DLITE: HISTORY-FREE SPECULATIVE DECODING FOR AGENTIC AI

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding has become a mainstream technique for accelerating large language model (LLM) inference by using a lightweight draft model to generate candidate tokens that are subsequently verified by the target model. However, agentic workloads introduce long and continuously evolving contexts and frequent multi-turn interactions. Under such workloads, we observe that existing draft models suffer from up to an 80% degradation in acceptance length, while maintaining their historical KV cache introduces substantial system overhead. In this paper, we identify the growing historical context as a common source of both problems and introduce *DLite*, a history-free speculative decoding framework for agentic AI. *DLite* directly reuses contextual hidden states from the target model and performs local speculative prediction without explicitly maintaining the historical context, transforming an increasingly complex modeling problem into local simple prediction problem. A lightweight sequential head further restores intra-block causal dependencies with low overhead. To effectively train this heterogeneous history-free architecture, we introduce a tailored two-stage training strategy. Experiments on general-purpose, long-context and agentic workloads show that *DLite* achieves $2 \sim 6$ acceptance length and outperforms the state-of-the-art speculative decoding method, DSpark, up to 70%.

1 INTRODUCTION

Large language models (LLMs) have become the foundation of modern AI agents in applications such as coding, web search, and medicine. As agent execution increasingly relies on frequent LLM invocations, inference efficiency has become a critical factor in both system throughput and user experience. Consequently, extensive research Zhou et al. (2024); Li et al. (2024) has explored various techniques to achieve extreme optimization on LLM inference, encompassing algorithm optimization Chen et al. (2026); Cheng et al. (2026), system enhancements Dao et al. (2022); Hong et al. (2024), and hardware advancements Moon et al. (2024); Prabhakar et al. (2024).

Speculative decoding has emerged as a mainstream inference acceleration technique, which has been widely adopted by prominent LLMs (*e.g.*, DeepSeek-V4 Xu et al. (2026)) and deployed in real-world serving systems(*e.g.*, SGLang Zheng et al. (2024) and vLLM Kwon et al. (2023)). A lightweight draft model first generates multiple candidate tokens, which are then verified by the target model in a single forward pass. By accepting multiple tokens at once, speculative decoding can substantially improve inference efficiency without compromising accuracy.

However, agentic workloads differ significantly from conventional conversational inference. As illustrated in Figure 1(a), agent execution typically involves long inputs, short outputs, frequent multi-turn interactions, and continuously growing contexts. Under such workloads, we observe two major limitations of existing speculative decoding methods. **(1) From the perspective of algorithmic performance**, Figure 1(b) shows that the methods that achieve high acceptance lengths on conventional benchmarks can drop to only $1 \sim 2$ accepted tokens on agentic workloads, corresponding to a degradation of more than 80%. Lightweight draft models increasingly struggle to independently model the long historical context required to match the predictive distribution of the target model. **(2) From the perspective of system efficiency**, maintaining the draft model's historical states introduces non-negligible system overhead. Existing speculative drafters must continuously update their KV cache with newly accepted tokens and maintain historical states for subsequent genera-

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.

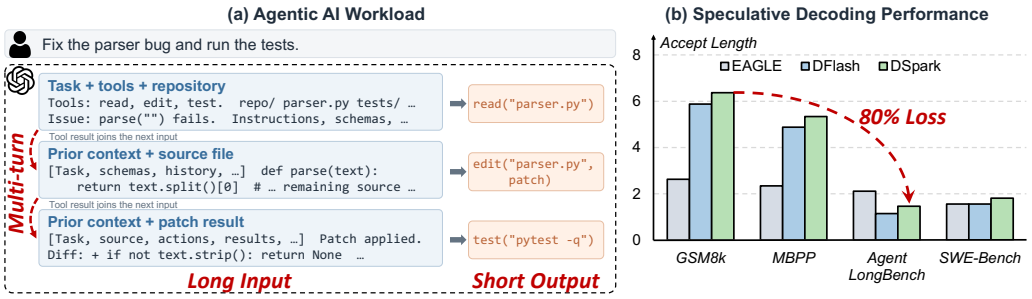


Figure 1: (a) Agentic workloads characteristics. (b) Speculative decoding performance loss.

tion and prefix reuse. Under frequent multi-turn agent execution, these operations can account for $10\% \sim 20\%$ of end-to-end inference latency.

These two limitations share a common source: the draft model is required to independently encode and maintain an increasingly long history, though the target model has processed the same context. More importantly, this growing history continuously increases the explicit context length and positional range faced by the lightweight drafter, making the prediction problem increasingly difficult. We therefore adopt a different design principle: **rather than reconstructing the full history inside the draft model, the drafter directly reuses contextual representations produced by the target model and performs only local speculative prediction around the current decoding position.**

Based on this insight, we propose *DLite*, a history-free speculative decoding framework for agentic AI. *DLite* employs a history-free parallel backbone that directly consumes target hidden states and performs local prediction around the current decoding position without maintaining historical draft KV cache. A lightweight sequential head restores the missing intra-block causal dependency directly in the hidden space, avoiding expensive intermediate vocabulary projections. Finally, we introduce a two-stage training strategy that first bridges the structural mismatch using an isomorphic history-aware teacher and then directly aligns *DLite* with the target model.

We train *DLite* for Qwen3-4B, Qwen3-8B models and evaluate it on both general-purpose, long-context and agentic workloads. *DLite* achieves $2 \sim 6$ acceptance length and improves over DSpark by up to 70%, completely eliminating the draft model's historical KV-cache maintenance overhead.

2 Background

2.1 Speculative Decoding

Speculative decoding accelerates LLM inference through a draft-and-verify pipeline shown in Figure 2(a). Given the current context, a lightweight draft model M_D generates N candidate tokens, which are subsequently verified in parallel by the target model M_T. If τ tokens are produced per verification round, the practical speedup NVIDIA (2026) can be approximately expressed as

$$\text{Speedup} = \frac{\text{Time}_{M_T}(1) \times \tau}{\text{Time}_{M_T}(N) + \text{Time}_{M_D}(N)} \tag{1}$$

Therefore, speculative decoding performance mainly depends on two factors: acceptance length τ and draft-model latency $\text{Time}_{M_D}(N)$.

Existing draft models mainly follow three architectural paradigms, summarized in Figure 2(b)(c)(d). Autoregressive drafters, represented by EAGLE Li et al. (2026), explicitly preserve causal dependencies among speculative tokens but require sequential draft steps. Parallel drafters, such as DFlash Chen et al. (2026), predict all speculative positions in one forward pass, substantially reducing drafting latency but removing intra-block causal dependency. Semi-autoregressive drafters, including DSpark Cheng et al. (2026) and Domino Huang et al. (2026), combine a parallel backbone with a lightweight sequential module to balance drafting latency and prediction quality. Despite their different structures, these designs share one property: the draft model still explicitly depends on the

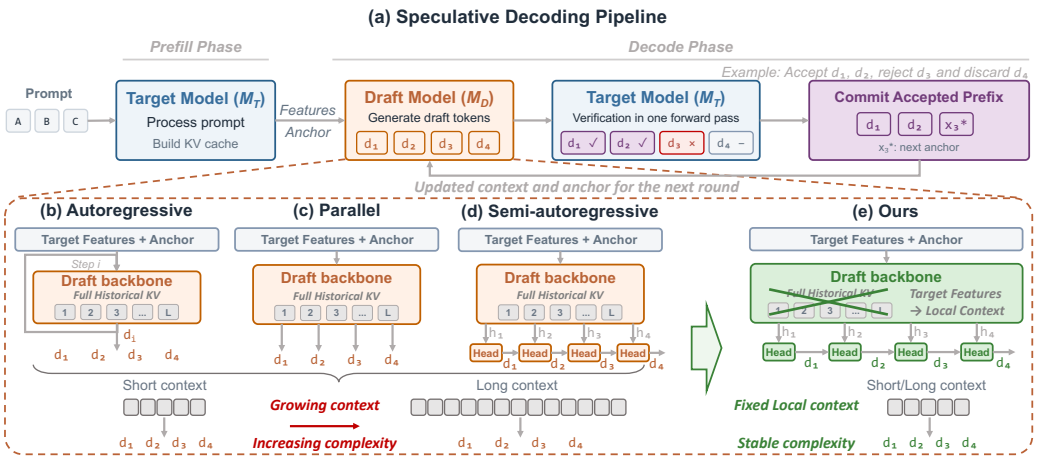


Figure 2: Speculative decoding pipeline and comparison of existing draft architectures with *DLite*.

historical context, whose length grows with decoding. In contrast, Figure 2(e) shows that *DLite* removes this growing historical dependency by directly reusing contextual features produced by the target model and performs local prediction.

2.2 Model Distillation

Training a speculative draft model can be viewed as knowledge distillation from the target model. Different architectures adopt different supervision objectives. EAGLE performs distribution-level distillation using soft-target cross-entropy (CE), which is equivalent to minimizing forward Kullback-Leibler divergence up to a constant. DFlash primarily adopts token-level CE, while DSpark combines CE with total variation (TV) distance to improve distribution overlap between the draft and target models. Domino applies token-level supervision to both parallel and causally corrected predictions through a training curriculum. These differences are particularly relevant to *DLite* because its history-free prediction structure differs substantially from the autoregressive target model, making direct distribution-level distillation non-trivial.

3 Motivation

3.1 Two Important Observations

Agentic workloads expose two limitations of existing speculative decoding methods: degraded acceptance length and increasing historical-state maintenance overhead.

Degraded Acceptance Length. As illustrated in Figure 1(b), existing speculative decoding methods achieve high acceptance lengths on general workloads such as GSM8K and MBPP, but degrade substantially on agentic workloads such as SWE-Bench and AgentLongBench. As the context grows, lightweight draft models increasingly struggle to capture the information required to match the target model's predictions, causing acceptance lengths to drop to only $1 \sim 2$ tokens in some workloads.

KV Cache Maintenance. Existing speculative drafters must maintain historical KV states for subsequent generation. After each target-model verification, accepted tokens are committed to the draft model to update its KV cache, while prefix caching further requires historical draft states to be preserved across agent turns. Under frequent multi-turn workloads with continuously growing contexts, these operations become increasingly expensive. Using SGLang to evaluate the multi-token prediction module in DeepSeek-V4, we observe that the KV cache updating and maintenance can account for $10\% \sim 20\%$ of end-to-end inference latency.

3.2 Analysis and Findings

Although the above problems appear at different levels, they share a common cause: the draft model is required to independently process and maintain the growing historical context shown in Figure 2.

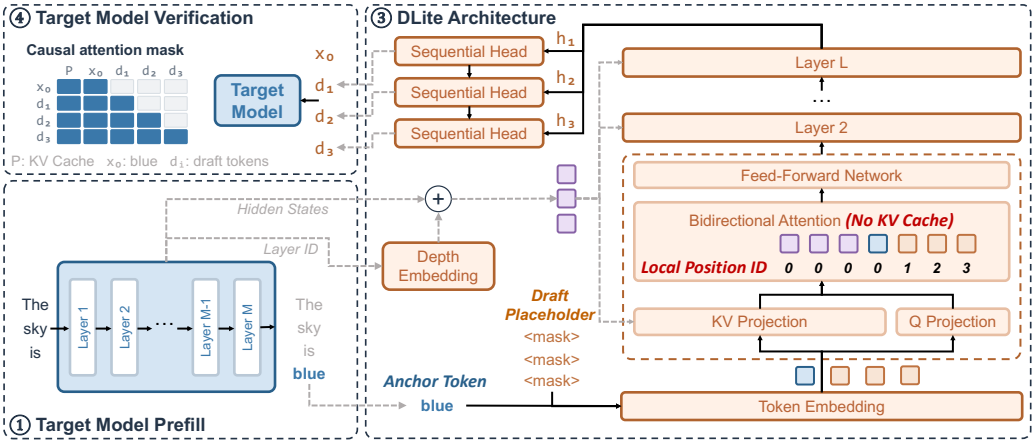


Figure 3: The inference dataflow of *DLite*.

From the model perspective, a lightweight drafter has only a fraction of the target model's capacity, making it increasingly difficult to reproduce the target model's long-context representation as history grows. From the system perspective, preserving the same history requires maintaining and repeatedly updating an increasingly large draft KV cache.

The key inefficiency is therefore that the draft model repeatedly reconstructs historical information that has already been processed by the target model. We instead argue that the drafter should directly reuse the target model's contextual representations and dedicate its limited capacity to modeling the information required for local speculative prediction.

4 *DLite* Framework

Motivated by the observations in Section 3, we design *DLite* to decouple historical-context modeling from speculative generation. As illustrated in Figure 2(e), instead of requiring a lightweight draft model to repeatedly process and maintain an ever-growing history, *DLite* delegates global-context modeling to the target model and performs drafting only over compact target hidden states. This design transforms speculative drafting **from an increasingly complex growing-history modeling problem into a local prediction problem** whose structure and positional range remain independent of the accumulated sequence length. *DLite* realizes this principle through a history-free semi-autoregressive architecture and a two-stage training strategy.

4.1 History-free Semi-autoregressive Architecture

Figure 3 illustrates the inference dataflow of *DLite*. The target model first processes the input context and maintains its original KV cache. For each speculative round, *DLite* takes contextual hidden states from the target model, the current anchor token, and K draft placeholders as input. A history-free parallel backbone produces hidden representations for all speculative positions simultaneously, after which a lightweight sequential head restores the causal dependency among tokens within the speculative block. The resulting candidate tokens are finally verified by the target model following the standard speculative decoding procedure. It is worth noting that *DLite* maintains no historical draft KV cache throughout this process. Each speculative round is independently reconstructed from the latest target representations and the current anchor token, allowing the drafter to perform prediction locally without explicitly processing the accumulated history.

4.1.1 History-free Parallel Backbone

The key idea of the history-free parallel backbone is to **transform speculative drafting from growing-history modeling into local prediction**. To this end, the backbone combines compact target hidden states with explicit local token information and performs local prediction.

Target hidden states. *DLite* directly reuses hidden states produced by the target model as compact contextual features. Since target hidden states and ordinary token embeddings correspond to different input sources, we introduce a learnable depth embedding to distinguish them. When hidden states from multiple target layers are used, the same embedding also identifies their source layers. For a target hidden state $h_t^{(l)}$ extracted from layer l, we construct

$$z_t^{(l)} = h_t^{(l)} + e_{\text{depth}}^{(l)} \tag{2}$$

where $e_{\text{depth}}^{(l)}$ is a learnable source-specific embedding.

Local prediction. Conventional drafters operate over an ever-growing history, such that both the explicit context and global positions increase as decoding proceeds. *DLite* instead delegates long-context modeling to the target model and performs draft prediction only around the current decoding position. The backbone takes the current anchor token and K speculative positions, while using local rather than globally accumulated positions. Historical information is retained through the target hidden states, allowing the drafter to focus on local prediction without explicitly reconstructing the history. *DLite* therefore transforms growing-history modeling into a local prediction problem.

4.1.2 Lightweight Sequential Head

While the parallel backbone efficiently produces representations for multiple speculative positions, parallel computation removes the autoregressive dependency among tokens within the speculative block. *DLite* therefore introduces a lightweight sequential head that **restores only the missing intra-block dependency**, rather than performing another full contextual modeling process.

Let t denote the starting position of a speculative block and T the intra-block offset. The target model predicts x_{t+T} autoregressively according to

$$p_T\left(x_{t+T} \mid x_{<t+T}\right) = p_T\left(x_{t+T} \mid x_{<t}, x_{t:t+T-1}\right) \tag{3}$$

The parallel backbone, however, produces h_T before the tokens at earlier speculative positions are sampled. Since the preceding context $x_{<t}$ has already been processed by the target model and conveyed through its contextual representations into h_T, the information missing from parallel drafting is precisely the intra-block prefix $x_{t:t+T-1}$. This decomposition suggests that the sequential module does not need to reconstruct the complete historical context. It only needs to encode tokens generated earlier within the current speculative block.

To recover this missing dependency efficiently, we employ a low-rank RNN-style sequential head. It recursively compresses previously generated draft tokens into a recurrent state

$$s_T = g_\phi\left(s_{T-1}, E(x_{t+T-1})\right) \tag{4}$$

where $E(\cdot)$ denotes the token embedding. The prediction at speculative position T is modeled as

$$q_{\theta,\phi}\left(x_{t+T} \mid h_T, s_T\right) \approx p_T\left(x_{t+T} \mid x_{<t}, x_{t:t+T-1}\right) \tag{5}$$

Here, h_T supplies the contextual information already encoded by the target model, while s_T summarizes the missing intra-block prefix. Together, they recover the information required for autoregressive prediction without reconstructing the full history inside the draft model. This preserves most contextual computation in the parallel backbone while restricting sequential execution to a lightweight low-rank module.

Furthermore, unlike existing semi-autoregressive methods that first project every backbone representation into full-vocabulary logits and subsequently refine these predictions with an additional sequential module, *DLite* performs sequential modeling directly in the hidden space. The vocabulary projection is applied only after the intra-block dependency has been incorporated.

For hidden dimension D and vocabulary size V, this avoids repeatedly applying an intermediate $D \times V$ projection to every speculative representation before sequential refinement. Such computation becomes particularly expensive under high-concurrency serving, where a large number of speculative hidden states are processed simultaneously.

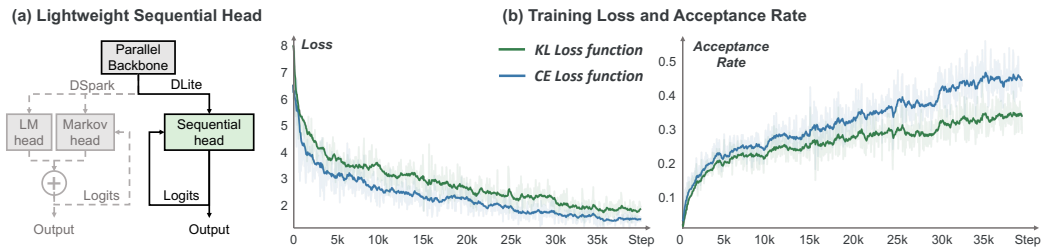


Figure 4: (a) The design architecture of the lightweight sequential head. (b) The training loss and acceptance rate curves of direct target-model distillation using KL and CE loss function.

4.2 Two-stage Training Strategy

The objective of a speculative draft model is to approximate the predictive distribution of the target model, since a larger distribution overlap generally leads to a higher speculative acceptance probability. Distribution-level objectives such as KL divergence are therefore natural choices for distillation. However, Figure 4(b) shows that directly applying KL distillation from the target model to the parallel backbone in *DLite* performs poorly. The two models make predictions under substantially different conditions: the target model performs autoregressive prediction over the complete historical context, whereas the backbone performs parallel prediction using compressed target hidden states, making direct distribution alignment difficult. To bridge this structural gap, we introduce a two-stage training strategy. The warm-up stage first trains *DLite* against an isomorphic history-aware teacher, providing a structurally matched intermediate distillation target. The stable stage subsequently removes this teacher and directly aligns *DLite* with the original target model. Let K denote the speculative block length and V the vocabulary size. At speculative position i, we denote the predictive distributions of the target model, history-aware teacher, and *DLite* as p_i^T, p_i^H, q_i respectively, and denote the ground-truth token as y_i.

Warm-up Stage. In the warm-up stage, we first train an isomorphic history-aware teacher by standard token-level cross-entropy ($-\sum_i w_i \log p_i^H(y_i)$). The teacher shares the same backbone and prediction structure as *DLite*, but retains access to historical KV cache. Once the history-aware teacher is sufficiently trained, we use it as an intermediate distillation target for *DLite* using forward KL divergence:

$$\mathcal{L}_{\text{warmup}} = \sum_{i=1}^{K} w_i D_{\text{KL}}\left(p_i^H \,\|\, q_i\right) = \sum_{i=1}^{K} w_i \sum_{v \in \mathcal{V}} p_i^H(v) \log \frac{p_i^H(v)}{q_i(v)} \tag{6}$$

where $w_i = \exp(\frac{-(i-1)}{\gamma})$ emphasizes the contribution of earlier positions. Because the teacher and *DLite* share the same architecture and prediction interface, their output distributions are naturally aligned despite the difference in historical information available to them.

The history-aware teacher therefore serves as an optimization bridge between the autoregressive target model and the history-free drafter. Instead of forcing *DLite* to directly match a target distribution, the warm-up stage first exposes it to a structurally matched predictive distribution, providing a more stable starting point for subsequent target-model alignment.

Stable Stage. Although the intermediate teacher facilitates optimization, the ultimate objective of *DLite* is to match the predictive distribution of the target model. We therefore discard the history-aware teacher in the stable stage and directly optimize *DLite* against the target model using

$$\mathcal{L}_{\text{stable}} = \lambda_{\text{CE}} \mathcal{L}_{\text{CE}} + \lambda_{\text{TV}} \mathcal{L}_{\text{TV}} \tag{7}$$

where $\mathcal{L}_{\text{CE}}$ denotes token-level cross-entropy and $\mathcal{L}_{\text{TV}} = \sum_i w_i \|p_i^T - q_i\|_1$ measures target–draft distribution discrepancy.

5 Experiments

5.1 Experimental Setup

Models and Baselines. We evaluate *DLite* on Qwen3-4B and Qwen3-8B. We compare against vanilla autoregressive decoding (baseline) and representative speculative decoding methods, includ-

Table 1: Acceptance length τ evaluation on general-purpose workloads.

		Math		Code			Chat	
Model	**Method**	**GSM8K**	**MATH-500**	**HumanEval**	**MBPP**	**LCB**	**MT-Bench**	**Alpaca**
Qwen3-4B	EAGLE3	2.63	2.39	2.47	2.34	2.17	3.07	2.28
	DFlow	5.13	4.92	4.33	4.12	-	3.80	2.90
	DFlash	5.97	5.82	5.25	4.97	4.82	3.39	3.56
	DFlash2	6.14	5.80	5.38	5.14	4.79	**4.14**	2.50
	DSpark	6.27	6.25	5.62	5.48	5.20	3.81	3.54
	DLite	**6.38**	**6.30**	**5.68**	**5.64**	**5.31**	3.94	**3.64**
Qwen3-8B	EAGLE3	2.60	2.28	2.51	2.17	2.08	3.07	2.56
	DFlow	5.10	4.94	4.28	4.08	-	3.58	2.93
	DFlash	6.04	5.91	5.27	4.88	4.89	3.39	3.56
	DSpark	6.36	6.34	5.86	5.36	5.43	3.88	3.64
	DLite	**6.48**	**6.35**	**5.96**	**5.65**	**5.66**	**3.96**	**3.78**

ing EAGLE-3 Li et al. (2026), DFlash Chen et al. (2026), DFlash2 inco.ai (2026), DSpark Cheng et al. (2026), and DFlow Zhang et al. (2026). All serving experiments are conducted on NVIDIA H800 GPUs. The results of DFlash2 and DFlow are from RedHatAI RedHatAI (2026) or paper.

Benchmarks. We evaluate the performance in three different domains, general-purpose tasks, long-context tasks and agentic tasks. General-purpose tasks include three categories, Math: GSM8K Cobbe et al. (2021) and MATH-500 Lightman et al. (2023), Code: HumanEval Chen et al. (2021), MBPP Austin et al. (2021), and LiveCodeBench Jain et al. (2025), Chat: MT-Bench Zheng et al. (2023) and Alpaca Taori et al. (2023). Long-context tasks include Multi-Document QA, Long In-context Learning, Code Repository Understanding, and Long dialogue History in LongBench_v2 Bai et al. (2024). Agentic tasks include AgentMemoryBench Liu et al. (2024), AgentLongBench Fang et al. (2026), BFCL-v3 Yan et al. (2024) and SWE-Bench Jimenez et al. (2024).

Datasets. We use Open-PerfectBlend Xu et al. (2024) which consists of 1.3 million samples containing chat, math, code and instruction-following data. We also collect 600K samples from NVIDIA Nemotron Post-Training Dataset V2 Nathawani et al. (2025) to augment the dataset. We only use the prompts from the dataset and generate the response using the target models.

Implementation Details. For all speculative decoding methods, we use a speculative block size of $K = 7$ and follow the original implementation whenever available. *DLite* uses 12/14 layer target hidden states with sequence-wise concatenation, and a sequential-head rank of $R = 320/512$ for Qwen3-4B/8B. We report average acceptance length τ and end-to-end decoding speedup over vanilla autoregressive decoding.

5.2 Performance Evaluation

General-purpose workloads. Table 1 reports acceptance lengths on conventional reasoning, coding, and conversational benchmarks. *DLite* consistently matches or outperforms existing speculative drafters on both Qwen3-4B and Qwen3-8B. These results show that removing the historical draft KV cache does not sacrifice drafting quality on standard workloads, despite *DLite* operating only on target representations and a local anchor token.

Long-context workloads. As shown in Table 2, the advantage of *DLite* becomes more pronounced on LongBench v2, where it consistently improves both acceptance length and end-to-end speedup over DFlash and DSpark. This trend supports our central design motivation: as the context grows, conventional drafters increasingly rely on their own long-history modeling, whereas *DLite* delegates global-context modeling to the target model and performs only local prediction at the draft side.

Agentic workloads. Table 3 shows the largest relative gains under agentic workloads. In particular, on SWE-Bench, *DLite* substantially improves acceptance length over DFlash and DSpark while achieving around $2\times$ end-to-end speedup. These results indicate that history-free drafting is especially beneficial under the long, evolving contexts characteristic of agent execution

Table 2: Speedup and acceptance length (τ) evaluation on long-context workloads.

Model	Method	LongBench_v2							
		Multi-Doc QA		Long In-ctx Lrn.		Code Repo Ud.		Long-dialog Hist.	
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Qwen3-4B	Eagle3	1.70×	2.10	1.76×	2.28	1.68×	2.12	2.23×	2.88
	DFlash	1.60×	1.90	1.71×	2.00	1.45×	1.83	2.62×	3.04
	DSpark	1.81×	2.20	2.01×	2.41	2.02×	2.44	3.16×	3.93
	DLite	**2.18**×	**2.74**	**2.58**×	**3.11**	**2.66**×	**3.14**	**3.88**×	**4.62**
Qwen3-8B	Eagle3	1.64×	2.05	1.87×	2.33	1.42×	1.80	2.01×	2.56
	DFlash	1.81×	2.10	1.86×	2.28	1.65×	2.16	2.58×	3.07
	DSpark	1.98×	2.52	2.28×	2.67	2.02×	2.59	3.02×	4.01
	DLite	**2.32**×	**2.86**	**2.74**×	**3.22**	**2.53**×	**3.04**	**3.96**×	**4.77**

Table 3: Speedup and acceptance length (τ) evaluation on agentic workloads.

Metric	Method	AgentMemBench		AgentLongBench		BFCL-v3		SWE-bench	
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Qwen3-4B	Eagle3	1.66×	2.09	1.70×	2.12	1.58×	2.00	1.48×	1.88
	DFlash	1.22×	1.65	0.95×	1.14	1.44×	1.80	1.47×	1.72
	DSpark	1.72×	2.10	1.45×	1.84	2.11×	2.54	1.64×	1.89
	DLite	**2.39**×	**2.80**	**2.07**×	**2.51**	**2.37**×	**2.78**	**2.68**×	**3.15**
Qwen3-8B	Eagle3	1.77×	2.19	1.60×	2.00	1.49×	1.88	1.33×	1.71
	DFlash	1.80×	2.20	1.28×	1.65	1.78×	2.07	1.53×	1.83
	DSpark	1.83×	2.26	1.68×	2.09	1.82×	2.23	1.60×	1.94
	DLite	**2.20**×	**2.69**	**2.31**×	**2.79**	**2.30**×	**2.76**	**2.74**×	**3.19**

High-concurrency throughput evaluation. Table 4 evaluates end-to-end throughput under concurrency from 1 to 64. *DLite* consistently achieves higher throughput than DSpark across both model sizes and all evaluated workloads. Notably, the throughput gain remains visible even when their average acceptance lengths are close, showing that the improvement does not solely come from better draft quality. This result is consistent with the lower drafting overhead of our lightweight sequential head, which avoids the intermediate full-vocabulary projection.

5.3 Ablation Study and Design Space Exploration

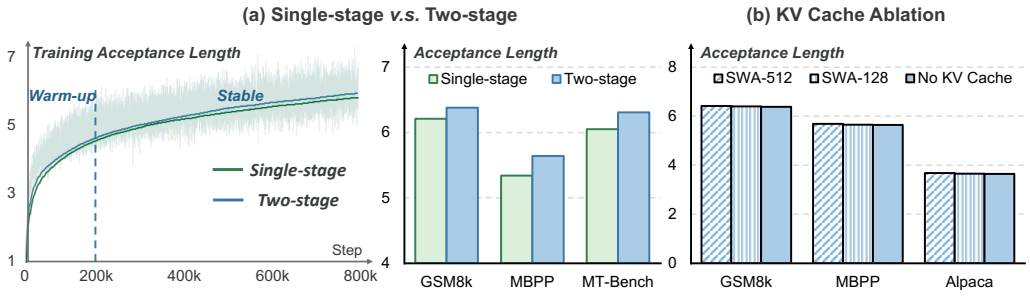


Figure 5: Ablation on training strategy and KV cache.

Training strategy. Figure 5(a) compares our two-stage training strategy with direct single-stage training. The two-stage strategy consistently achieves higher acceptance length throughout the stable stage and improves the final performance. This result supports our motivation in Section 4.2: the history-aware isomorphic teacher provides an effective optimization bridge before direct alignment.

Table 4: High-concurrency throughput evaluation on Qwen3 models.

Task	Method	Concurrency (TPS / Speedup)					Avg. τ
		1	8	16	32	64	
Qwen3-4B							
Alpaca	Baseline	246 / 1.00×	1798 / 1.00×	3377 / 1.00×	5849 / 1.00×	10808 /1.00×	–
	DSpark	609 / 2.47×	4131 / 2.30×	7018 / 2.08×	11118 / 1.90×	13548 / 1.25×	3.49
	DLite	**635 / 2.58×**	**4222 / 2.35×**	**7293 / 2.16×**	**11792 / 2.02×**	**14300 / 1.32×**	**3.63**
MT-Bench	Baseline	245 / 1.00×	1792 / 1.00×	3377 / 1.00×	5883 / 1.00×	10113 / 1.00×	–
	DSpark	620 / 2.53×	4589 / 2.56×	8177 / 2.42×	12565 / 2.14×	15830 / 1.57×	3.79
	DLite	**633 / 2.58×**	**4699 / 2.62×**	**8162 / 2.42×**	**12994 / 2.21×**	**16820 / 1.66×**	**3.88**
LCB	Baseline	242 / 1.00×	1665 / 1.00×	2918 / 1.00×	4498 / 1.00×	6586 / 1.00×	–
	DSpark	905 / 3.74×	5457 / 3.28×	8248 / 2.83×	10243 / 2.28×	11309 / 1.72×	5.28
	DLite	**960 / 3.97×**	**5592 / 3.36×**	**8418 / 2.88×**	**10873 / 2.42×**	**12075 / 1.83×**	**5.98**
Qwen3-8B							
Alpaca	Baseline	142 / 1.00×	993 / 1.00×	1705 / 1.00×	2920 / 1.00×	4330 / 1.00×	–
	DSpark	364 / 2.56×	2260 / 2.28×	3758 / 2.20×	5238 / 1.79×	6827 / 1.58×	3.69
	DLite	**379 / 2.67×**	**2409 / 2.43×**	**3965 / 2.33×**	**5325 / 1.82×**	**7042 / 1.63×**	**3.86**
MT-Bench	Baseline	143 / 1.00×	977 / 1.00×	1749 / 1.00×	2950 / 1.00×	4450 / 1.00×	–
	DSpark	398 / 2.78×	2454 / 2.51×	3956 / 2.26×	5455 / 1.85×	7236 / 1.63×	3.89
	DLite	**407 / 2.85×**	**2531 / 2.59×**	**4131 / 2.36×**	**5558 / 1.88×**	**7305 / 1.64×**	**4.10**
LCB	Baseline	142 / 1.00×	902 / 1.00×	1401 / 1.00×	2196 / 1.00×	2708 / 1.00×	–
	DSpark	511 / 3.60×	2562 / 2.84×	3803 / 2.71×	4815 / 2.19×	5621 / 2.08×	5.47
	DLite	**561 / 3.95×**	**2761 / 3.06×**	**4016 / 2.87×**	**5059 / 2.30×**	**5758 / 2.13×**	**6.04**

Historical KV cache. To examine whether historical draft states are necessary for accurate speculation, we additionally train two variants with sliding-window KV caches of 128 and 512 tokens using the same training data and setup as *DLite*. As shown in Figure 5(b), after sufficient training, the three models achieve comparable acceptance lengths across the evaluated benchmarks. This suggests that historical KV states can facilitate prediction at earlier stages of training, but their advantage diminishes as the drafter learns to exploit contextual information transferred through the target hidden states. With sufficient training, *DLite* can therefore achieve comparable quality without explicitly maintaining KV cache.

Design space exploration. We further explore several architectural design choices of *DLite*, including the number of hidden states extracted from the target model, how multiple target hidden states are composed (feature-wise versus sequence-wise concatenation), and the rank of the lightweight sequential head. These dimensions jointly determine the trade-off between contextual representation capacity, drafting quality, and execution overhead. We use this design-space exploration in Appendix A.2 to determine the final configurations reported in Section 5.1.

6 Conclusion

In this work, we present *DLite*, a history-free speculative decoding framework for agentic AI. *DLite* removes the need for maintaining historical draft KV cache by directly reusing contextual hidden states from the target model and reformulating draft generation as local speculative prediction. On top of the history-free parallel backbone, a lightweight sequential head restores intra-block causal dependencies with low overhead, while a tailored two-stage training strategy addresses the optimization difficulty caused by the heterogeneous prediction structures of the draft and target models. Extensive experiments on general-purpose, long-context, and agentic workloads show that *DLite* achieves $2 \sim 6$ acceptance length and improves over the state-of-the-art semi-autoregressive drafter DSpark by up to 70%, while eliminating historical draft KV-cache maintenance. These results demonstrate that removing redundant historical modeling from the drafter can simultaneously improve speculative decoding quality and system efficiency for emerging agentic workloads.

AI USE STATEMENT

In this work, we used generative AI tools for generating synthetic training data, designing and providing feedback on research methodology and experiments, implementing methods, assisting with translation, and interpreting experimental results. We have not used generative AI tools for cleaning or reformatting datasets, and developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, and supporting qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures, suggesting experimental parameters, creating or editing software code, creating research artifacts, drafting and editing parts of the manuscript, brainstorming, sourcing and searching for information, identifying relevant literature, improving readability, and suggesting the structure of the paper. We have reviewed all AI-assisted work. AI-assisted code and methodological suggestions were manually reviewed and validated by the authors, experimental interpretations were checked against the original results, and literature-related information was verified against the corresponding sources. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde 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, Josh 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, 2021.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Shicheng Fang, Yuxin Wang, XiaoRan Liu, Jiahao Lu, Chuanyuan Tan, Xinchi Chen, Yining Zheng. Xuanjing Huang, and Xipeng Qiu. Agentlongbench: A controllable long benchmark for long-contexts agents via environment rollouts, 2026. URL https://arxiv.org/abs/2601.20730.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. Flashdecoding++: Faster large language model inference with asynchronization, flat gemm optimization, and heuristics. *Proceedings of Machine Learning and Systems*, 6:148–161, 2024.

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding. *arXiv preprint arXiv:2605.29707*, 2026.

inco.ai. Dflash 2: Keep drafting parallel. [Online], 2026. https://inco.ai/blog/dflash2/.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

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

Jinhao Li, Jiaming Xu, Shan Huang, Yonghua Chen, Wen Li, Jun Liu, Yaoxiu Lian, Jiayi Pan, Li Ding, Hao Zhou, et al. Large language model inference acceleration: A comprehensive hardware perspective. *arXiv preprint arXiv:2410.04466*, 2024.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Seungjae Moon, Jung-Hoon Kim, Junsoo Kim, Seongmin Hong, Junseo Cha, Minsu Kim, Sukbin Lim, Gyubin Choi, Dongjin Seo, Jongho Kim, et al. A latency processing unit: A latency-optimized and highly scalable processor for large language model inference. *IEEE Micro*, 44(6): 17–33, 2024.

Dhruv Nathawani, Shuoyang Ding, Vitaly Lavrukhin, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Boris Ginsburg, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v2, August 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v2.

NVIDIA. Co-designing ai models using speculative decoding for faster llm inference. [Online], 2026. https://developer.nvidia.com/blog/co-designing-ai-models-using-speculative-decoding-for-faster-llm-inference/.

Raghu Prabhakar, Ram Sivaramakrishnan, Darshan Gandhi, Yun Du, Mingran Wang, Xiangyu Song, Kejie Zhang, Tianren Gao, Angela Wang, Xiaoyan Li, et al. Sambanova sn40l: Scaling the ai memory wall with dataflow and composition of experts. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1353–1366. IEEE, 2024.

RedHatAI. Qwen3-4b-speculator.dflash2. https://huggingface.co/RedHatAI/Qwen3-4B-speculator.dflash2, 2026.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Tengyu Xu, Eryk Helenowski, Karthik Abinav Sankararaman, Di Jin, Kaiyan Peng, Eric Han, Shaoliang Nie, Chen Zhu, Hejia Zhang, Wenxuan Zhou, et al. The perfect blend: Redefining rlhf with mixture of judges. *arXiv preprint arXiv:2409.20370*, 2024.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, 2024.

Yaojie Zhang, Linfeng Zhang, Bin Cui, and Xupeng Miao. Dflow: Enabling verifier information flow in block diffusion speculative decoding. *arXiv preprint arXiv:2609.06498*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*, 2024.

A Appendix

A.1 Training Details

We train *DLite* on the datasets described in Section 5.1. For all training data, we only use the original prompts and regenerate responses using the corresponding target model, ensuring that the supervision is consistent with the target distribution.

Model configuration. For Qwen3-4B and Qwen3-8B, the *DLite* backbone contains 5 Transformer layers with hidden dimension $2560/4096$. We extract 12/14 hidden states from the target model using sequence-wise composition. Specifically, we always retain the hidden states from the first and last layers, while uniformly sampling the remaining hidden states from the intermediate layers. The rank of lightweight sequential head is $R = 320/512$ for Qwen3-4B/8B, respectively. The speculative block length is set to $K = 7$.

Training data. Our training corpus contains approximately 1.9M samples in total, including 1.3M samples from Open-PerfectBlend and 600K samples from NVIDIA Nemotron Post-Training Dataset V2. We use only the prompts from these datasets and generate responses with the target model. The maximum sequence length is 4096, and each training sample contains 512 speculative blocks / anchors on average.

Optimization. We train *DLite* using AdamW with a learning rate of 4×10^{-4} and gradient clipping threshold 1.0. The global batch size is 32, with gradient accumulation of 1 step. We use cosine learning-rate decay with 0.04 warm-up ratio. Training is performed on 32 NVIDIA H800 GPUs.

Two-stage training. The positional weight decay γ is set to 4. Training consists of a warm-up stage followed by a stable stage. In the warm-up stage, we first train the isomorphic history-aware teacher with the same data mentioned above using token-level cross-entropy for 8 epochs. We then

train *DLite* against this teacher using forward KL divergence for 2 epochs. In the stable stage, the history-aware teacher is removed and *DLite* is directly optimized against the target model using

$$\mathcal{L}_{\text{stable}} = \lambda_{\text{CE}}\mathcal{L}_{\text{CE}} + \lambda_{\text{TV}}\mathcal{L}_{\text{TV}},$$

where $\lambda_{\text{CE}} = 0.1$ and $\lambda_{\text{TV}} = 1$. The stable stage lasts for 6 epochs.

A.2 Design Space exploration

We conduct a design space exploration on Qwen3-8B to study three key architectural choices of *DLite*: (1) the number of target hidden states used as contextual features, (2) the composition strategy for multiple target hidden states, and (3) the rank of the lightweight sequential head. To reduce the cost of design exploration, all variants in this section are trained on the same 80K-sample subset, while keeping the remaining training configurations unchanged. We evaluate the average acceptance length on Alpaca, GSM8K, MBPP, and Multi-Document QA, covering conversational, mathematical, coding, and long-context workloads.

Table 5: Design space exploration of *DLite* on Qwen3-8B. All variants are trained on the same 80K-sample subset, and we report average acceptance length (τ). For the target hidden-state exploration, the first and last target layers are always included in addition to the sampled intermediate layers. Unless otherwise specified, we use 14 target hidden states, sequence-wise hidden-state composition, and a sequential-head rank of 512. Bold indicates the configuration adopted in the final model.

Dimension	Configuration	Alpaca	GSM8K	MBPP	Multi-Doc QA	Avg. τ
Target Hidden States	7	2.34	4.36	3.63	2.22	3.14
	9	2.42	4.40	3.75	2.24	3.20
	14	**2.63**	**4.53**	**3.76**	**2.28**	**3.30**
	20	2.62	4.55	3.74	2.26	3.29
	36	2.58	4.51	3.71	2.26	3.27
Hidden Composition	Feature-wise	2.20	4.28	3.50	2.16	3.04
	Sequence-wise	**2.63**	**4.53**	**3.76**	**2.28**	**3.30**
Sequential-Head Rank	256 ($1/16D$)	2.48	4.40	3.61	2.16	3.16
	512 ($1/8D$)	**2.63**	**4.53**	**3.76**	**2.28**	**3.30**
	1024 ($1/4D$)	2.64	4.55	3.80	2.28	3.32
	2048 ($1/2D$)	2.64	4.53	3.82	2.28	3.32
	4096 (D)	2.64	4.56	3.82	2.30	3.33

Number of target hidden states. Table 5 shows that increasing the number of target hidden states initially leads to clear improvements in acceptance length. The gains, however, quickly saturate: increasing the number from 7 to 14 improves performance consistently across all four workloads, while using 20 or 36 hidden states provides little or no additional benefit. This suggests that a moderate number of intermediate target representations is sufficient to capture complementary contextual information, while further increasing the number of features introduces redundant representations and additional computation.

Hidden-state composition. We further compare two ways of combining hidden states extracted from different target layers. Sequence-wise concatenation treats representations from different layers as separate input elements, while feature-wise concatenation concatenates them along the hidden dimension. Sequence-wise concatenation consistently outperforms feature-wise concatenation across all evaluated workloads, increasing the average acceptance length from 3.04 to 3.30. We therefore adopt sequence-wise composition in *DLite*. This result also supports the use of depth embeddings in Section 4.1, which explicitly identify the source layers of hidden states when they are presented as separate inputs.

Sequential-head rank. Finally, we vary the rank of the lightweight sequential head from 1/16 of the hidden dimension (256) to the full hidden dimension (4096). Increasing the rank from 256 to 512 yields a noticeable improvement in drafting quality while increasing the end-to-end latency by only 0.3%, whereas larger ranks provide only marginal gains while introducing substantial computational overhead. This suggests that the information required by the draft model can be sufficiently resolved in a low-rank space. For example, the average acceptance length increases from 3.16 at rank 256

to 3.30 at rank 512, but only reaches 3.33 with the full 4096-dimensional head. Considering the additional sequential computation introduced by a larger rank and the diminishing performance gains, we choose a relatively small rank to balance drafting quality and execution efficiency.

Implementation details. We evaluate *DLite* on four agent-oriented benchmarks using Qwen3-4B and Qwen3-8B targets. For SWE-Bench, we use 50 prompts from 64K settings (40K–64K input tokens). For AgentLongBench, we use 50 prompts from the 64K setting (55.5K–74.1K input tokens). For MemoryAgentBench, we select 100 FactConsolidation, 50 EventQA, and 30 free-form DetectiveQA questions (66K–118K input tokens), constructing one independent long-context prompt per question. For BFCL, we use 41 sessions from the `multi_turn_long_context` category, run its multi-turn tool-execution harness through a local OpenAI-compatible inference adapter, and aggregate model calls whose input exceeds 32K tokens. BFCL tool-execution time is excluded.