CANNOT WAIT TO VERIFY: PARALLEL DRAFTING TO ELIMINATE VERIFIER IDLE TIME IN SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive inference in large transformer models is limited by token-level sequential execution, resulting in high latency and low GPU utilization. Speculative decoding (SD) mitigates this bottleneck by introducing a draft-and-verifier model. However, the drafting stage remains sequential, leaving the verifier idle and underutilizing compute. To address this, we propose a plug-and-play drafting adaptive drafting method that parallelizes inference across transformer layers during draft generation. Starting from a shared hidden state, our approach evaluates multiple layers in parallel and verifies consistency by measuring divergence in hidden-state space. Layer-wise draft states are accepted when deviations are small; otherwise, a correction step is applied. By exploiting redundancy across transformer layers, our method accelerates draft generation and improves hardware utilization without modifying architectures or requiring retraining. Experimental results on NLP tasks show that our method speeds up the drafting stage by up to $1.65\times$ with minimal degradation in accuracy, while maintaining the same performance as SD. The anonymized source code for our method can be found at `https://anonymous.4open.science/r/PDEVIS-3EDD/README.md`.

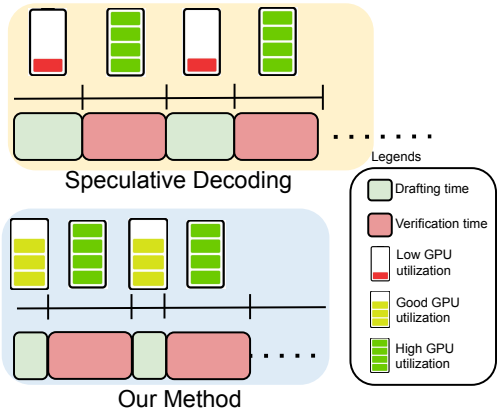
1 INTRODUCTION

Large Language Models (LLMs) (Minaee et al., 2024) have achieved impressive performance across various tasks (reasoning, generation, decision-making, etc.) largely driven by aggressive scaling in depth and parameter count. This scaling has made autoregressive decoding a primary bottleneck in deployment, as each generated token requires a full forward pass through all transformer layers, enforcing strict token-wise sequential execution. This results in high inference latency, limiting the practicality of large models in latency-sensitive settings (Bajpai & Hanawal, 2025a; Cai et al., 2024).

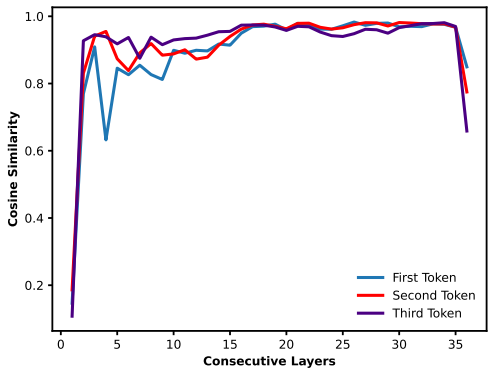
Numerous approaches have been proposed to mitigate inference latency, including pruning (Vadera & Ameen, 2022), quantization (Gholami et al., 2022), distillation (Xu et al., 2024), and early-exit (Rahmath P et al., 2024) mechanisms. While effective in reducing computation, these methods typically trade accuracy for efficiency by altering model architectures, compressing representations, or terminating computation prematurely. In contrast, decoding-time methods such as speculative decoding (SD) (Hu et al., 2025) achieve acceleration without sacrificing output quality by reducing sequential dependence: a lightweight draft model rapidly generates candidate tokens autoregressively, which are then verified in parallel by a larger, more accurate model.

Despite its success, SD suffers from two fundamental inefficiencies that have received limited attention. First, the drafting stage remains fully autoregressive, forcing the verifier to idle until a batch of draft tokens is produced. This leads to substantial underutilization of compute hardware and limits the achievable speedup (see Figure 1a). Second, SD is governed by a trade-off in the drafting stage: a slightly bigger/slower, more accurate draft model improves token acceptance and reduces verification frequency, but incurs higher autoregressive latency during drafting, while a smaller/faster, less accurate drafter lowers per-step drafting cost at the expense of increased rejections and reduced parallel efficiency. Together, these effects make the sequential nature of drafting the dominant bottleneck in SD, impacting the overall system efficiency.

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.



(a) GPU utilization in SD vs our method. SD shows verifier idle time during drafting; our method reduces idle time and improves parallel efficiency.



(b) Layer-wise similarity between consecutive transformer layers in the LLaMA-3B draft model across multiple decoding timesteps.

Prior work that attempts to improve the drafting process relies on weaker or partially executed draft models (Wu et al., 2025), assuming that verification can cheaply correct drafting errors. However, frequent rejections limit end-to-end speedup in such methods. Moreover, existing approaches employ static drafting strategies (Leviathan et al., 2023; Elhoushi et al., 2024), allocating a fixed speedup method regardless of input difficulty. These limitations motivate an adaptive drafting mechanism that dynamically balances speed and accuracy, accelerating draft generation for easy inputs while selectively increasing computation for harder cases to maximize acceptance.

To overcome this limitation, we introduce a *layer-parallel drafting mechanism* that adaptively and efficiently accelerates the draft-generation stage. Instead of executing transformer layers of the draft model strictly sequentially, our method evaluates multiple layers concurrently by propagating from the most recently available hidden state. A lightweight similarity check continuously monitors divergence in hidden-state space and triggers recomputation only when the proposed hidden state deviates substantially from the true hidden state. When divergence remains small, intermediate layers are effectively skipped by reusing hidden representations, allowing the draft model to advance rapidly with minimal additional computation and approximation error. We detail our method in Figure 2.

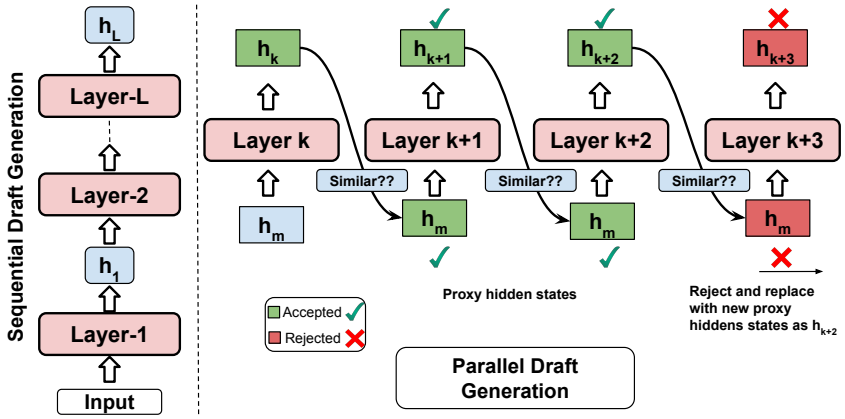


Figure 2: **Vanilla versus Adaptive Layer-parallel Drafting (ALD).** Vanilla drafting executes layers sequentially, leaving parallel compute underutilized. ALD evaluates future layer transformations concurrently using proxy hidden states and adaptively controls their advancement through a similarity check, reducing drafting latency while preserving draft quality.

Our approach is motivated by empirical evidence that transformer models exhibit substantial *layer-wise redundancy* during inference. As seen in Fig. 1b, many inputs do not require the full depth of the network to produce stable representations. By exploiting this redundancy at the layer level, we reduce the *effective sequential depth* of the draft model while retaining the ability to self-correct when necessary. Crucially, the proposed method is *plug-and-play*, requires no architectural modifications

or retraining, and integrates seamlessly with existing speculative decoding pipelines. As a result, it unlocks additional parallelism during inference, improves hardware utilization, and enables latency reductions beyond what standard speculative decoding can achieve, without compromising quality.

Empirical results demonstrate that layer-parallel drafting achieves substantial speedups while fully preserving draft quality, yielding comparable token acceptance rates during verification by the larger model. These features make our method a practical enhancement for real-time, large-scale autoregressive inference. In summary, the key contributions of this paper are as follows:

- We introduce a plug-and-play method that parallelizes transformer layers during speculative drafting, reducing sequential bottlenecks and draft latency.
- We exploit inter-layer redundancy to adaptively reduce sequential depth while preserving draft quality, without retraining or architecture changes.
- Our method improves the efficiency of speculative decoding while maintaining high token acceptance and output quality, thereby enabling efficient autoregressive inference.
- Experimentally, across diverse tasks, we achieve up to $> 1.65\times$ drafter speedup with minimal loss in draft accuracy while preserving SD's end-to-end performance.

2 Related Work

Speculative Decoding (SD) (Stern et al., 2018; Sun et al., 2021; Leviathan et al., 2023; Chen et al., 2023; Li et al., 2024b;c; 2025) has been one of the major advances in faster inference methods for autoregressive models without performance loss. Several prior works have studied ways to improve SD. Given the pivotal role of distributional consistency between the draft and target models, researchers have focused on aligning the draft model with the target model through additional training (Cai et al., 2024; Ankner et al., 2024). Methods that focus on the drafting stage are mainly training the drafters or using self-speculation (Zhang et al., 2024; Elhoushi et al., 2024). Specinfer (Miao et al., 2024) uses boost-trained draft models and tree-structured attention for efficient speculation.

Medusa (Cai et al., 2024) trains a set of extra MLP heads for future token prediction using the original LLM's features. GLIDE and CAPE (Du et al., 2024) reuse the target model's KV-cache, while methods (Hooper et al., 2025; Yang et al., 2023; Monea et al., 2023; Li et al., 2024a; Yi et al., 2024; Liu et al., 2024a; Sun et al., 2024; Svirschevski et al., 2024) such as Self-Speculative Decoding (Zhang et al., 2023; Elhoushi et al., 2024) combine early exits or layer skipping with speculative decoding. LayerSkip (Elhoushi et al., 2024) and Draft & Verify (Zhang et al., 2023) use some initial layers of the same model as a draft. Lookahead (Fu et al., 2024) uses n-gram Jacobi decoding to increase acceptance rates. REST (He et al., 2024) leverages a data storage for retrieval speculation.

Recent works also explore optimized tree-structured attention mechanisms. SpecExec (Svirschevski et al., 2024) takes the most probable continuations from the draft model to build a cache tree, improving inference efficiency on resource-constrained GPUs. OPT-Tree (Wang et al., 2025) proposes an adaptive and scalable draft-tree construction algorithm that maximizes the expected acceptance length. Beyond the drafter itself, other studies investigate stage-level parallelism and theoretical explanations of SD. For instance, PEARL (Liu et al., 2024b) overlaps the drafting and verification stages to achieve stage-level pipelines, while SpecTr (Sun et al., 2023) adopts optimal transport theory to provide a theoretical explanation and optimization. EasySpec (Wu et al., 2025) produces fuzzy speculative drafts in parallel, hypothesizing that drafting was never meant to be precise, thereby leading to lower acceptance rates and limiting efficiency.

In contrast, we introduce a method that challenges the hypothesis that faster draft generation necessarily leads to lower acceptance rates due to reduced draft quality. We achieve this not only by executing draft layers in parallel but also by performing explicit quality checks to ensure that the drafter's overall generation quality is not compromised and that the token acceptance rate remains stable. To the best of our knowledge, we are the first to propose an adaptive drafting mechanism that improves draft model quality without affecting the token acceptance rate.

3 Setup

In this section, we introduce Speculative Decoding (SD) and the distributed inference setup.

3.1 SPECULATIVE DECODING

SD (Leviathan et al., 2023) decomposes decoding into two alternating components: a fast *drafting* step executed by a small draft model and a *verification* step carried out by the target model. This design enables the verification of multiple candidate tokens generated by the draft model in a single iteration. Let $x_{1:t}$ denote the generated prefix at decoding step t. In the drafting step, a lightweight model M_q autoregressively generates a sequence of k speculative tokens $x'_{t+1:t+k} = (x'_{t+1}, \ldots, x'_{t+k})$, together with draft output distributions $\{q_i\}_{i=t+1}^{t+k}$, where $q_i(\cdot) = M_q(\cdot|x_{1:t}, x'_{t+1:t+i-1})$. The draft model is chosen to be computationally efficient, allowing speculative exploration of future tokens at low cost.

The candidate tokens are then verified by the target model M_p in a single forward pass. Conditioning on the concatenated sequence $(x_{1:t}, x'_{t+1:t+k})$, the verifier produces distributions $\{p_i\}_{i=t+1}^{t+k+1}$,, where $p_i(\cdot) = M_p(\cdot|x_{1:t}, x'_{t+1:t+i-1})$, including one additional distribution beyond the drafting horizon.

Each speculative token x'_i is accepted independently according to an importance-sampling-based acceptance rule. Specifically, the acceptance probability is defined as

$$\alpha_i = \min\left(1, \frac{p_i(x'_i)}{q_i(x'_i)}\right).$$

Let $m \leq k$ denote the largest index such that tokens $x'_{t+1}, \ldots, x'_{t+m}$ are consecutively accepted. These tokens are appended to the output sequence, yielding an updated prefix $x_{1:t+m}$.

To correct the $(m+1)^{th}$ rejected token, decoding proceeds by sampling an additional token from a distribution derived from the verifier outputs. If $m < k$, this distribution is constructed by subtracting the draft model's probability mass from the verifier distribution and re-normalizing, otherwise the verifier distribution at the subsequent position is used:

$$p = \begin{cases} \text{norm}(\max(0,\, p_{t+m+1} - q_{t+m+1}))\,, & m < k, \\ p_{t+m+1}, & m = k. \end{cases} \tag{1}$$

By iteratively repeating the drafting and verification steps, speculative decoding reduces the number of expensive autoregressive calls required during inference. The acceptance-rejection mechanism guarantees that the resulting sequence is distributed exactly according to the target model, thereby preserving correctness (Leviathan et al., 2023).

However, this setup assumes a sufficient amount of compute that can perform a parallel verification from a verifier model. This compute remains idle during the draft generation process, waiting for the draft tokens available. This makes the overall compute underutilized in the SD method.

3.2 LAYER SKIPPING

Layer skipping is a widely studied strategy for reducing inference latency in LLMs, most commonly through early-exit mechanisms (Bajpai & Hanawal, 2025b). The core premise is that not all inputs require full-depth computation: when sufficiently informative representations are formed at intermediate layers, further processing may be redundant. By adaptively terminating inference, such methods can substantially reduce average computational cost.

Why Naive Layer Skipping Fails in SD. Although skipping layers can reduce draft computation, naive layer-skipping strategies are ill-suited for accelerating the drafting stage in SD. Decisions based on heuristic confidence signals can be unreliable, particularly for ambiguous or distribution-shifted inputs. Such inaccuracies directly reduce the token acceptance rate during verification, causing speculative tokens to be rejected more frequently and increasing the number of draft-verification iterations, affecting overall speedup.

Need for Acceptance-Aware Skipping. Effective skipping strategies in SD must explicitly account for their downstream impact on verifier acceptance. Skipping decisions should preserve draft model performance, ensuring that speculative tokens remain compatible with the verifier's evaluation. Without such acceptance-aware constraints, aggressive skipping degrades overall system efficiency and speedup despite lowering the per-step computational cost of the draft model.

Motivation for Reliable, Resource-Aware Skipping. Rather than terminating computation early or relying on heuristic confidence thresholds, the draft model should selectively eliminate redundant

computation while explicitly constraining representational deviation. These considerations motivate a principled mechanism that exploits otherwise idle computational resources during the drafting stage. We next discuss a mechanism that accelerates candidate token generations while maintaining high acceptance rates and stable SD dynamics.

4 METHODOLOGY

In this section, we detail our method, where we perform a layer parallel strategy during the drafting process and then verify the generated draft tokens.

4.1 DRAFTING STAGE

Let the draft model M_q consist of L transformer layers $\{f_0, \ldots, f_{L-1}\}$, and let h_0 denote the hidden representation produced by the embedding layer. Under standard execution, the hidden states are computed sequentially as

$$h^*_{k+1} = f_k(h^*_k), \qquad k = 0, \ldots, L-1,$$

where h^*_k denotes the hidden state produced by the exact sequential execution of the draft model.

Our objective is to reduce the sequential depth of this computation by evaluating multiple future layer transformations concurrently. ALD maintains a committed hidden state $\bar{h}_m$ at depth m, where m denotes the deepest layer reached by the current committed execution. Initially, $m = 0$ and $\bar{h}_0 = h_0$.

For a parallel window beginning at depth m, we define its endpoint as

$$K_m = \min(m + K, L),$$

where K is the maximum parallel window size. ALD constructs a shared proxy input for the layers in this window by reusing the current committed representation:

$$\tilde{h}_k = \bar{h}_m, \qquad k = m, \ldots, K_m - 1.$$

The corresponding candidate states are evaluated independently and in parallel:

$$u_{k+1} = f_k(\tilde{h}_k) = f_k(\bar{h}_m), \qquad k = m, \ldots, K_m - 1.$$

The first candidate $u_{m+1} = f_m(\bar{h}_m)$ is an exact one-layer advancement from the current committed state. Candidates farther into the window replace their usual sequential inputs with the shared proxy $\bar{h}_m$, thereby exposing layer-level parallelism.

ALD determines how far the parallel window can be advanced using hidden-state similarity. For each subsequent candidate transition $k \in \{m+1, \ldots, K_m - 1\}$, we compute

$$s_k = \frac{\langle u_{k+1}, \bar{h}_m \rangle}{\|u_{k+1}\|_2 \, \|\bar{h}_m\|_2}.$$

The cosine similarity serves as an online signal for the stability of the proxy execution relative to the current committed representation. A high value indicates that the corresponding layer transformation largely preserves the direction of the committed representation, whereas a lower value indicates a larger representational change. The threshold τ therefore controls the trade-off between parallel advancement and correction frequency: a higher threshold makes ALD more conservative, whereas a lower threshold permits more aggressive parallel execution.

Let the first layer whose proxy candidate fails the similarity criterion be

$$j = \min\{k \in \{m+1, \ldots, K_m - 1\} : s_k < \tau\},$$

when such a layer exists. If the criterion first fails at layer j, the candidate u_{j+1} is rejected, and u_j is retained as the last accepted candidate state. ALD then updates

$$m \leftarrow j, \qquad \bar{h}_m \leftarrow u_j,$$

and constructs a new parallel window from this committed representation. If no rejection occurs in the current window, the window is accepted through its endpoint and

$$m \leftarrow K_m, \qquad \bar{h}_m \leftarrow u_{K_m}.$$

Algorithm 1 Adaptive Layer-parallel Drafting (ALD)

Require: Draft model layers $\{f_k\}_{k=0}^{L-1}$, initial hidden state h_0, similarity threshold τ, maximum parallel window K
Ensure: Final draft hidden state $\bar{h}_L$

Initialize committed depth $m \leftarrow 0$ and committed state $\bar{h}_m \leftarrow h_0$
while $m < L$ **do**
  $K_m \leftarrow \min(m + K, L)$
  Proxy construction:
  for $k = m, \ldots, K_m - 1$ **do**
    $\tilde{h}_k \leftarrow \bar{h}_m$
  end for
  Parallel candidate evaluation:
  for $k = m, \ldots, K_m - 1$ **in parallel do**
    $u_{k+1} \leftarrow f_k(\tilde{h}_k)$
  end for
  Similarity check:
  for $k = m + 1, \ldots, K_m - 1$ **do**
    $s_k \leftarrow \dfrac{\langle u_{k+1}, \bar{h}_m \rangle}{\|u_{k+1}\|_2 \|\bar{h}_m\|_2}$
  end for
  $j \leftarrow \min\{k \in \{m + 1, \ldots, K_m - 1\} : s_k < \tau\}$, if such an index exists
  if j exists **then**
    $m \leftarrow j$; $\bar{h}_m \leftarrow u_j$
  else
    $m \leftarrow K_m$; $\bar{h}_m \leftarrow u_{K_m}$
  end if
end while
return $\bar{h}_L$

The procedure is repeated until $m = L$, after which the final committed state $\bar{h}_L$ is used to obtain the draft-model output distribution for the current decoding step. By evaluating multiple future layer transformations concurrently and adaptively advancing the committed state, ALD reduces the effective sequential depth of draft-model execution while preserving the original transformer architecture and parameters. The method, therefore, trades additional parallel computation for reduced sequential drafting latency, with the similarity criterion determining how aggressively the proxy execution is advanced through the remaining layers.

We refer to the resulting procedure as *Adaptive Layer-parallel Drafting (ALD)*. Because ALD can advance across multiple layers using proxy evaluations, a committed ALD representation may deviate from the representation obtained by sequentially composing the corresponding transformer layers. We next characterize how these local approximation errors propagate to the final hidden representation.

Theorem 1. *Let $0 = m_0 < \cdots < m_T = L$ be the depths committed by ALD, and define $F_r := f_{m_{r+1}-1} \circ \cdots \circ f_{m_r}$ with Lipschitz constant $\Lambda_r := \prod_{k=m_r}^{m_{r+1}-1} L_k$. Let $H_0^* = h_0$, $H_{r+1}^* = F_r(H_r^*)$, and define the local ALD approximation error $\varepsilon_r := \|\bar{h}_{m_{r+1}} - F_r(\bar{h}_{m_r})\|_2$. Then*

$$\|\bar{h}_L - H_T^*\|_2 \le \sum_{r=0}^{T-1} \left(\prod_{q=r+1}^{T-1} \Lambda_q \right) \varepsilon_r, \tag{2}$$

where an empty product equals 1.

Theorem 1 shows that the final deviation depends on the local approximation errors introduced when ALD advances between committed depths and on their propagation through subsequent layer blocks. Smaller proxy deviations contribute less to the final error, whereas errors introduced earlier in the execution can be propagated through more subsequent transformations.

The factors Λ_q characterize the worst-case amplification of a local error by the exact layer compositions between successive committed depths. Although their products can grow in the worst case, their

practical magnitude depends on the local stability of the transformer mappings along the inference trajectory. Residual connections and normalization can mitigate perturbation growth over moderate layer horizons; we therefore interpret the bound as a worst-case characterization of error propagation rather than assuming uniformly small Lipschitz constants.

Empirically, the draft models considered in this work exhibit substantial layer-wise representational similarity, motivating cosine similarity as the online signal used by ALD to identify stable regions for parallel proxy execution. Theorem 1 complements this observation by characterizing how resulting local approximation errors propagate through subsequent layer blocks. In Section 5, we evaluate their practical effect through draft quality, verifier acceptance, and end-to-end decoding speedup.

4.2 Verification Process in Target Model

Once the speculative token sequence is generated by the draft model, it is passed to the target model for verification. We use the standard speculative decoding verification procedure described in Section 3.1. The target model evaluates the speculative sequence in a single forward pass and produces the target distributions for all candidate positions. The speculative tokens are then processed in order using the standard acceptance-rejection rule until the first rejection occurs. If a speculative token is rejected, the next token is sampled from the corrected distribution defined in Section 3.1; if all speculative tokens are accepted, an additional token is sampled from the target model distribution at the subsequent position. Consequently, ALD modifies only the draft-generation procedure, while target-model verification and the standard SD correction mechanism remain unchanged.

4.3 KV-Cache Alignment

KV-cache error accumulation. During layer-parallel drafting, ALD generates speculative tokens, draft distributions, and the associated key-value (KV) states using proxy hidden representations rather than fully sequential layer propagation. While errors in speculative tokens and their output distributions are resolved at the end of each draft–verify iteration through standard speculative verification, approximate KV states can persist across decoding iterations if they are retained in the cache. Reusing such approximate states as conditioning context may cause representation errors to accumulate over decoding steps, potentially degrading draft quality and verifier acceptance.

Bonus calibration for KV-cache recovery. To prevent persistent KV-cache misalignment, we adopt the bonus calibration strategy of Wu et al. (2025). After each draft–verify iteration, the approximate KV entries associated with the speculative execution are discarded. The accepted speculative tokens, together with the bonus token produced by the target model, are then reprocessed by the draft model to reconstruct the KV states required for the next drafting iteration. This restores the draft-model cache before subsequent speculative generation, preventing approximation errors in the KV cache from accumulating across iterations. The calibration introduces an additional draft-model pass, but it can process the accepted segment in parallel and is included in our end-to-end runtime measurements.

5 Experiments

In this section, we present the experimental setup and a detailed analysis of all empirical results.

Datasets. We evaluate our method on a diverse set of benchmarks spanning multiple NLP tasks. For machine translation, we use WMT (Kocmi et al., 2024). General language understanding and reasoning are evaluated on MMLU (Hendrycks et al., 2009). Mathematical reasoning is assessed using MGSM (Shi et al., 2022) and the Math dataset, covering multilingual grade-school problems and more challenging multi-step mathematical reasoning, respectively. Instruction following ability is evaluated using IF-Eval (Zhou et al., 2023). Finally, we use SpecBench (Xia et al., 2024) to evaluate speculative decoding across multiple generation tasks. Together, these benchmarks cover translation, language understanding, mathematical reasoning, instruction following, and SD efficiency.

Metrics. We evaluate efficiency using end-to-end decoding speedup relative to standard autoregressive target-model decoding, draft-generation latency d (seconds per 100 draft tokens), and speculative token acceptance rate. Speedup captures the overall decoding acceleration, while d isolates the drafting cost. The acceptance rate measures draft–target agreement and directly affects speculative

Table 1: Acceptance rate, end-to-end (E2E) speedup, and draft time across datasets. E2E speedup is relative to autoregressive target-model decoding; draft time is in seconds per 100 tokens, with parenthetical values denoting speedup over standard SD.

Dataset	Method	Qwen-4B → 30B			LLaMA-3B → 70B		
		Accept.	E2E Spd.	Draft Time (s/100 tok.)	Accept.	E2E Spd.	Draft Time (s/100 tok.)
WMT	SD	0.40	2.47×	30.2 (1.00×)	0.50	1.91×	48.5 (1.00×)
	EasySpec	0.32	2.72×	19.4 (1.56×)	0.44	2.08×	34.4 (1.41×)
	LayerSkip	0.25	2.54×	18.1 (1.67×)	0.31	1.96×	30.8 (1.57×)
	ALD	**0.38**	**2.95**×	18.7 (1.62×)	**0.46**	**2.29**×	32.5 (1.49×)
MMLU	SD	0.39	2.05×	31.2 (1.00×)	0.55	1.96×	48.9 (1.00×)
	EasySpec	0.31	2.49×	18.8 (1.66×)	0.42	2.24×	33.9 (1.44×)
	LayerSkip	0.23	2.36×	17.5 (1.78×)	0.34	2.09×	31.2 (1.57×)
	ALD	**0.36**	**2.77**×	17.2 (1.81×)	**0.51**	**2.65**×	30.7 (1.59×)
MGSM	SD	0.35	1.85×	30.8 (1.00×)	0.61	2.32×	47.8 (1.00×)
	EasySpec	0.29	2.30×	19.2 (1.60×)	0.56	2.61×	34.5 (1.39×)
	LayerSkip	0.20	2.05×	18.4 (1.67×)	0.41	2.50×	31.8 (1.50×)
	ALD	**0.35**	**2.61**×	19.5 (1.58×)	**0.60**	**2.79**×	34.1 (1.40×)
IF-Eval	SD	0.36	1.92×	31.4 (1.00×)	0.39	1.71×	48.2 (1.00×)
	EasySpec	0.29	2.52×	19.6 (1.60×)	0.32	2.04×	34.8 (1.39×)
	LayerSkip	0.18	2.38×	17.8 (1.76×)	0.28	1.97×	30.4 (1.59×)
	ALD	**0.35**	**2.87**×	19.2 (1.64×)	**0.35**	**2.21**×	30.8 (1.57×)
Math	SD	0.42	2.19×	29.9 (1.00×)	0.53	2.06×	48.5 (1.00×)
	EasySpec	0.35	3.25×	18.9 (1.58×)	0.48	2.45×	34.0 (1.43×)
	LayerSkip	0.28	2.76×	18.0 (1.66×)	0.32	2.38×	31.9 (1.52×)
	ALD	**0.39**	**3.75**×	17.5 (1.71×)	**0.52**	**2.59**×	33.5 (1.45×)

decoding efficiency. Since ALD retains the standard acceptance–rejection mechanism, the final output distribution remains governed by the target model (Leviathan et al., 2023).

Configurations. All experiments are conducted on a system equipped with four NVIDIA RTX A6000 GPUs. To enable layer-parallel draft execution, we use batched inference to process proxy layer evaluations concurrently. The speculative draft length, defined as the number of tokens generated between consecutive verification steps, is fixed to 6 across all experiments. We use a similarity threshold of $\tau = 0.95$ for all models and datasets, motivated by the high layer-wise representational similarity observed during draft-model inference (Figure 1b). We use the same threshold across all experiments rather than tuning it separately for individual datasets. The effect of varying τ on the speed–quality trade-off is analyzed in Appendix A.3 and Figure 3.

For the maximum parallel window size, we set $K = 10$ across all experiments. Here, K determines the maximum number of future layer transformations evaluated concurrently from the current committed representation. We choose $K = 10$ based on the observed range over which ALD can effectively exploit layer-level parallelism before the similarity criterion typically triggers an earlier update of the committed state. Increasing K exposes additional parallelism but yields diminishing returns because candidates farther into the window are more likely to violate the similarity criterion. In our experiments, increasing K beyond 10, including using the full remaining depth, provides only limited additional speedup.

Baselines. We compare ALD against the following representative speculative decoding approaches.

1) Speculative Decoding (SD). Standard SD using an autoregressive draft model to generate candidate tokens, followed by parallel verification with the target model (Leviathan et al., 2023).

2) EasySpec. A training-free layer-parallel speculative decoding method that relaxes sequential dependencies within the draft model to generate approximate draft representations in parallel, followed by KV-cache calibration (Wu et al., 2025). Unlike ALD, EasySpec uses a fixed layer-parallel execution strategy rather than adapting the parallel advancement using online hidden-state similarity.

3) LayerSkip. A self-speculative decoding approach that combines layer skipping and early-exit prediction, using a reduced subset of the target model's layers for drafting and the full model for verification (Elhoushi et al., 2024).

4) ALD (Ours). Our training-free adaptive layer-parallel drafting method evaluates multiple future layer transformations concurrently from the current committed representation and dynamically controls the advancement of the proxy execution using hidden-state similarity.

For all baselines, we use the hyperparameters reported in their official implementations unless otherwise stated. ALD uses the configuration described in Section 5.

5.1 RESULTS

Table 1 and Table 2 compare ALD with the baseline methods across model pairs and tasks. ALD consistently improves end-to-end decoding speedup over standard SD and the compared draft-acceleration baselines, while retaining acceptance rates close to those of standard SD. These results highlight the central trade-off in speculative decoding: reducing drafting latency alone is insufficient if the resulting approximation substantially lowers verifier acceptance.

Comparison with Existing Baselines. EasySpec reduces drafting latency through approximate layer-parallel execution, but its lower acceptance rates can offset part of the resulting draft-time savings. LayerSkip often achieves low drafting latency through more aggressive computation reduction, but this is accompanied by a larger decrease in acceptance. In contrast, ALD uses the online similarity criterion to control the extent of proxy execution, providing substantial draft acceleration while preserving acceptance rates closer to standard SD. For example, on MMLU with Qwen-4B $\rightarrow$ Qwen-30B, ALD achieves a $2.77\times$ end-to-end speedup with an acceptance rate of 0.36, compared with $2.49\times$ and 0.31 for EasySpec, and $2.36\times$ and 0.23 for LayerSkip.

Benefits of Adaptive Layer-Parallel Drafting. The results demonstrate the benefit of adapting the amount of layer-parallel execution rather than applying a fixed approximation uniformly. When the hidden-state similarity remains high, ALD can advance farther through the draft model using parallel proxy evaluations; when the similarity criterion is violated, the committed state is updated earlier. This input-dependent behavior allows ALD to balance drafting latency against the quality of the speculative proposals without requiring retraining or architectural modification.

End-to-End Efficiency and Draft Runtime. ALD does not always have the lowest standalone drafting time, as some baselines use cheaper draft computation. However, its higher acceptance can compensate at the system level. For example, on WMT with Qwen-4B $\rightarrow$ Qwen-30B, LayerSkip has a slightly lower draft time than ALD (18.1 vs. 18.7 seconds per 100 draft tokens), but ALD achieves higher end-to-end speedup ($2.95\times$ vs. $2.54\times$) and acceptance (0.38 vs. 0.25). This highlights that end-to-end SD efficiency depends jointly on draft cost and verifier acceptance.

Overall, ALD provides a favorable balance between draft acceleration and verifier acceptance across the evaluated model pairs and tasks. The results indicate that adaptively controlling layer-parallel proxy execution can improve end-to-end speculative decoding efficiency while limiting the acceptance degradation associated with more aggressive draft approximations.

Limitations. While the proposed method is plug-and-play and does not require retraining, the parallel drafting procedure introduces additional memory and synchronization overhead, as multiple intermediate hidden states must be stored and compared concurrently. Consequently, the benefits of the approach may be reduced on memory-constrained hardware or in low-batch regimes, where these overheads constitute a larger fraction of the total computation. Nevertheless, this trade-off can be mitigated by adjusting the speculative horizon parameter, allowing the method to operate with a smaller parallel window and reduced memory footprint in memory-bound settings.

6 CONCLUSION

In this work, we propose a draft generation method that adaptively speeds up the draft model using the layer-parallel drafting mechanism, which reduces the effective sequential depth of the model without retraining, architectural modifications, or loss of draft fidelity. Our method adaptively exploits this redundancy by skipping computation for easy inputs while retaining the ability to revert to full execution for harder cases, thereby preserving high verifier acceptance rates and stable decoding dynamics. As a plug-and-play enhancement to speculative decoding, the proposed approach unlocks additional parallelism, significantly reduces verifier idle time, and improves overall throughput.

AI USE STATEMENT

In this work, we used generative AI tools solely to assist with writing and language refinement, specifically, improving the clarity, grammar, phrasing, and readability of author-drafted text. All technical content, including the problem formulation, methodology, theoretical results and proofs, experimental design, and analysis, was conceived, developed, and written by the authors. AI tools were not used to generate research ideas, derive or verify proofs, produce experimental results, write or debug code, conduct the literature review, or create any figures, tables, or data. Any such tasks requiring disclosure under the policy are therefore not applicable to this work.

We have reviewed all AI-assisted text. Every passage edited or rephrased with AI assistance was checked by the authors for factual accuracy, correct representation of our claims, and consistency with the underlying technical content, and was revised as needed. We take full responsibility for the final content of this work, including all text, claims, and artifacts.

ETHICS STATEMENT

This work studies inference-time acceleration for large language models and does not involve human subjects, collection of personal or sensitive information, or the introduction of new datasets containing such information. Our experiments use existing publicly available benchmarks and pretrained language models. The proposed method is a general-purpose inference optimization and does not modify the underlying capabilities, training data, or safety mechanisms of the target language model. As with other techniques that reduce the computational cost of large-model inference, improved efficiency may facilitate broader deployment of language models and therefore inherits the potential societal benefits and risks associated with the underlying models and their applications. We encourage deployment in accordance with the licenses, usage policies, safety requirements, and applicable legal requirements of the underlying models and datasets.

REPRODUCIBILITY STATEMENT

We provide the information required to reproduce the proposed method and evaluate its main claims throughout the paper and supplementary material. Section 4.1 formally specifies Adaptive Layer-parallel Drafting (ALD) and provides its algorithm, including proxy construction, parallel layer evaluation, the similarity criterion, and adaptive advancement. Section 5 reports the principal experimental configuration and hyperparameters. Additional implementation and evaluation details are provided in Appendix A.6, including the speculative draft length, similarity threshold, parallel window size, model pairs, baseline configuration, and KV-cache handling procedure. The effect of the similarity threshold is analyzed in Appendix A.3, and task-level SpecBench results and adaptive parallel-depth behavior are reported in Appendix A.2 and Appendix A.4, respectively. The assumptions underlying Theorem 1 are stated explicitly, and a complete proof is provided in Appendix A.1. We will provide the implementation and experimental configuration as anonymized supplementary material to facilitate reproduction of the reported results.

REFERENCES

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. *arXiv preprint arXiv:2402.05109*, 2024.

Divya Jyoti Bajpai and Manjesh Kumar Hanawal. Beyond greedy exits: Improved early exit decisions for risk control and reliability. *arXiv preprint arXiv:2509.23666*, 2025a.

Divya Jyoti Bajpai and Manjesh Kumar Hanawal. A survey of early exit deep neural networks in nlp. *arXiv preprint arXiv:2501.07670*, 2025b.

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

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

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, et al. Glide with a cape: A low-hassle method to accelerate speculative decoding. *arXiv preprint arXiv:2402.02082*, 2024.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

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

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, pp. 291–326. Chapman and Hall/CRC, 2022.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 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)*, pp. 1582–1595, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. *URL https://arxiv. org/abs*, pp. 20, 2009.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Hasan Genc, Kurt Keutzer, Amir Gholami, and Yakun Sophia Shao. Speed: Speculative pipelined execution for efficient decoding. In *Enhancing LLM Performance: Efficacy, Fine-Tuning, and Inference Techniques*, pp. 19–32. Springer, 2025.

Yunhai Hu, Zining Liu, Zhenyuan Dong, Tianfan Peng, Bradley McDanel, and Sai Qian Zhang. Speculative decoding and beyond: An in-depth survey of techniques. *arXiv preprint arXiv:2502.19732*, 2025.

Tom Kocmi, Eleftherios Avramidis, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Markus Freitag, Thamme Gowda, Roman Grundkiewicz, et al. Findings of the wmt24 general machine translation shared task: The llm era is here but mt is not solved yet. In *Proceedings of the Ninth Conference on Machine Translation*, pp. 1–46, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Minghan Li, Xilun Chen, Ari Holtzman, Beidi Chen, Jimmy Lin, Scott Yih, and Victoria Lin. Nearest neighbor speculative decoding for llm generation and attribution. *Advances in Neural Information Processing Systems*, 37:80987–81015, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024c.

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

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*, 2024a.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. Pearl: Parallel speculative decoding with adaptive draft length. *arXiv preprint arXiv:2408.11850*, 2024b.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.

Giovanni Monea, Armand Joulin, and Edouard Grave. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*, 2023.

Haseena Rahmath P, Vishal Srivastava, Kuldeep Chaurasia, Roberto G Pacheco, and Rodrigo S Couto. Early-exit deep neural network-a comprehensive survey. *ACM Computing Surveys*, 57(3):1–37, 2024.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*, 2022.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024.

Xin Sun, Tao Ge, Furu Wei, and Houfeng Wang. Instantaneous grammatical error correction with shallow aggressive decoding. *arXiv preprint arXiv:2106.04970*, 2021.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36:30222–30242, 2023.

Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. Specexec: Massively parallel speculative decoding for interactive llm inference on consumer devices. *Advances in Neural Information Processing Systems*, 37:16342–16368, 2024.

Sunil Vadera and Salem Ameen. Methods for pruning deep neural networks. *Ieee Access*, 10: 63280–63300, 2022.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. Opt-tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*, 13:188–199, 2025.

Yize Wu, Ke Gao, Ling Li, and Yanjun Wu. Easyspec: Layer-parallel speculative decoding for efficient multi-gpu utilization. *arXiv preprint arXiv:2502.02493*, 2025.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *arXiv preprint arXiv:2401.07851*, 2024.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.

Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris Papailiopoulos, and Kangwook Lee. Predictive pipelined decoding: A compute-latency trade-off for exact llm decoding. *arXiv preprint arXiv:2307.05908*, 2023.

Hanling Yi, Feng Lin, Hongbin Li, Ning Peiyang, Xiaotian Yu, and Rong Xiao. Generation meets verification: Accelerating large language model inference with smart parallel auto-correct decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 5285–5299, 2024.

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. *arXiv preprint arXiv:2309.08168*, 2023.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. *arXiv preprint arXiv:2408.15766*, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

A Appendix

A.1 Proof of Theorem 1

Theorem 1. *Let $0 = m_0 < \cdots < m_T = L$ be the depths committed by ALD, and define $F_r := f_{m_{r+1}-1} \circ \cdots \circ f_{m_r}$ with Lipschitz constant $\Lambda_r := \prod_{k=m_r}^{m_{r+1}-1} L_k$. Let $H_0^* = h_0$, $H_{r+1}^* = F_r(H_r^*)$, and define the local ALD approximation error $\varepsilon_r := \|\bar{h}_{m_{r+1}} - F_r(\bar{h}_{m_r})\|_2$. Then*

$$\|\bar{h}_L - H_T^*\|_2 \le \sum_{r=0}^{T-1} \left(\prod_{q=r+1}^{T-1} \Lambda_q \right) \varepsilon_r, \tag{2}$$

where an empty product equals 1.

Proof. Let

$$\delta_r := \left\|\bar{h}_{m_r} - H_r^*\right\|_2 \tag{3}$$

denote the deviation between the hidden state produced by ALD and the corresponding hidden state obtained by exact sequential execution at the r-th committed depth. By construction, both executions start from the same initial hidden state, so

$$\bar{h}_{m_0} = h_0 = H_0^*, \tag{4}$$

which gives

$$\delta_0 = 0. \tag{5}$$

Consider an arbitrary committed block $r \in \{0, \ldots, T-1\}$. By definition,

$$F_r = f_{m_{r+1}-1} \circ \cdots \circ f_{m_r} \tag{6}$$

maps the hidden state at depth m_r to the hidden state at depth m_{r+1} under exact sequential execution. Hence,

$$H_{r+1}^* = F_r(H_r^*). \tag{7}$$

The local approximation error introduced by the corresponding ALD advancement is

$$\varepsilon_r = \left\|\bar{h}_{m_{r+1}} - F_r(\bar{h}_{m_r})\right\|_2. \tag{8}$$

We can therefore write

$$\delta_{r+1} = \left\|\bar{h}_{m_{r+1}} - H_{r+1}^*\right\|_2 \tag{9}$$

$$= \left\|\bar{h}_{m_{r+1}} - F_r(H_r^*)\right\|_2. \tag{10}$$

Adding and subtracting $F_r(\bar{h}_{m_r})$ inside the norm and applying the triangle inequality yields

$$\delta_{r+1} \le \left\|\bar{h}_{m_{r+1}} - F_r(\bar{h}_{m_r})\right\|_2 + \left\|F_r(\bar{h}_{m_r}) - F_r(H_r^*)\right\|_2 \tag{11}$$

$$= \varepsilon_r + \left\|F_r(\bar{h}_{m_r}) - F_r(H_r^*)\right\|_2. \tag{12}$$

It remains to bound the second term. By assumption, each layer mapping f_k is L_k-Lipschitz. Therefore, for any two hidden states a and b,

$$\|f_k(a) - f_k(b)\|_2 \le L_k \|a - b\|_2. \tag{13}$$

Since F_r is the composition of the layers $f_{m_r}, \ldots, f_{m_{r+1}-1}$, a Lipschitz constant for F_r is given by the product of the individual layer-wise constants:

$$\Lambda_r = \prod_{k=m_r}^{m_{r+1}-1} L_k. \tag{14}$$

Consequently,

$$\left\|F_r(\bar{h}_{m_r}) - F_r(H_r^*)\right\|_2 \le \Lambda_r \left\|\bar{h}_{m_r} - H_r^*\right\|_2 = \Lambda_r \delta_r. \tag{15}$$

Combining the above inequalities gives the recursive bound

$$\delta_{r+1} \le \Lambda_r \delta_r + \varepsilon_r. \tag{16}$$

Table 2: Acceptance rate and end-to-end decoding speedup on SpecBench. E2E speedup is measured relative to standard autoregressive decoding with the corresponding target model. The draft models are Qwen-4B and LLaMA-3B for the Qwen-30B and LLaMA-70B target models, respectively.

Task	Method	Qwen-4B → 30B		LLaMA-3B → 70B	
		Accept.	E2E Spd.	Accept.	E2E Spd.
MT	SD	**0.38**	2.15×	**0.47**	1.98×
	EasySpec	0.31	2.52×	0.39	2.18×
	LayerSkip	0.26	2.31×	0.36	2.05×
	ALD	0.35	**2.75**×	0.42	**2.25**×
TL	SD	**0.41**	2.34×	**0.41**	1.84×
	EasySpec	0.34	2.67×	0.34	1.94×
	LayerSkip	0.30	2.45×	0.28	1.98×
	ALD	0.36	**2.91**×	0.40	**2.01**×
SUM	SD	**0.34**	1.97×	**0.42**	1.76×
	EasySpec	0.29	2.13×	0.38	1.91×
	LayerSkip	0.24	2.08×	0.31	1.83×
	ALD	0.33	**2.48**×	0.39	**2.28**×
QA	SD	**0.32**	1.76×	**0.36**	1.68×
	EasySpec	0.31	1.85×	0.27	1.79×
	LayerSkip	0.19	1.86×	0.24	1.71×
	ALD	0.30	**2.09**×	0.32	**2.11**×
MR	SD	**0.53**	3.14×	**0.68**	2.67×
	EasySpec	0.45	3.22×	0.61	2.74×
	LayerSkip	0.35	3.20×	0.52	2.70×
	ALD	0.48	**3.42**×	0.67	**2.97**×
RAG	SD	**0.35**	2.09×	**0.59**	2.17×
	EasySpec	0.30	2.30×	0.54	2.29×
	LayerSkip	0.23	2.16×	0.48	2.26×
	ALD	0.34	**2.51**×	0.56	**2.49**×

We now unroll this recursion. Since $\delta_0 = 0$,

$$\delta_1 \le \varepsilon_0, \tag{17}$$

$$\delta_2 \le \Lambda_1 \delta_1 + \varepsilon_1 \le \Lambda_1 \varepsilon_0 + \varepsilon_1, \tag{18}$$

$$\delta_3 \le \Lambda_2 \delta_2 + \varepsilon_2 \le \Lambda_2 \Lambda_1 \varepsilon_0 + \Lambda_2 \varepsilon_1 + \varepsilon_2. \tag{19}$$

Continuing recursively, after T committed blocks, we obtain

$$\delta_T \le \sum_{r=0}^{T-1} \left(\prod_{q=r+1}^{T-1} \Lambda_q \right) \varepsilon_r. \tag{20}$$

The product is interpreted as 1 when $r = T-1$, corresponding to the fact that the error introduced in the final committed block is not propagated through any subsequent block.

Finally, since $m_T = L$, we have

$$\delta_T = \left\| \bar{h}_L - H_T^* \right\|_2. \tag{21}$$

Therefore,

$$\left\| \bar{h}_L - H_T^* \right\|_2 \le \sum_{r=0}^{T-1} \left(\prod_{q=r+1}^{T-1} \Lambda_q \right) \varepsilon_r, \tag{22}$$

which proves the result. □

A.2 Task-Level Results on SpecBench

Table 2 evaluates ALD across the heterogeneous generation tasks included in SpecBench. Across both model pairs, ALD improves end-to-end decoding speedup over standard SD and the compared

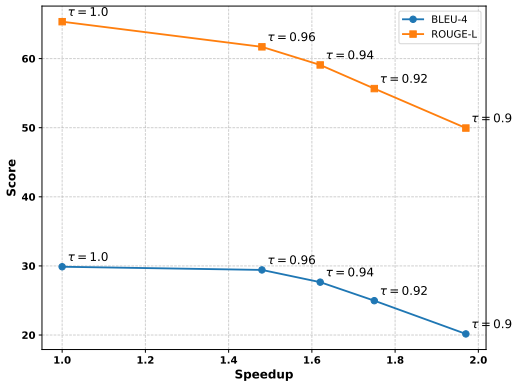


Figure 3: Effect of the similarity threshold τ on drafting speedup and draft-model quality, measured using BLEU-4 and ROUGE-L. Lower τ permits more aggressive layer-parallel proxy execution and increases speedup, while higher τ produces more conservative execution and better preserves draft quality. The default setting $\tau = 0.95$ provides a favorable operating point between these objectives.

draft-acceleration baselines while retaining acceptance rates closer to standard SD than the more aggressive approximations.

The advantage is particularly evident in settings where preserving draft quality is critical to end-to-end efficiency. For example, on mathematical reasoning with LLaMA-3B $\rightarrow$ LLaMA-70B, standard SD obtains an acceptance rate of 0.68 and a $2.67\times$ speedup. EasySpec increases speedup to $2.74\times$ while reducing acceptance to 0.61, whereas ALD retains an acceptance rate of 0.67 and reaches $2.97\times$ speedup. Similarly, on summarization with Qwen-4B $\rightarrow$ Qwen-30B, ALD achieves $2.48\times$ speedup with an acceptance rate of 0.33, compared with $2.13\times$ and 0.29 for EasySpec.

These results illustrate why drafting latency alone is insufficient to characterize the efficiency of speculative decoding. A more aggressive approximation can reduce the cost of producing speculative tokens while simultaneously increasing rejection frequency, thereby reducing the amount of useful work obtained from each target-model verification. ALD instead uses the online similarity criterion to regulate the extent of proxy execution, retaining greater draft–target agreement while exposing substantial layer-level parallelism. The task-level variation motivates adaptive drafting. Different generation tasks induce different representation dynamics and draft–target agreement, so a uniform approximation strategy need not provide the same latency–acceptance trade-off across all inputs. ALD responds to these variations via its online hidden-state similarity criterion, without requiring task-specific retraining or architectural modifications.

A.3 Effect of the Similarity Threshold

Figure 3 studies the effect of the similarity threshold τ, which directly controls how conservatively ALD advances through a layer-parallel window. Increasing τ causes ALD to refresh the committed representation more frequently, reducing effective parallel advancement but keeping the execution closer to standard sequential draft inference. Conversely, decreasing τ allows ALD to advance farther using proxy representations, increasing parallelism at the cost of greater approximation.

The resulting behavior produces a speed–quality trade-off. At high values of τ, ALD behaves conservatively: fewer proxy transitions are retained, the effective sequential depth increases, and the resulting draft representation remains closer to that of the original draft model. As τ decreases, more candidate transitions are retained within each parallel window, reducing the number of sequential drafting stages and increasing drafting speedup. A reduction in draft-model quality accompanies this additional acceleration, as increasingly aggressive proxy execution deviates further from standard sequential propagation.

We use $\tau = 0.95$ as the default operating point throughout the main experiments. Importantly, this value is shared across models and datasets rather than tuned separately for each benchmark. At this operating point, ALD provides substantial drafting acceleration while maintaining draft quality

Table 3: Average reduction in effective sequential layer depth achieved by ALD across datasets. Values are reported as parallelized layer advances relative to the total number of draft-model layers. Larger values indicate that ALD advances farther through the model using layer-parallel proxy execution before refreshing the committed representation.

Model	WMT	MMLU	MGSM	IF-Eval	Math
Qwen-4B	15.2 / 36	18.5 / 36	14.8 / 36	15.4 / 36	16.3 / 36
LLaMA-3B	12.9 / 28	13.6 / 28	10.8 / 28	13.7 / 28	12.0 / 28

and verifier acceptance close to standard SD. The use of a common threshold also reduces tuning overhead and evaluates whether a single operating point can transfer across the considered settings.

The behavior at lower thresholds further motivates the adaptive similarity check. If the threshold is excessively permissive, ALD retains proxy transitions even when the corresponding layer transformation substantially changes the representation. Such aggressive advancement can increase approximation error and degrade the resulting draft distribution. The similarity threshold, therefore, acts as a control parameter that regulates how much layer-level parallelism is exposed before the committed representation is refreshed.

The cosine score should be interpreted as an inexpensive online control signal rather than a direct estimator of the Euclidean approximation error appearing in Theorem 1. The theorem characterizes how local errors arising from ALD propagate through subsequent layer blocks, while Figure 3 empirically demonstrates how varying the similarity criterion governs the practical trade-off between approximation and drafting latency.

A.4 Adaptive Parallel-Depth Behavior

To better understand the adaptive behavior of ALD, Table 3 reports the average reduction in effective sequential depth across datasets. For Qwen-4B, which contains 36 transformer layers, ALD achieves average parallel advances corresponding to 15.2 layers on WMT, 18.5 on MMLU, 14.8 on MGSM, 15.4 on IF-Eval, and 16.3 on Math. For LLaMA-3B, which contains 28 layers, the corresponding values are 12.9, 13.6, 10.8, 13.7, and 12.0 layers.

These results show that ALD substantially reduces the effective sequential depth of the draft model across different tasks. The amount of parallel advancement is input- and task-dependent: representationally stable regions allow ALD to advance farther within a parallel window, whereas larger hidden-state changes trigger earlier updates of the committed representation. This variation is the behavior that the adaptive similarity criterion targets.

For Qwen-4B, the largest average effective-depth reduction occurs on MMLU, where ALD advances across an equivalent of 18.5 out of 36 layers through layer-parallel proxy execution. At the same time, MGSM exhibits a smaller reduction of 14.8 layers. A similar variation is observed for LLaMA-3B, where the average reduction ranges from 10.8 layers on MGSM to 13.7 on IF-Eval. These differences indicate that ALD does not impose a uniform execution pattern across tasks.

Importantly, the values in Table 3 represent reductions in *sequential dependency depth*, rather than transformer layers physically removed from the model. The corresponding layer transformations are evaluated through the proxy execution described in Section 4.1; acceleration arises from evaluating multiple transformations concurrently and reducing the number of sequential stages required to obtain the final draft representation.

A.5 Relation to Training-Based Draft Optimization

A complementary line of work designs draft models explicitly optimized for speculative decoding through additional training, including EAGLE-style approaches such as EAGLE-3. These methods reduce drafting cost by learning lightweight predictors or specialized draft architectures that produce speculative representations or tokens more efficiently than a conventional autoregressive drafter.

Training-based draft optimization operates under different deployment assumptions from ALD. Such approaches require an additional optimization procedure and access to suitable training data, model features, or distillation pipelines. In contrast, ALD operates directly on an existing multi-layer draft model without modifying its parameters or architecture. Its objective is therefore not to learn a new drafter, but to change how the layers of an existing drafter are executed at inference time.

The two directions are complementary. A trained lightweight drafter reduces the intrinsic cost of speculative proposal generation, whereas ALD targets sequential dependencies within an existing multi-layer drafter. The potential benefit of ALD consequently depends on the depth and computational structure of the draft model. When the drafter is already extremely shallow, there is less layer-level parallelism available to exploit. Conversely, for off-the-shelf multi-layer draft models, ALD provides a training-free mechanism for reducing sequential drafting latency.

We focus our main comparisons on methods that operate in the training-free or layer-reduction setting targeted by ALD. Investigating whether adaptive layer-parallel execution can be combined with learned speculative drafters is an interesting direction for future work.

A.6 Additional Implementation and Reproducibility Details

All experiments use a speculative draft length of 6, similarity threshold $\tau = 0.95$, and maximum parallel window $K = 10$, unless otherwise stated. ALD uses the same pretrained draft and target models as standard speculative decoding and requires no parameter updates, auxiliary prediction heads, or task-specific fine-tuning. The similarity check is computed online from the hidden representations produced during draft execution.

For each draft token, ALD begins from the most recently committed hidden representation. The first layer transformation from this state is evaluated exactly, while the remaining transformations within the current window are evaluated concurrently using the committed representation as their shared proxy input. The cosine-similarity criterion determines how far the resulting proxy execution is advanced before the committed representation is refreshed. This procedure is repeated until the final draft model depth is reached.

Following target-model verification, approximate KV states produced during proxy execution are discarded and reconstructed using the bonus-calibration procedure described in Section 4.3. This prevents approximate KV states from persisting across speculative decoding iterations. All reported end-to-end measurements use the same verification and KV-cache handling procedure for the corresponding method throughout an experiment.

Unless otherwise stated, hyperparameters for baseline methods follow their official implementations. We use identical model pairs, speculative draft lengths, datasets, and hardware when comparing methods to minimize differences unrelated to the drafting strategy.

A.7 Sensitivity to the Maximum Parallel Window K

The maximum parallel window K determines the number of future layer transformations that ALD may evaluate concurrently from the current committed representation. In all main experiments, we use $K = 10$. We additionally study the sensitivity of ALD to this choice by varying K while keeping the remaining inference configuration fixed.

Importantly, K specifies an *upper bound* on the available layer parallelism rather than the number of proxy transitions that are necessarily committed. Within each parallel window, the similarity criterion continues to determine how far ALD advances. If the hidden-state similarity falls below the threshold τ, the remaining candidates are discarded and execution resumes from the last accepted representation. Consequently, increasing K exposes additional candidate parallelism without forcing ALD to use increasingly distant proxy states.

Increasing K from 5 to 10 reduces draft time from 20.4 to 18.7 seconds per 100 tokens and improves end-to-end speedup from $2.81\times$ to $2.95\times$. Beyond this point, the gains saturate: using the entire remaining model depth ($K = L - 1$) improves speedup only marginally further to $3.03\times$. At the same time, acceptance decreases gradually from 0.42 at $K = 5$ to 0.34 at $K = L - 1$.

K	Acceptance Rate	E2E Speedup	Draft Time (s/100 tok.)
5	0.42	$2.81\times$	20.4
10	0.38	$2.95\times$	18.7
15	0.36	$2.99\times$	18.2
20	0.35	$3.01\times$	18.0
$L-1$	0.34	$3.03\times$	17.7

Table 4: Sensitivity to the maximum parallel window K. Increasing K exposes more candidate layer transformations to parallel execution. The similarity criterion limits how much of this additional parallelism is actually committed, leading to diminishing speedup gains beyond $K = 10$.

Batch Size	Vanilla Drafting	ALD	Improvement
1	43.5%	**57.4%**	+13.9 pts
8	87.9%	**98.4%**	+10.5 pts

Table 5: GPU utilization during draft generation. ALD increases utilization even at batch size one, indicating that its benefit does not depend exclusively on request-level batching.

This behavior is consistent with the adaptive design of ALD. A parallel window rarely advances through substantially more than approximately ten layers before representational divergence causes the similarity criterion to refresh the committed state. Additional candidates at larger K are therefore increasingly likely to be discarded, yielding limited latency reduction while introducing additional parallel computation.

These results motivate $K = 10$ as the default configuration used throughout the main experiments. It captures most of the available layer-level parallelism while avoiding unnecessary candidate computation, and we use the same value across tasks without per-dataset tuning.

A.8 System-Level Profiling and Overhead Analysis

ALD trades additional parallel computation within the draft model for reduced sequential execution depth. We therefore complement the end-to-end speedups in Section 5 with a detailed characterization of GPU utilization, memory overhead, synchronization cost, and the runtime of individual ALD operations. For this profiling experiment, we use the Qwen-4B drafter on an NVIDIA Tesla T4 GPU and representative prompts spanning translation, summarization, question answering, and reasoning. The T4 provides a substantially more resource-constrained single-GPU environment than the multi-GPU setup used for the main experiments, allowing us to evaluate ALD when computational resources are limited.

A.8.1 GPU Utilization

We first compare GPU utilization during standard sequential drafting and ALD.

At batch size one, standard sequential drafting utilizes 43.5% of the GPU, whereas ALD increases utilization to 57.4%, an absolute improvement of 13.9 percentage points. The improvement therefore does not rely on batching multiple independent requests: ALD exposes parallelism within the transformer-layer execution of a single request. At batch size eight, utilization further increases from 87.9% to 98.4%.

These measurements support the system-level motivation of ALD. Sequential layer execution can leave substantial compute capacity unused during drafting, and layer-parallel execution converts part of this idle capacity into useful computation.

A.8.2 Memory Overhead

ALD does not duplicate model parameters or introduce additional transformer layers. Its additional memory usage consists of transient activations and temporary buffers required by parallel proxy evaluations and the similarity check.

Batch Size	Peak Additional Memory	% of Drafter Weights
1	2.7 MB	0.03%
8	22.0 MB	0.27%

Table 6: Peak additional GPU memory introduced by ALD. The percentage is measured relative to the approximately 8 GB drafter weight footprint.

Stage	Batch 1	Batch 8
Proxy construction	0.33 ms	0.31 ms
Proxy layer evaluation ($K = 10$)	11.8 ms	41.9 ms
Similarity check	2.0 ms	2.3 ms
Vanilla single-layer forward	5.8 ms	15.6 ms

Table 7: Runtime breakdown of ALD operations. Proxy layer evaluation evaluates up to $K = 10$ candidate layer transformations concurrently.

We measure the peak additional GPU memory required for one complete ALD drafting round, including proxy construction, K parallel candidate evaluations, similarity computation, correction, and KV-cache reconstruction.

At batch size one, ALD requires only 2.7 MB of additional peak memory, corresponding to approximately 0.03% of the drafter's weight footprint. At batch size eight, this increases to 22.0 MB, or approximately 0.27%. The overhead therefore remains below 1% of the drafter weight footprint in both evaluated settings.

The additional memory scales with the number of concurrently processed activations and is released after each drafting round. Moreover, ALD modifies only draft execution; target-model verification follows the standard speculative-decoding procedure and incurs no ALD-specific memory overhead.

A.8.3 Synchronization and Runtime Breakdown

ALD introduces one synchronization point per parallel drafting iteration. After the K proxy layer evaluations complete, synchronization is required before computing their similarities and selecting the deepest accepted candidate. Since the proxy evaluations correspond to transformer layers with identical architecture and similar computational workloads, they exhibit limited load imbalance and generally complete at similar times.

We profile the principal components of one ALD iteration in Table 7.

A complete ALD iteration costs approximately 2.4–$2.7\times$ the latency of a single sequential layer transition while evaluating up to $K = 10$ candidate transformations concurrently. The resulting break-even point is therefore reached when an iteration advances through approximately 2–3 layer transitions. Beyond this point, replacing several sequential layer evaluations with one parallel ALD round reduces sequential latency.

The adaptive-depth measurements in Appendix A.4 show that ALD advances substantially farther than this break-even point on average across the evaluated tasks, explaining why the additional bookkeeping and synchronization costs are amortized in practice.

Finally, the KV-cache reconstruction described in Section 4.3 is included in all reported end-to-end measurements. This operation is required to realign the draft cache after approximate execution and has latency comparable to generating a single token in our implementation. Thus, the speedups reported in Section 5 represent *net end-to-end improvements*, including proxy construction, similarity evaluation, synchronization, correction, and KV-cache reconstruction.

Dataset	Acceptance Rate	E2E Speedup
WMT	0.38 ± 0.0000	$2.95 \pm 0.0057\times$
MMLU	0.36 ± 0.0000	$2.77 \pm 0.0037\times$
MGSM	0.35 ± 0.0000	$2.61 \pm 0.0077\times$
Math	0.39 ± 0.0000	$3.75 \pm 0.0087\times$

Table 8: Repeated ALD measurements with a fixed random seed. Values report mean $\pm$ standard deviation over five runs.

Dataset	Acceptance Rate	E2E Speedup
WMT	0.39 ± 0.1062	$2.95 \pm 0.2165\times$
MMLU	0.36 ± 0.0663	$2.76 \pm 0.1559\times$
MGSM	0.35 ± 0.0679	$2.60 \pm 0.0816\times$
Math	0.38 ± 0.0951	$3.77 \pm 0.2538\times$

Table 9: Robustness across random seeds. Values report mean $\pm$ standard deviation over five independent seeds.

A.9 Robustness Across Repeated Runs and Random Seeds

To distinguish runtime variability from sequence-dependent variation, we perform two complementary five-repetition studies: (i) repeated execution using an identical random seed and (ii) independent executions using different random seeds.

A.9.1 Repeated Execution with the Same Seed

At a fixed seed, the acceptance decisions are reproducible, resulting in zero observed standard deviation in acceptance rate. End-to-end latency exhibits only small system-level variation: the standard deviation in measured speedup remains below $0.009\times$ across all four benchmarks. This residual variation reflects ordinary runtime effects such as GPU kernel scheduling rather than instability in the ALD decision procedure.

A.9.2 Variation Across Random Seeds

We next repeat the experiment with independently sampled random seeds.

As expected, variability is larger across seeds because different sampled sequences induce different hidden-state trajectories and therefore different opportunities for adaptive layer-parallel execution. Nevertheless, the mean acceptance rates and end-to-end speedups remain closely aligned with those reported in the main experiments.

Together, these results separate two sources of variation. Repeated execution of the same decoding trajectory is highly reproducible, whereas variation across seeds reflects genuine differences in the generated sequences rather than instability of the ALD execution mechanism.

A.10 Generalization to the Gemma Model Family

The main experiments evaluate ALD using Qwen and LLaMA model pairs. To examine whether the observed layer-parallel behavior extends beyond these two model families, we additionally evaluate ALD using Gemma-3-1B as the draft model and Gemma-3-27B as the target model.

On MMLU, ALD reduces draft time from 8.5 to 5.1 seconds per 100 tokens, corresponding to a $1.67\times$ drafting acceleration. This improves end-to-end speedup from $1.79\times$ under standard SD to $2.05\times$, while acceptance changes from 0.45 to 0.41.

A similar pattern appears on Math. Draft time decreases from 8.2 to 4.7 seconds per 100 tokens ($1.75\times$), and end-to-end speedup increases from $1.93\times$ to $2.15\times$, while acceptance remains close to standard SD (0.50 versus 0.52).

Dataset	Method	Acceptance	E2E Speedup	Draft Time (s/100 tok.)
MMLU	SD	0.45	1.79×	8.5 (1.00×)
	ALD	0.41	**2.05**×	**5.1** (1.67×)
Math	SD	0.52	1.93×	8.2 (1.00×)
	ALD	0.50	**2.15**×	**4.7** (1.75×)

Table 10: Generalization to the Gemma model family using Gemma-3-1B → Gemma-3-27B. Draft-time values in parentheses denote acceleration relative to standard SD drafting.

These results provide additional evidence that the layer-level parallelism exploited by ALD is not specific to the Qwen and LLaMA architectures used in the main experiments. Nevertheless, the magnitude of the benefit depends on the depth and representation dynamics of the particular draft model, as discussed in Appendix A.5.

A.11 Similarity Metric Ablation

ALD uses cosine similarity as an inexpensive online signal for deciding whether proxy layer transitions remain sufficiently stable to be retained. We additionally evaluate mean squared error (MSE) between hidden representations as an alternative similarity criterion.

Both criteria provide a signal of representational deviation, but they differ in an important respect. MSE depends directly on the absolute magnitude of the hidden activations and was empirically more sensitive to changes in representation scale across decoding positions and model configurations. Consequently, a fixed MSE threshold was less stable across the evaluated settings.

Cosine similarity instead measures directional agreement,

$$s(\mathbf{h}, \tilde{\mathbf{h}}) = \frac{\langle \mathbf{h}, \tilde{\mathbf{h}} \rangle}{\|\mathbf{h}\|_2 \|\tilde{\mathbf{h}}\|_2}, \tag{23}$$

and is invariant to the norms of the two representations. Since ALD seeks an inexpensive indicator of whether hidden representations continue to evolve in a similar direction under proxy execution, this normalization makes the criterion less sensitive to changes in activation magnitude. In our experiments, MSE produced qualitatively comparable behavior but required greater sensitivity in threshold selection, whereas cosine similarity provided a more stable operating criterion across decoding steps and model scales.

We emphasize that neither criterion constitutes a correctness certificate for the approximate hidden state. In particular, two representations can have high cosine similarity while differing in fine-grained components relevant to subsequent computation. We therefore interpret cosine similarity as an *online control signal* for adaptive execution rather than as a direct estimator of the Euclidean approximation error ε_r in Theorem 1. When the similarity falls below the threshold, ALD refreshes the committed representation and begins a new parallel window.

This interpretation is consistent with Appendix A.3, where varying τ empirically controls the trade-off between parallel advancement and draft quality, and with Theorem 1, which separately characterizes how local hidden-state approximation errors propagate through subsequent layer blocks.