2024), BBH (Suzgun et al., 2023), HumanEval+ and MBPP+ (Chen et al., 2021; Austin et al., 2021; Liu et al., 2023) for base models, alongside GSM8K-CoT, MATH-500, AIME 2024, and AIME 2025 (Mathematical Association of America, 2024; 2025) for reasoning models. Base models are evaluated under greedy decoding, whereas reasoning models are evaluated using chat templates with thinking modes enabled, under both greedy decoding and sampling ($T = 1.0$, top-$p = 0.7$). All evaluations are conducted in a single-batch setting integrated with `lm-evaluation-harness` (Gao et al., 2023). Detailed checkpoint identifiers and benchmark configurations are provided in Section A.

Implementation and Metrics. We implement LOOPSPEC by extending the SGLang serving engine (Zheng et al., 2024). All experiments are executed on an NVIDIA RTX PRO 6000 Blackwell GPU in BF16 precision using the Triton attention backend and the TinyGEMM backend provided by FlashInfer (Ye et al., 2025). Further SGLang implementation details are in Section B. We report speedup on wall-clock time and mean accepted length γ. Wall-clock time speedup is measured over the decoding phase, after the completion of prompt prefilling until the end of generation. We define γ as the average number of generated tokens produced per full model execution of R recurrent steps:

$$\gamma := \frac{N_{\text{decode}} \cdot R}{N_{\text{accept},1} \cdot d_1 + N_{\text{accept},2} \cdot d_2 + (N_{\text{decode}} - N_{\text{accept},1} - N_{\text{accept},2}) \cdot R}, \tag{7}$$

Table 1: LOOPSPEC decoding speedup over the standard autoregressive decoding and mean accepted length γ on general reasoning, math, and code benchmarks across different proposal depths. Bold values mark per-model, per-benchmark maxima, with speedup and γ selected independently.

Models	Proposal Depths	GSM8K		MATH-500		BBH		HumanEval+		MBPP+	
		Speedup	γ	Speedup	γ	Speedup	γ	Speedup	γ	Speedup	γ
Greedy Setting: Temperature=0.0											
`Ouro-1.4B`	1	2.64×	3.13	2.64×	3.25	2.66×	3.30	**3.17**×	3.73	2.79×	3.36
	1, 2	**2.85**×	**3.50**	**2.78**×	**3.55**	**2.79**×	**3.59**	3.16×	**3.86**	**2.92**×	**3.64**
`Ouro-2.6B`	1	3.03×	3.43	2.93×	3.50	2.89×	3.49	**3.36**×	3.78	3.00×	3.44
	1, 2	**3.12**×	**3.68**	**2.99**×	**3.72**	**2.98**×	**3.72**	3.33×	**3.89**	**3.09**×	**3.69**
`Raven-Llama-3.2`	2	3.04×	7.62	4.77×	11.56	4.19×	10.35	4.93×	11.73	3.96×	9.70
	4	4.06×	6.55	4.76×	7.43	4.57×	7.21	4.89×	7.52	4.55×	7.14
	4, 8	4.33×	7.51	4.90×	7.83	4.76×	7.74	5.01×	7.84	4.72×	7.70
	2, 10	4.38×	12.02	5.70×	14.40	5.32×	13.72	5.83×	14.36	5.07×	13.08
	2, 8	**4.49**×	**12.50**	**5.82**×	**14.60**	**5.39**×	**13.98**	**5.88**×	**14.55**	**5.14**×	**13.44**
`Raven-OLMo-2-0425`	2	3.75×	8.95	3.16×	8.34	3.52×	9.24	5.05×	12.12	5.04×	12.06
	4	4.38×	6.97	4.04×	6.94	4.17×	7.07	4.78×	7.59	4.81×	7.62
	4, 8	4.60×	7.67	4.26×	7.67	4.38×	7.73	4.86×	7.87	4.89×	7.88
	2, 10	5.06×	12.87	**4.29**×	12.65	4.57×	13.17	5.75×	14.37	5.86×	14.56
	2, 8	**5.20**×	**13.35**	**4.29**×	**13.11**	**4.66**×	**13.60**	**5.86**×	**14.74**	**5.97**×	**14.86**
`Raven-TinyLlama-3T`	2	3.86×	8.18	prompts exceed max context length				5.85×	11.94	4.60×	9.69
	4	4.69×	6.72					5.40×	7.46	4.98×	7.08
	4, 8	5.01×	7.58					5.58×	7.85	5.18×	7.71
	2, 10	5.38×	12.43					**6.83**×	14.49	**5.96**×	13.37
	2, 8	**5.53**×	**12.89**					6.81×	**14.70**	5.76×	**13.55**

where N_{decode} is the number of generated tokens, and $N_{\text{accept},1}$ and $N_{\text{accept},2}$ denote the number of tokens accepted from the first and second proposals, respectively. By construction, $\gamma \leq R/d_1$, with the upper bound attained when all tokens are accepted at the first proposal depth d_1.

6.1 BASE MODELS

Table 1 presents the decoding speedup and mean accepted length γ across base model checkpoints. Across various tasks, LOOPSPEC provides consistent speedups over standard autoregressive decoding. For the `Ouro` family ($R = 4$), LOOPSPEC achieves speedups up to $3.36\times$, with γ exceeding 3.5 out of $R/d_1 = 4$. For the deeper `Raven` family ($R = 32$), LOOPSPEC yields peak speedups ranging from $5.88\times$ to $6.83\times$ across models, with γ exceeding 14.5. Here, while a single proposal at depth $d_1 = 2$ suffers from lower acceptance rates, adding a second proposal (e.g., $d_1 = 2, d_2 = 8$) compensates for early errors, allowing the first proposal depth to be pushed down to $d_1 = 2$ to enlarge the pipeline depth and maximize speedup. In Section E, we further show that the first proposal accounts for over 86% of committed tokens, while the second proposal recovers the majority of the remainder, leaving at most 2.95% of all committed tokens to full pipeline restart.

Notably, γ measures the reduction in serial recurrent steps but excludes the cost of $\mathcal{P}$ and $\mathcal{C}$ computation. A smaller d_1 triggers more frequent drafting and branch initialization, increasing these costs, particularly in Raven models with heavy pre- and post-layers. This overhead can outweigh the recurrent step savings, explaining why some configurations on `Raven` models achieve higher γ values but lower wall-clock time speedup.

6.2 REASONING MODELS

Table 2 reports the performance of LOOPSPEC in reasoning models with the thinking mode enabled. Across long-context chain-of-thought generation on GSM8K, MATH-500, and AIME 2024/2025

Table 2: LOOPSPEC decoding speedup over the standard autoregressive decoding and mean accepted length γ with thinking enabled across different proposal depths. Results under temperature 0.0 and 1.0 (with top-$p = 0.7$) are reported separately. Bold values mark per-model, per-benchmark maxima, with speedup and γ selected independently.

Models	Proposal Depths	GSM8K		MATH-500		AIME 2024		AIME 2025	
		Speedup	γ	Speedup	γ	Speedup	γ	Speedup	γ
Greedy Setting: Temperature=0.0									
`Ouro-1.4B-Thinking`	1	2.47×	2.95	2.49×	3.12	**2.56**×	3.26	2.45×	3.23
	1, 2	**2.70**×	**3.36**	**2.65**×	**3.50**	2.52×	**3.54**	**2.59**×	**3.55**
`Ouro-2.6B-Thinking`	1	2.76×	3.21	2.63×	3.33	2.53×	3.31	2.58×	3.22
	1, 2	**2.95**×	**3.55**	**2.78**×	**3.64**	**2.58**×	**3.63**	**2.66**×	**3.64**
Sampling Setting: Temperature=1.0									
`Ouro-1.4B-Thinking`	1	2.11×	2.64	2.17×	2.88	2.07×	2.77	2.05×	2.76
	1, 2	**2.42**×	**3.21**	**2.41**×	**3.35**	**2.34**×	**3.31**	**2.27**×	**3.28**
`Ouro-2.6B-Thinking`	1	2.62×	3.11	2.52×	3.21	2.37×	3.14	2.39×	3.10
	1, 2	**2.83**×	**3.54**	**2.70**×	**3.61**	**2.51**×	**3.54**	**2.65**×	**3.53**

Table 3: Ablation study of different second proposal methods in LOOPSPEC.

Model: `Raven-Llama-3.2`	GSM8K	MATH-500	HumanEval+	MBPP+
Standard Autoregressive Decoding	1.00×	1.00×	1.00×	1.00×
+ First & Non-residual Second Proposal	3.24×	4.49×	4.59×	4.07×
+ Residual Second Proposal	3.52×	4.73×	4.76×	4.57×
+ Gating Mechanism	**3.74**×	**4.85**×	**5.00**×	**4.71**×

benchmarks, LOOPSPEC delivers speedups up to 2.95×. For sampling, LOOPSPEC maintains competitive speedups up to 2.83×. Here, the second proposal proves particularly effective: drafting from the residual distribution upon rejection increases γ from 3.11 to 3.54 on `Ouro-2.6B-Thinking` under the GSM8K benchmark, yielding substantial speedup gains over using only the first proposal.

6.3 ABLATION STUDY

Table 3 presents the results of an ablation study on the second proposal in LOOPSPEC, which are the residual second proposal and the gating mechanism. The ablation study is conducted on `Raven-Llama-3.2` model under sampling ($T = 1.0$, top-$p = 0.7$). We mask out the first proposal token $\widetilde{x}_b^{(1)}$ from $\widetilde{q}_2$ for the non-residual second proposal and from q_2 for the residual second proposal, and renormalize the corresponding distribution before sampling. Table 3 shows replacing the non-residual second proposal to residual second proposal and adding the gating mechanism consistently improve the speedups. Section C further examines how often the gate opens and how gating reduces the effective batch size during decoding.

In Section D, we provide comparison of LOOPSPEC with DFlash (Chen et al., 2026) and LayerSkip (Elhoushi et al., 2024). LOOPSPEC consistently outperforms both of the counterparts.

7 CONCLUSION

We introduced LOOPSPEC, a training-free self-speculative decoding framework for Looped Transformers that converts recurrent depth into token-position parallelism. An early recurrent state drafts the next token and immediately begins computing it while the current token continues toward final-depth verification. Furthermore, we proposed a residual second proposal with a gating mechanism from a deeper recurrent depth alongside a cascade verification that keeps decoding lossless. Across the `Ouro` ($R = 4$) and `Raven` ($R = 32$) families, LOOPSPEC achieves speedups of up to 6.83× on general reasoning, math, and coding benchmarks.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic data for DFlash training following the procedure described in Section D, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, and assist in implementing methods, including LOOPSPEC, by generating or editing implementation code. We also used generative AI tools to identify relevant literature and to edit the manuscript to improve readability.

We did not use generative AI tools to formulate mathematical claims, propose or refine hypotheses, design or provide feedback on the research methodology or experiments, interpret results, develop theoretical models or conceptual frameworks, assist with translation, or clean or reformat datasets. Qualitative and thematic data analysis was not applicable to this work.

We reviewed all AI-assisted work. In particular, AI-assisted mathematical proofs and AI-provided ingredients used in proving mathematical claims were checked and verified by the authors, AI-assisted implementation code was reviewed and tested for correctness, and the synthetic-data generation procedure used for DFlash training was reviewed by the authors. We take responsibility for the final content of this work, including text, claims, proofs, code, data, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

In the paper, we provide detailed descriptions of LOOPSPEC and the experimental setup to facilitate reproduction of our results. The LOOPSPEC algorithm is described in the main text and accompanying with pseudocode (Sections 4 and H). Code and instructions for using LOOPSPEC and reproducing the experiments are available in our anonymous repository https://anonymous.4open.science/r/loopspec/. The repository contains the necessary configurations, dependencies, and scripts.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732. 6

Seongjin Cha, Gyuwan Kim, Dongsu Han, Tao Yang, and Insu Han. KnapSpec: Self-speculative decoding via adaptive layer selection as a knapsack problem. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=k5nKHWp9VC. 1

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318. 1, 2, 3.2

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Oz335dV48X. 1, 1, 2, 6.3, D

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374. 6

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. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168. 4.1, 6

Chunyuan Deng, Yizhe Zhang, Rui-Jie Zhu, Yuanyuan Xu, Jiarui Liu, T. S. Eugene Ng, and Hanjie Chen. LT2: Linear-time looped transformers, 2026. URL https://arxiv.org/abs/2605.20670. 2

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/. 1, 2, 6.3, D, D.2

Tianyu Fu, Yichen You, Zekai Chen, Guohao Dai, Huazhong Yang, and Yu Wang. Think-at-Hard: Dynamic looped transformers for improved reasoning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=eQaJSRZiGn. 2

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836. 6

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-1380. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Paper-Conference.pdf. 1, 2

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf. 4.1, 6

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Hasan Genc, Kurt Keutzer, Amir Gholami, and Yakun Sophia Shao. SPEED: Speculative pipelined execution for efficient decoding. In *Third Workshop on Efficient Natural Language and Speech Processing (ENLSP-III): Towards the Future of Large Language Models and Their Emerging Descendants*, New Orleans, Louisiana, USA, 2023. URL https://neurips2023-enlsp.github.io/papers/paper_17.pdf. 1, 2

Hynek Kydlicek, Alina Lozovskaya, Nathan Habib, and Clémentine Fourrier. Fixing open llm leaderboard with math-verify, 2025. 4.1, 6

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html. 1, 2, 3.2

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35. Curran Associates, Inc., 2022. doi: 10.52202/068431-0278. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/18abbeef8cfe9203fdf9053c9c4fe191-Paper-Conference.pdf. 4.1, 6

Shenggui Li, Chao Wang, YIKAI ZHU, Yubo Wang, Fan Yin, Shuai Shi, Yefei Chen, Xiaomin Dong, Qiaoling Chen, Jin Pan, Ji Li, Yineng Zhang, Lei Yu, Yonggang Wen, Ivor Tsang, and Tianwei Zhang. SpecForge: A flexible and efficient open-source training framework for speculative decoding. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=CQOEbxy0tE. D.1

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-4562. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/c7b5a35ea98b62512a869c19ea7b03cb-Paper-Conference.pdf. 1, 2

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi. 4.1, 6

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatgpt really correct? Rigorous evaluation of large language models for code generation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36. Curran Associates, Inc., 2023. doi: 10.52202/075280-0943. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf. 6

Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Guilherme Penedo, Edward Beeching, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. OpenR1-Math-220k. https://huggingface.co/datasets/open-r1/OpenR1-Math-220k, 2025. D.1

Mathematical Association of America. American invitational mathematics examination (AIME) 2024. https://maa.org/maa-invitational-competitions/, February 2024. 6

Mathematical Association of America. American invitational mathematics examination (AIME) 2025. https://maa.org/maa-invitational-competitions/, February 2025. 6

Sean Michael McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=PXVQTHYwgt. 1, 2, 2, 3.1, 4.1, 6

Nanbeige Lab, Chen Yang, Chengrui Huang, Fufeng Lan, Hanhui Chen, Hao Zhou, Huatong Song, Jiaqi Cao, Jiaying Zhu, Jinlin Niu, Kai Wang, Lisheng Huang, Qiliang Liang, Ran Le, Ruixiang Feng, Shuang Sun, Tao Gu, Tao Zhang, Tianyu Luo, Yang Song, Yun Xing, Yuntao Wen, Ziyao Xu, Zongchao Chen, and Zongqiang Li. Nanbeige4.2-3B: Unlocking agentic capabilities in a compact model, 2026. URL https://arxiv.org/abs/2607.22083. 1, 2

Taekhyun Park, Yongjae Lee, Dohee Kim, and Hyerim Bae. LoopUS: Recasting pretrained llms into looped latent refinement models, 2026. URL https://arxiv.org/abs/2605.11011. 2

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.824. URL https://aclanthology.org/2023.findings-acl.824/. 6

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.emnlp-demos.6/. D.2

Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris Papailiopoulos, and Kangwook Lee. Predictive Pipelined Decoding: A compute-latency trade-off for exact LLM decoding. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=yUmJ483OB0. 2

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In M. Zaharia, G. Joshi, and Y. Lin (eds.), *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/file/dbf02b21d77409a2db30e56866a8ab3a-Paper-Conference.pdf. 6

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & Verify: Lossless large language model acceleration via self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.607. URL https://aclanthology.org/2024.acl-long.607/. 1, 2

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf. 1, 6, B

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models, 2025. URL https://arxiv.org/abs/2510.25741. 1, 2, 2, 3.1, 6

A Experimental Setup Details

Table 4 presents the complete mapping between evaluated Looped Transformer checkpoints and the model names used throughout this paper. Table 5 outlines the benchmark specifications, prompting protocols, and maximum generation token budgets across base and reasoning models. For reasoning models, evaluations apply chat templates with thinking mode enabled to support long chain-of-thought derivations. For code generation on HumanEval+ and MBPP+, we employ zero-shot custom prompts that instruct models to produce self-contained Python scripts within markdown code blocks.

Table 4: Mapping between evaluated checkpoints and the model names used in this paper.

Name in Paper	Hugging Face Checkpoint
Base Models	
Ouro-1.4B	`ByteDance/Ouro-1.4B`
Ouro-2.6B	`ByteDance/Ouro-2.6B`
Raven-Llama-3.2	`smcleish/Recurrent-Llama-3.2-train-recurrence-32`
Raven-OLMo-2-0425	`smcleish/Recurrent-OLMo-2-0425-train-recurrence-32`
Raven-TinyLlama-3T	`smcleish/Recurrent-TinyLlama-3T-train-recurrence-32`
Reasoning Models	
Ouro-1.4B-Thinking	`ByteDance/Ouro-1.4B-Thinking`
Ouro-2.6B-Thinking	`ByteDance/Ouro-2.6B-Thinking`

Table 5: Evaluation benchmark configurations for base and reasoning models.

Benchmark	Prompting Protocol	Base Max Gen Toks	Reasoning Max Gen Toks
GSM8K	3-shot CoT	512	4,096
MATH-500	4-shot	2,048	8,192
BBH	3-shot CoT	1,024	–
HumanEval+	0-shot (Custom Prompt)	1,024	–
MBPP+	0-shot (Custom Prompt)	1,024	–
AIME 2024	0-shot	–	16,384
AIME 2025	0-shot	–	16,384

B Implementation Details of Pipelined Decoding

We implement LoopSpec on top of SGLang (Zheng et al., 2024). In this section, we cover key implementation details as below.

KV Cache Management. The biggest challenge during LoopSpec decoding is correctly maintaining the KV states. Computation at different recurrent depths requires different KV states, even though they share model parameters. Moreover, pruning a rejected branch must preserve the prefix KV states still needed by other branches. We maintain the KV cache for each branch separately at each recurrent depth. A newly created branch reuses its parent's cached prefix by copying the KV indices rather than the KV states and stores new KV separately. After verification, we keep the selected branch and its descendants and release only KV states no longer needed by the remaining branches. For Raven, we also maintain KV caches for $\mathcal{P}$ and $\mathcal{C}$, with separate caches for $\mathcal{C}$ at each proposal and verification depth.

CUDA Graph Execution. Each decoding step involves several GPU kernels, and launching them individually adds overhead to the computation. CUDA Graphs reduce this overhead by recording a sequence of operations and replaying it with updated inputs. However, drafting and pruning change the batch size, while each recorded graph requires fixed input shapes. Since the maximum number of branches can be determined in advance, we prepare graphs for the supported batch sizes separately for $\mathcal{P}$, $\mathcal{B}$, and $\mathcal{C}$. At each step, we select the graph matching the number of branches processed by that component. Each branch's hidden state is stored between steps and loaded into the selected

batch, so changing the batch size does not discard its progress. The graphs also allocate space for new KV entries and update the KV indices before model computation, avoiding separate launches for these cache-management operations.

Depth Configuration. We require d_2 and R to be multiples of d_1, with $d_1 < d_2 < R$. This aligns the proposal and verification stages with the execution schedule defined by d_1 and synchronizes the pre-layers $\mathcal{P}$ and post-layers $\mathcal{C}$ across branches, allowing their computations to be batched.

C Empirical Results on Gating Strategy and Effective Batch Size

Gating reduces the larger parallel computation introduced by second proposals. Each proposal spawns a child branch that continues drafting future tokens, so unconditionally issuing second proposals increases the number of active branches. We examine how often the gate opens and how gating affects the effective batch size on the two Raven models under sampling.

Gate Opening Frequency. We measure how often the gate opens at d_2 to trigger a second proposal across four benchmarks. As shown in Table 6, the opening frequency ranges from 18.75% to 27.10% for `Raven-Llama-3.2` and from 20.11% to 40.40% for `Raven-OLMo-2-0425`. These results show that gating effectively reduces additional branch creation by selectively triggering second proposals.

Table 6: Gate opening frequency under sampling, measured as the percentage of gate evaluations that trigger a second proposal.

Model	GSM8K	MATH-500	HumanEval+	MBPP+
`Raven-Llama-3.2`	27.10%	18.75%	19.66%	20.40%
`Raven-OLMo-2-0425`	24.19%	40.40%	20.11%	20.49%

Effective Batch Size. We define effective batch size as the number of active branches processed by each recurrent forward pass and report its distribution over all recurrent forward passes during evaluation. Figure 4 compares decoding with and without gating on GSM8K. Gating reduces the mean effective batch size from 43.75 to 34.06 for `Raven-Llama-3.2` and from 40.74 to 31.70 for `Raven-OLMo-2-0425`. The distributions also shift toward smaller batch sizes, with observed peaks decreasing from 239 to 176 and from 243 to 191, respectively. These results show that gating reduces both the mean and peak effective batch size during inference.

D LoopSpec Comparison with Other Speculative Decoding Methods

We compare LoopSpec with DFlash(Chen et al., 2026) and LayerSkip(Elhoushi et al., 2024). DFlash represents the training based speculative decoding method with an external draft model and LayerSkip represents training free self-speculative decoding method. All the experiments below are conducted on a single NVIDIA RTX PRO 6000 Blackwell GPU.

D.1 LoopSpec and DFlash

We trained 5-layer DFlash drafters with block size 16 on the `Ouro` thinking models. We used the SpecForge (Li et al., 2026) training framework, and trained on the OpenR1-Math-220k dataset (Lozhkov et al., 2025) with regenerated answers, following the recommended recipe. We trained for 10,000 steps with a batch size of 32 on both `Ouro-1.4B-Thinking` and `Ouro-2.6B-Thinking`. During inference, we use block size of 16 for DFlash, and identical hyperparameters and SGLang configurations for LoopSpec and DFlash for fair comparison.

As shown in Figure 5, LoopSpec consistently outperforms DFlash by showing up to $1.7\times$ speedup on decoding time under both greedy and stochastic sampling settings.

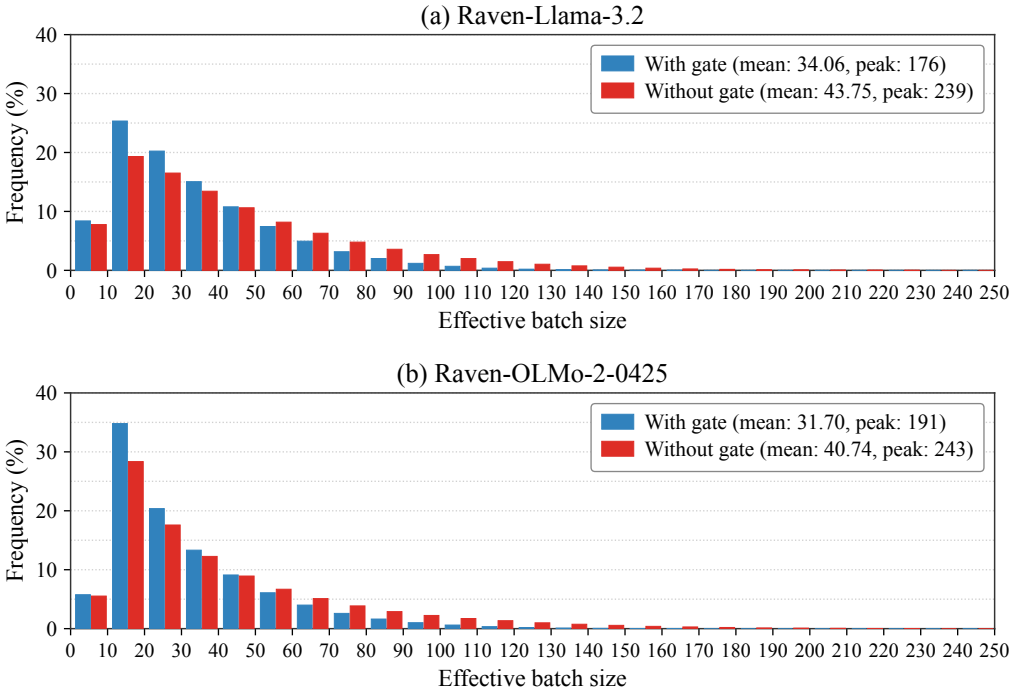


Figure 4: Effective batch size distributions with and without gating on GSM8K under sampling. Effective batch size is the number of active branches processed by each recurrent forward pass. Bars show the percentage of recurrent forward passes in each batch-size bin of width 10. Legends report the mean and peak batch sizes.

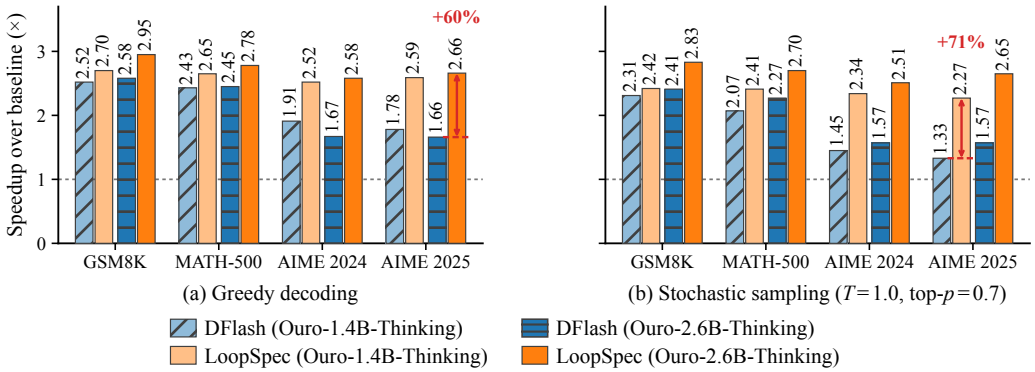


Figure 5: Decoding speedup over the standard autoregressive decoding for LOOPSPEC versus DFlash on the `Ouro` thinking models, under greedy decoding and stochastic sampling.

D.2 LOOPSPEC AND LAYERSKIP

We adapt LayerSkip (Elhoushi et al., 2024) to Looped Transformers by treating recurrent steps as successive layers. It autoregressively drafts a block of tokens by exiting at an intermediate recurrent depth, then jointly verifies them through the remaining recurrent steps, reusing the hidden states and KV cache computed during drafting. Unlike LOOPSPEC, drafting and verification proceed in separate stages.

We implement and compare both methods in the Transformers library (Wolf et al., 2020). We compared the `Ouro` and `Raven` base models under greedy decoding. The models are not finetuned. For LOOPSPEC, we use proposal depths $(d_1, d_2) = (1, 2)$ for `Ouro` and $(2, 8)$ for `Raven`. For LayerSkip, we use a draft length of 12 and an early-exit depth equal to d_1 in LOOPSPEC.

As shown in Table 7, LOOPSPEC consistently outperforms LayerSkip across all evaluated models and benchmarks.

Table 7: Greedy decoding speedup over standard autoregressive decoding for LOOPSPEC and LayerSkip on the Ouro and Raven base models.

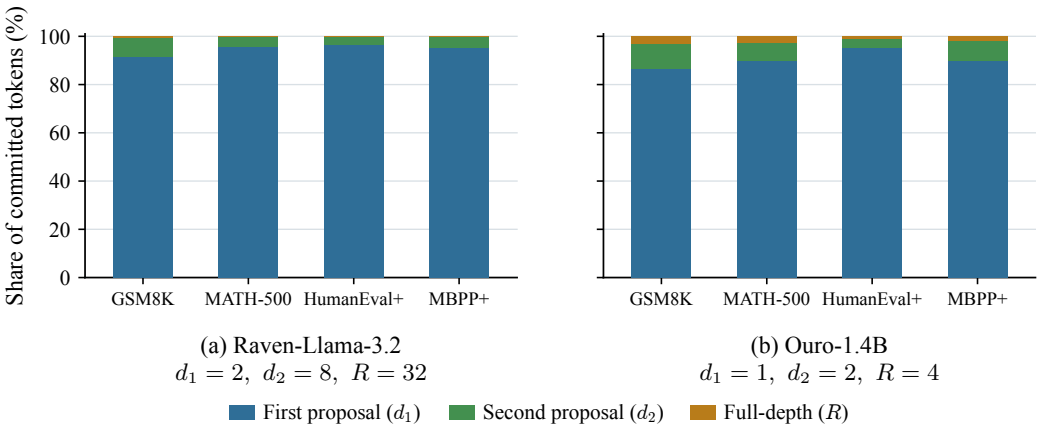
Model	Method	GSM8K	MATH-500	HumanEval+	MBPP+
Ouro-1.4B	LayerSkip	1.68×	1.80×	2.48×	1.80×
	LOOPSPEC	**2.28×**	**2.09×**	**2.55×**	**2.43×**
Ouro-2.6B	LayerSkip	2.07×	2.07×	2.61×	1.96×
	LOOPSPEC	**2.51×**	**2.15×**	**2.66×**	**2.53×**
Raven-Llama	LayerSkip	3.05×	3.59×	4.09×	3.39×
	LOOPSPEC	**4.25×**	**4.36×**	**5.22×**	**4.69×**
Raven-OLMo	LayerSkip	3.54×	3.13×	4.31×	4.25×
	LOOPSPEC	**4.40×**	**3.35×**	**4.84×**	**4.93×**

E TOKEN COMMITMENT SOURCES ACROSS PROPOSAL DEPTHS

We examine how the two proposal depths contribute to token commitment in LOOPSPEC. We partition committed tokens into three categories: accepted first proposals, accepted second proposals after first-proposal rejection, and full-depth fallback when neither proposal is accepted. These categories identify the source of each committed token. Figure 6 shows the share of tokens committed from each source for `Raven-Llama-3.2` with $(d_1, d_2, R) = (2, 8, 32)$ and `Ouro-1.4B` with $(d_1, d_2, R) = (1, 2, 4)$ in greedy decoding scenario for multiple benchmarks.

First Proposals Cover Most Tokens. First proposals account for 91.59%–96.31% of committed tokens for `Raven-Llama-3.2` and 86.57%–95.35% for `Ouro-1.4B` across the four benchmarks. Thus, the shallow proposal supplies most committed tokens in the evaluated settings, allowing the pipeline to retain computation started at the first proposal depth.

Second Proposals Recover Early Rejections. Second proposals contribute a further 3.50%–7.93% of tokens for `Raven-Llama-3.2`, leaving only 0.19%–0.48% to full-depth fallback. For `Ouro-1.4B`, the corresponding shares are 3.74%–10.48% and 0.90%–2.95%. Across both models and all four benchmarks, second proposals therefore recover more tokens than require full-depth fallback. These results support the complementary roles of the two depths: the first proposal enables early drafting, while the second preserves speculative progress for a substantial fraction of first-proposal rejections.



(a) Raven-Llama-3.2
$d_1 = 2,\ d_2 = 8,\ R = 32$

(b) Ouro-1.4B
$d_1 = 1,\ d_2 = 2,\ R = 4$

Figure 6: Share of committed tokens by different proposal sources.

F Empirical Power-Law Decay and Optimal Proposal Depths for Raven-OLMo-2-0425

In this section, we present the `Raven-OLMo-2-0425` sampling ($T = 1.0$, top-$p = 0.7$) total variation (TV) distance to the target on GSM8K and MATH-500 in Figure 7. Similar to Figure 3(b), we obtain the power-law envelope $\phi(r) = \beta r^{-\alpha}$, and it gives $\alpha = 1.08$ and $\beta = 0.207$. With Theorem 2, we can compute the optimal proposal depths $(d_1^*, d_2^*) = (1.41, 9.17)$, which again aligns with the empirical optimal depths $(d_1, d_2) = (2, 8)$ in Section 6.

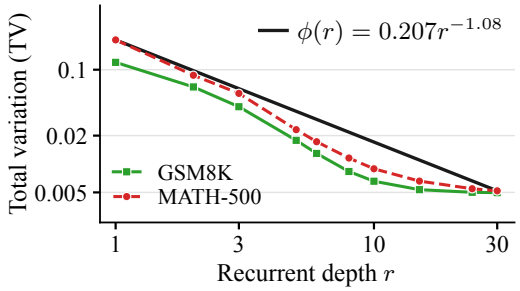


Figure 7: Sampling ($T = 1.0$, top-$p = 0.7$) total variation (TV) distance between intermediate proposals at depth r and the target distribution at depth $R = 32$ for Raven-OLMo-2-0425 on GSM8K and MATH-500. TV is obtained on 10 depths and averaged per token position on the top-p distribution. The power-law envelope $\phi(r)$ is constructed as described in Section 4.1.

G PROOFS

We provide a formal definition of the total variation distance, which is used to evaluate the performance of speculative decoding.

Definition 3 (Total variation distance)**.** *For distributions p and q over a finite sample space $\mathcal{X}$, the total variation distance between p and q is defined as*

$$\mathrm{TV}\left(p, q\right) := \frac{1}{2}\sum_{x\in\mathcal{X}}|p(x) - q(x)| = \sum_{x\in\mathcal{X}}[p-q]_+(x) = \sum_{x\in\mathcal{X}}[q-p]_+(x).$$

In the standard speculative decoding setting on vocabulary $\mathcal{X}$, given the target distribution p and a proposal distribution q, the accepted probability is

$$\begin{aligned}\Pr[\text{accept}] &= \sum_{x\in\mathcal{X}} q(x)\min\left(1, \frac{p(x)}{q(x)}\right)\\ &= \sum_{x\in\mathcal{X}}\min\left(p(x), q(x)\right)\\ &= \sum_{x\in\mathcal{X}} p(x) - [p(x)-q(x)]_+ = 1 - \mathrm{TV}\left(p, q\right).\end{aligned}$$

In other words, $\mathrm{TV}\left(p, q\right)$ measures a probability of rejection.

G.1 PROOF OF THEOREM 1

To prove Theorem 1, we need the following lemma in advance.

Lemma 4. *Let $\nu_1 := p_{d_2} \ominus p_{d_1}$, $\nu_2 := p_R \ominus p_{d_1}$ and define $\varepsilon_1 := \mathrm{TV}\left(p_{d_1}, p_R\right)$, $\varepsilon_2 := \mathrm{TV}\left(p_{d_2}, p_R\right)$. Suppose $\varepsilon_1 > 0$ and $p_{d_1} \neq p_{d_2}$. Then $\mathrm{TV}\left(\nu_1, \nu_2\right) \leq \frac{\varepsilon_2}{\varepsilon_1}$.*

Proof of Lemma 4. We write $u := [p_R - p_{d_1}]_+$, $v := [p_{d_2} - p_{d_1}]_+$ and $\delta := \mathrm{TV}\left(p_{d_1}, p_{d_2}\right)$. By the definition of the residual distribution in Equation (2), $\nu_2 = \frac{u}{\varepsilon_1}$, $\nu_1 = \frac{v}{\delta}$. Consider two cases depending on the relative sizes of δ and ε_1.

Case 1: $\delta \leq \varepsilon_1$.

$$\begin{aligned}\varepsilon_1 \mathrm{TV}\left(\nu_1, \nu_2\right) &= \varepsilon_1 \sum_x \max(0, \nu_2(x) - \nu_1(x))\\ &= \sum_x \max\left(0, u(x) - \frac{\varepsilon_1}{\delta}v(x)\right)\\ &\leq \sum_x \max(0, u(x) - v(x))\\ &\leq \sum_x \max(0, p_R(x) - p_{d_2}(x)) = \varepsilon_2\end{aligned}$$

where the last inequality holds from the fact that $[a]_+ - [b]_+ \leq [a-b]_+$. Similarly,

Case 2: $\delta > \varepsilon_1$.

$$\begin{aligned}\varepsilon_1 \mathrm{TV}\left(\nu_1, \nu_2\right) &= \varepsilon_1 \sum_x \max(0, \nu_1(x) - \nu_2(x))\\ &= \sum_x \max\left(0, \frac{\varepsilon_1}{\delta}v(x) - u(x)\right)\\ &\leq \sum_x \max(0, v(x) - u(x))\\ &\leq \sum_x \max(0, p_{d_2}(x) - p_R(x)) = \varepsilon_2\end{aligned}$$

By combining both cases, $\mathrm{TV}\left(\nu_1, \nu_2\right) \leq \frac{\varepsilon_2}{\varepsilon_1}$. This completes the proof of Lemma 4. □

We are now ready to prove Theorem 1.

Proof of Theorem 1. Recall that the rejection probability from the first proposal is $\varepsilon_1 = \mathrm{TV}\left(p_{d_1}, p_R\right)$ and that from the second one is $\mathrm{TV}\left(p_R \ominus p_{d_1}, p_{d_2} \ominus p_{d_1}\right)$.

If $\varepsilon_1 = 0$, the first proposal is always accepted, so the probability of both candidates are rejected is bounded by $\Pr[\text{reject}_1 \wedge \text{reject}_2] \le \Pr[\text{reject}_1] = \varepsilon_1 = 0 \le \varepsilon_2$. If $p_{d_1} = p_{d_2}$, then $\varepsilon_1 = \varepsilon_2$, so the probability of both candidates are rejected is bounded by $\Pr[\text{reject}_1 \wedge \text{reject}_2] \le \Pr[\text{reject}_1] = \varepsilon_1$.

Without loss of generality, we assume that $\varepsilon_1, \delta > 0$. When the gate is open, Lemma 4 gives

$$\Pr[\text{reject}_1 \wedge \text{reject}_2 \wedge \text{gate open}] \le \varepsilon_1 \cdot \frac{\varepsilon_2}{\varepsilon_1} = \varepsilon_2.$$

The gate in Equation (4) is closed when the primary candidate $\widetilde{x}^{(1)} \sim p_{d_1}$ satisfies $p_{d_2}(\widetilde{x}^{(1)}) \ge p_{d_1}(\widetilde{x}^{(1)})$, and there is no second proposal. Under this condition, the probability of drawing x and rejecting it is $p_{d_1}(x) - \min\left(p_{d_1}(x), p_R(x)\right) = [p_{d_1} - p_R]_+(x)$, and hence

$$\begin{aligned}
\Pr[\text{reject}_1 \wedge \text{gate closed}] &= \sum_{x\,:\,p_{d_2}(x) \ge p_{d_1}(x)} [p_{d_1} - p_R]_+(x) \\
&\le \sum_{x\,:\,p_{d_2}(x) \ge p_{d_1}(x)} [p_{d_2} - p_R]_+(x) \\
&\le \sum_x [p_{d_2} - p_R]_+(x) = \varepsilon_2,
\end{aligned}$$

where the first inequality holds from $p_{d_1}(x) \le p_{d_2}(x)$ under the gate-closed condition. Adding the two bounds gives $2\varepsilon_2$. This completes the proof of Theorem 1. □

G.2 Proof of Theorem 2

Let C be a random variable of the number of recurrent steps to commit a single token. Then,

$$C = \begin{cases} d_1 & \text{if accepted in the first proposal,} \\ d_2 & \text{else if accepted in the second proposal,} \\ R & \text{otherwise.} \end{cases}$$

The expectation of C can be computed as

$$\mathbb{E}[C] = \Pr[\text{accept}_1] d_1 + \Pr[\text{accept}_2] d_2 + (1 - \Pr[\text{accept}_1] - \Pr[\text{accept}_2]) R. \tag{8}$$

By the property of rejection sampling and Theorem 1,

$$\begin{aligned}
&\Pr[\text{accept}_1] = 1 - \mathrm{TV}\left(p_{d_1}, p_R\right) = 1 - \varepsilon_1 \le 1, \\
&\Pr[\text{accept}_2] \le \Pr[\text{reject}_1] = \varepsilon_1, \\
&1 - \Pr[\text{accept}_1] - \Pr[\text{accept}_2] \le 2\varepsilon_2.
\end{aligned}$$

For a prefix X, the assumption in Equation (5) yields

$$\begin{aligned}
\mathbb{E}_X[\varepsilon_1] &= \mathbb{E}_X\left[\mathrm{TV}\left(p_{d_1}, p_R\right)\right] \le \phi(d_1) = \beta d_1^{-\alpha}, \\
\mathbb{E}_X[\varepsilon_2] &= \mathbb{E}_X\left[\mathrm{TV}\left(p_{d_2}, p_R\right)\right] \le \phi(d_2) = \beta d_2^{-\alpha}.
\end{aligned}$$

Substituting these bounds into Equation (8) and taking the expectation over prefix X, we obtain

$$\mathbb{E}_X[C] \le d_1 + \beta d_1^{-\alpha} d_2 + 2\beta d_2^{-\alpha} R. \tag{9}$$

Our goal is to find optimal d_1 and d_2 that minimize the right-hand side in Equation (9). Denote the right-hand side by $f(d_1, d_2)$. The stationary conditions are obtained by taking derivative with respect to d_1 and d_2, respectively,

$$\begin{aligned}
\frac{\partial f}{\partial d_1} &= 1 - \alpha\beta d_1^{-\alpha-1} d_2 = 0, \\
\frac{\partial f}{\partial d_2} &= \beta d_1^{-\alpha} - 2\alpha\beta R d_2^{-\alpha-1} = 0.
\end{aligned}$$

Re-writing these conditions yields

$$d_1^* = (\alpha\beta)^{\frac{\alpha+1}{\alpha^2+\alpha+1}} (2\alpha R)^{\frac{1}{\alpha^2+\alpha+1}},$$
$$d_2^* = (\alpha\beta)^{\frac{\alpha}{\alpha^2+\alpha+1}} (2\alpha R)^{\frac{\alpha+1}{\alpha^2+\alpha+1}}.$$

To verify above (d_1^*, d_2^*) is the minimizer, we compute the Hessian of f as

$$H(d_1, d_2) = \begin{bmatrix} \alpha(\alpha+1)\beta d_1^{-\alpha-2} d_2 & -\alpha\beta d_1^{-\alpha-1} \\ -\alpha\beta d_1^{-\alpha-1} & 2\alpha(\alpha+1)\beta R d_2^{-\alpha-2} \end{bmatrix}$$

and show that the determinant of $H(d_1^*, d_2^*)$ is strictly positive. Using the fact that $R(d_2^*)^{-\alpha-1} = (d_1^*)^{-\alpha}/(2\alpha)$, we have

$$\det\left(H(d_1^*, d_2^*)\right) = \alpha\beta^2 (d_1^*)^{-2\alpha-2} \left(\alpha^2 + \alpha + 1\right) > 0.$$

Since this is the unique stationary point, (d_1^*, d_2^*) is the global minimizer. Finally, the condition

$$R > \frac{1}{2\alpha} \max\{(\alpha\beta)^{1/\alpha}, (\alpha\beta)^{-\alpha-1}, (\alpha\beta)^{1/\alpha} (2\alpha)^{\frac{\alpha^2+\alpha+1}{\alpha^2}}\}$$

ensures the validity of solution, i.e., $1 \leq d_1^* < d_2^* < R$. This completes the proof of Theorem 2.

H LOOPSPEC ALGORITHM PSEUDOCODE

Algorithm 1: LOOPSPEC with two proposals

Input : prompt $x_{1:n}$; pre-layers $\mathcal{P}$; recurrent-layers $\mathcal{B}$; post-layers $\mathcal{C}$; total recurrent depth R; proposal depths $d_1 < d_2 < R$

Global: active branch set $\mathcal{A} \leftarrow \varnothing$; output string $Y \leftarrow \varepsilon$

A branch b has its prefix $X^{(b)}$, current hidden state H_b, recurrent depth r_b, and proposal records $\mathcal{Q}_b$;

Function `Spawn(`X`)`**:**
- $c \leftarrow \text{NewBranch}()$;
- $X^{(c)} \leftarrow X$; $\quad H_c \leftarrow \mathcal{P}(X^{(c)})$;
- $r_c \leftarrow 0$; $\quad \mathcal{Q}_c \leftarrow \varnothing$;
- $\mathcal{A} \leftarrow \mathcal{A} \cup \{c\}$;
- **return** c;

Function `Verify(`b, p`)`**:**
- $\rho \leftarrow p$;
- **for** $j \leftarrow 1$ **to** $|\mathcal{Q}_b|$ **do**
  - $(\widetilde{x}_b^{(j)}, q_j, c_j) \leftarrow \mathcal{Q}_b[j]$;
  - $U_j \sim \mathcal{U}(0,1)$;
  - **if** $U_j < \min\{1, \rho(\widetilde{x}_b^{(j)})/q_j(\widetilde{x}_b^{(j)})\}$ **then**
    - **return** $(\widetilde{x}_b^{(j)}, c_j)$; `// accept`
  - $\rho \leftarrow \rho \ominus q_j$; `// reject`
- $x \sim \rho$;
- **return** $(x, \text{Spawn}(X^{(b)} \,\|\, x))$

$H \leftarrow \mathcal{P}(x_{1:n})$;
for $r \leftarrow 1$ **to** R **do**
- $H \leftarrow \mathcal{B}(H)$;
- **if** $r = d_1$ ***or*** $r = d_2$ **then**
  - $\mathcal{C}(H)$; `// prefill KV cache for depth` d_1 `and` d_2

$x \sim \mathcal{C}(H)$;
$Y \leftarrow Y \,\|\, x$; `// commit first token after prefill`
if *generation has not stopped* **then**
- $b_{\text{init}} \leftarrow \text{Spawn}(x_{1:n} \,\|\, x)$;

while *generation has not stopped* **do**
- $\mathcal{A}_{\text{cur}} \leftarrow \mathcal{A}$;
- **foreach** $b \in \mathcal{A}_{\text{cur}}$ ***in parallel*** **do**
  - $H_b \leftarrow \mathcal{B}(H_b)$; $\quad r_b \leftarrow r_b + 1$;
  - **if** $r_b = d_1$ **;** `// first proposal`
  - **then**
    - $q_1 \leftarrow \mathcal{C}(H_b)$; $\quad \widetilde{x}_b^{(1)} \sim q_1$;
    - $c_1 \leftarrow \text{Spawn}(X^{(b)} \,\|\, \widetilde{x}_b^{(1)})$;
    - $\mathcal{Q}_b[1] \leftarrow (\widetilde{x}_b^{(1)}, q_1, c_1)$;
  - **else if** $r_b = d_2$ **then**
    - $(\widetilde{x}_b^{(1)}, q_1, c_1) \leftarrow \mathcal{Q}_b[1]$; $\quad \widetilde{q}_2 \leftarrow \mathcal{C}(H_b)$;
    - **if** $\widetilde{q}_2(\widetilde{x}_b^{(1)}) < q_1(\widetilde{x}_b^{(1)})$ **;** `// gating mechanism`
    - **then**
      - $q_2 \leftarrow \widetilde{q}_2 \ominus q_1$; $\quad \widetilde{x}_b^{(2)} \sim q_2$ **;** `// residual second proposal`
      - $c_2 \leftarrow \text{Spawn}(X^{(b)} \,\|\, \widetilde{x}_b^{(2)})$;
      - $\mathcal{Q}_b[2] \leftarrow (\widetilde{x}_b^{(2)}, q_2, c_2)$;
- $b_{\max} \leftarrow \arg\max_{b \in \mathcal{A}} r_b$;
- **if** $r_{b_{\max}} = R$ **then**
  - $p \leftarrow \mathcal{C}(H_{b_{\max}})$;
  - $(x^\star, c^\star) \leftarrow \text{Verify}(b_{\max}, p)$;
  - $Y \leftarrow Y \,\|\, x^\star$; `// commit a new decoded token`
  - $\mathcal{A} \leftarrow \text{Reroot}(\mathcal{A}, c^\star)$; `// keep` $c^\star$ `and its descendants`

output Y;