Semantic Retrieval Speculative Decoding with Intermediate Representations

Fangming Zhao[1,†], Xiaofei Yue[2,†], Fulun Ye[1], Yu Peng[1], Ziming Zhao[1,*]
Lewei Jin[3], Wei Wu[4], Tianxiang Chen[4], Tingting Li[1], Jianwei Yin[1]

[1]School of Software Technology, Zhejiang University
[2]School of Computer Science and Technology, Beijing Institute of Technology
[3]College of Computer Science and Technology, Zhejiang University
[4]Ant Group

[†]Equal contribution [*]Corresponding author
zhaoziming@zju.edu.cn

Abstract

Autoregressive decoding remains a key bottleneck for LLM inference, due to its token-by-token nature. Speculative decoding is proposed to reduce latency by drafting multiple tokens and verifying them with fewer target-model passes. However, existing methods either incur auxiliary-drafter overhead or rely on brittle lexical span reuse. In structured tasks (*e.g.*, code generation and text editing), we observe significant locality of reference and semantic redundancy for potential semantic context reuse. In light of this, we present *Semantic Retrieval Speculative Decoding (SRSD)*, a training-free scheme that treats intermediate-layer hidden states as an in-context semantic index. At each decoding step, SRSD retrieves semantically similar past positions. Multiple candidates are merged into a compact tree and verified jointly in a single forward pass. SRSD requires no auxiliary models or external datastores. Across a variety of popular LLMs, SRSD achieves up to 5.42× throughput speedups over state-of-the-art training-free methods and parametric drafters. The code is available at https://github.com/Secbrain/SRSD.

1 Introduction

Large language models (LLMs) such as Llama 3 (Team, 2024) and Qwen 2.5 (Yang et al., 2024) have demonstrated strong capabilities across diverse domains (Yu et al., 2026; Zhao et al., 2025b), yet their generation latency remains dominated by autoregressive decoding (Li et al., 2026; Yue et al., 2026c; Liu et al., 2026; Yue et al., 2026a). Each decoding step depends on all preceding tokens and repeatedly accesses a growing key-value (KV) cache, making inference sequential and memory-bandwidth-bound (Yue et al., 2026b; Ma et al., 2026). This bottleneck is particularly acute in long-context workloads such as code generation, text editing, and document refinement, where thousands of tokens must be produced in a single session.

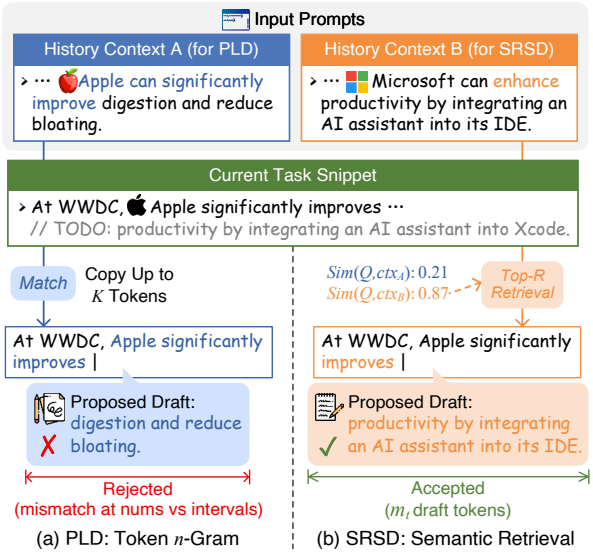

Figure 1: An illustrative comparison between lexical span reuse (*e.g.*, PLD) and SRSD. While lexical matching relies on rigid token n-gram overlap, SRSD exploits semantic similarity in the hidden state space to retrieve relevant continuations even when surface forms diverge (*e.g.*, retrieving "Microsoft" from "Apple").

Speculative decoding mitigates this bottleneck by drafting multiple tokens ahead and verifying them jointly, thereby amortizing target-model forward passes over several accepted tokens (Stern et al., 2018; Xia et al., 2023; Leviathan et al., 2023). Its practical benefit, however, hinges on

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 810–824
October 24-29, 2026 ©2026 Association for Computational Linguistics

drafts being both inexpensive to produce and likely to be accepted by the target model. Existing approaches obtain drafts from three broad sources: (i) *parametric auxiliaries*, including a smaller draft model (Xia et al., 2023) or additional prediction heads (Cai et al., 2024; Li et al., 2024b, 2025), introduce extra parameters that require training, and draft-target mismatch can reduce acceptance; (ii) *lexical span reuse*, exemplified by Prompt Lookup Decoding (PLD) (Saxena, 2023), copies continuations via exact token n-gram matching, but fails when the model produces semantically related yet surface-divergent text; (iii) *dense retrieval methods* (He et al., 2024; Gritta et al., 2025) go beyond exact matching but typically require an external datastore, adding storage overhead and engineering complexity.

In structured tasks such as code generation and text editing, we observe significant *locality of reference*: the model's output frequently echoes earlier context in meaning, even when surface tokens differ. Meanwhile, recent analyses reveal that intermediate Transformer layers encode stable semantic structure, clustering semantically equivalent inputs into nearby regions of hidden-state space regardless of surface form (Skean et al., 2025; Wu et al., 2025). This raises a natural question: *can these intermediate representations serve as an in-context semantic index, enabling the target LLM itself to produce speculative drafts without any auxiliary model or external datastore?*

We introduce *Semantic Retrieval Speculative Decoding (*SRSD*)*, a training-free framework that repurposes the target model's own intermediate-layer hidden states as a lightweight semantic index (Figure 1). At each decoding step, SRSD retrieves the most semantically similar past position via cosine similarity in the normalized hidden-state space, subject to a *first-token alignment* gate that ensures the copied continuation begins with the token already selected by the target model. The resulting primary trunk is augmented with shallow branches from cached logit candidates, and all paths are merged into a compact tree verified jointly in a single forward pass. SRSD operates entirely within the standard inference loop, requiring no auxiliary models, no additional training, and no external datastores, with only a single layer of cached hidden states (approximately 6% memory overhead for Llama3.1-8B). In summary, this paper makes the following contributions:

- We propose SRSD, a KV-cache-compatible, training-free speculative decoding framework that leverages the target LLM's intermediate representations for semantic retrieval and tree-structured verification.
- We identify a robust mid-to-late layer plateau for semantic retrieval and develop a one-time layer-selection procedure that generalizes across tasks.
- Extensive experiments across five models and six tasks show that SRSD consistently outperforms training-free and parametric baselines, achieving up to $5.42\times$ speedup on project-level code generation and $1.73\times$ average greedy speedup on Llama3.1-8B.

2 Related Work

Speculative Decoding and Parametric Drafters. Speculative decoding accelerates autoregressive generation by drafting multiple tokens and verifying them with fewer target-model passes (Stern et al., 2018; Xia et al., 2023; Leviathan et al., 2023; Zhang et al., 2025). Parametric drafters (*e.g.*, Medusa, EAGLE) (Cai et al., 2024; Li et al., 2024b, 2025) introduce additional parameters and typically require training or tuning; draft-target mismatch can reduce acceptance. Several of these methods further adopt tree-structured verification to evaluate multiple draft paths in a single forward pass (Cai et al., 2024; Li et al., 2024b). SRSD similarly employs tree verification, but constructs its candidate tree from retrieval-based proposals rather than a learned drafter.

Training-Free Span Reuse. Training-free methods reuse spans from the prompt or recent prefix via lexical matching (*e.g.*, token n-gram lookup) (Saxena, 2023; Yang et al., 2023; Fu et al., 2024; Lee et al., 2025; Oliaro et al., 2025; Luo et al., 2025; Dumitru et al., 2025). PLD+ (Somasundaram et al., 2025) follows this paradigm but ranks input-occurring spans using model-derived signals (*e.g.*, attention or hidden states). In contrast, SRSD retrieves anchors from the full context history through intermediate-layer semantic similarity, enabling reuse beyond exact surface matching.

External Retrieval for Drafting. Retrieval-based drafting extends reuse beyond lexical matching by searching contextual representations or external indices. REST, DReSD, and FastCoder follow this line but rely on external retrieval components or datastores (He et al., 2024; Gritta et al., 2025; Zhao et al., 2025a). In contrast, SRSD

searches only an in-context buffer of intermediate states, avoiding external storage.

Intermediate Representations as a Semantic Index. Intermediate-layer representations capture semantic structure beyond surface forms (Skean et al., 2025; Wu et al., 2025). Building on this observation, SRSD treats cached intermediate states as an in-context semantic index for drafting, enabling semantic proposals without auxiliary models or external datastores.

3 Proposed Method

An overview of SRSD is shown in Figure 2. SRSD follows the standard draft-and-verify paradigm of speculative decoding: at each decoding step t, the target model first selects the next token x_t, then proposes a small set of plausible continuations, and verifies them in a single forward pass. SRSD retrieves semantically similar past continuations and augments them with shallow branches before joint tree verification.

3.1 In-Context Semantic Buffer

SRSD caches the hidden states from a single intermediate layer l^* and reuses them as an in-context semantic index for draft proposal, requiring no external datastore. We select l^* once per model and keep it fixed across all tasks.

3.1.1 Prompt-Conditioned Normalization

Let $\mathbf{h}_i^{(l^*)} \in \mathbb{R}^{d_{\text{hid}}}$ denote the layer-l^* hidden state at committed position i. Raw hidden states are often anisotropic, weakening cosine retrieval (Ethayarajh, 2019). We therefore apply prompt-conditioned centering followed by ℓ_2-normalization. Let the prompt span be $\mathcal{P} = \{1, \ldots, T_p\}$. We first compute the prompt centroid:

$$\bar{\mathbf{u}} \triangleq \frac{1}{T_p} \sum_{i \in \mathcal{P}} \mathbf{h}_i^{(l^*)}. \tag{1}$$

The prompt centroid is computed once during prefill and remains fixed throughout decoding. We then center and normalize each hidden state:

$$\tilde{\mathbf{h}}_i \triangleq \frac{\mathbf{h}_i^{(l^*)} - \bar{\mathbf{u}}}{\left\|\mathbf{h}_i^{(l^*)} - \bar{\mathbf{u}}\right\|_2 + \epsilon}, \tag{2}$$

where ϵ is a small constant for numerical stability. After this transformation, the cosine similarity between $\tilde{\mathbf{h}}_i$ and $\tilde{\mathbf{h}}_j$ reduces to the dot product $\langle \tilde{\mathbf{h}}_i, \tilde{\mathbf{h}}_j \rangle$, which we use as the retrieval score throughout SRSD.

Memory Overhead. SRSD caches one layer of hidden states, adding $T \cdot d_{\text{hid}} \cdot b$ bytes. In contrast, the KV cache costs $T \cdot 2L \cdot d_{\text{kv}} \cdot b$ bytes. Here, L denotes the number of layers, d_{kv} the per-layer KV width, and b the bytes per element. The relative overhead is therefore $\frac{d_{\text{hid}}}{2L \cdot d_{\text{kv}}}$. For Llama3.1-8B, we measure a 6.37% overhead at $T{=}2048$.

3.2 History Retrieval and Primary Trunk

At step t, after selecting the base token x_t, SRSD constructs a draft continuation from the committed prefix. The retrieval query is the most recent normalized hidden state:

$$\mathbf{q}_t \triangleq \tilde{\mathbf{h}}_{t-1}. \tag{3}$$

Intuitively, $\mathbf{q}_t$ represents the current prefix immediately before x_t, while $\tilde{\mathbf{h}}_a$ represents a historical prefix ending at position a. A useful historical anchor should therefore be semantically similar to the current prefix and lead to the same next token x_t.

Not every past position is a valid anchor. A useful anchor must satisfy the first-token alignment condition

$$x_{a+1} = x_t, \tag{4}$$

otherwise the copied continuation would already disagree with the selected base token at its first position. We therefore retrieve only from the aligned candidate set

$$\mathcal{A}_t \triangleq \{\, a \mid 1 \le a \le t-2,\ x_{a+1} = x_t \,\}. \tag{5}$$

In practice, we maintain a token-to-positions index that maps each token ID to the list of positions where it appears as a next token, so $\mathcal{A}_t$ is retrieved in $O(1)$ without scanning the full history. Cosine scoring is then restricted to these aligned positions rather than the full history.

For each $a \in \mathcal{A}_t$, we compute the semantic similarity

$$s_t(a) \triangleq \langle \mathbf{q}_t,\ \tilde{\mathbf{h}}_a \rangle. \tag{6}$$

SRSD selects the highest-scoring aligned anchor

$$a_t^\star \triangleq \arg\max_{a \in \mathcal{A}_t} s_t(a). \tag{7}$$

If $\mathcal{A}_t$ is empty, SRSD skips history drafting at this step. In this case, SRSD falls back to standard autoregressive decoding for the current step. Otherwise, it forms the primary trunk by copying up to K_t tokens from the continuation after the aligned position, where $K_t = \min(K, ; t - a_t^\star - 2)$:

$$\hat{x}^{\text{hist}}_{t+1:t+K_t} \triangleq x_{a_t^\star+2:\, a_t^\star+1+K_t}. \tag{8}$$

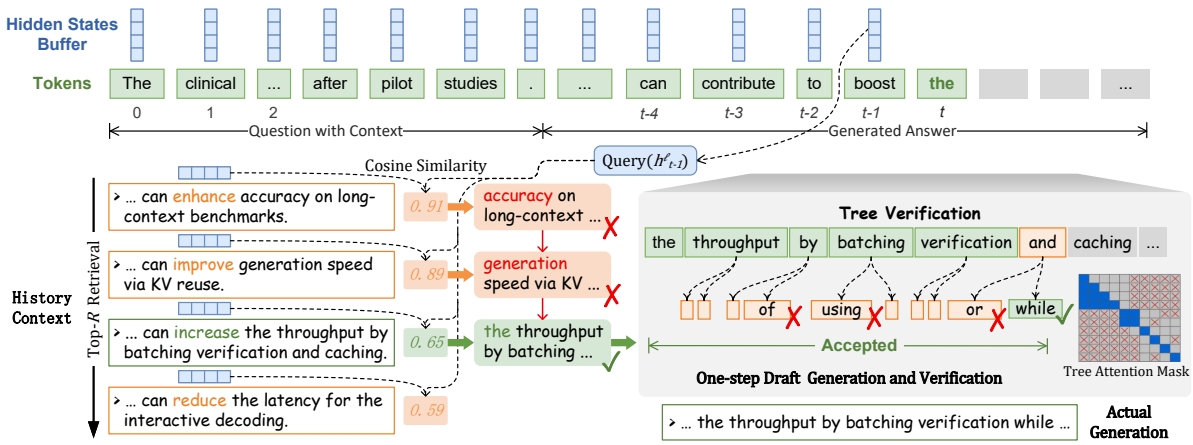

Figure 2: Overview of SRSD. A primary trunk is retrieved from the hidden-state buffer and augmented with shallow branches for joint tree verification.

The aligned token $x_{a_t^\star+1} = x_t$ itself is not drafted; SRSD copies only the subsequent tokens. Anchors with no remaining continuation ($K_t < 1$) are discarded. This history-derived continuation serves as the primary trunk for the current step. SRSD next augments this trunk with local alternatives before joint verification.

3.3 Shallow Branching and Tree Verification

Verifying only one linear draft can be wasteful: once that draft deviates from the target model, all remaining drafted tokens are discarded, even if a nearby local alternative would have matched better. To reduce this waste, SRSD augments the primary history-retrieved trunk with a small set of shallow local alternatives and verifies them jointly.

Shallow Branching. During prefill and after each verification step, SRSD caches up to top-R next-token candidates for committed positions, ranked by the target model's logits. At a later drafting step, once a primary trunk has been retrieved from a historical span, these cached candidates are reused to instantiate local alternatives around that trunk. Specifically, let the retrieved trunk be copied from a historical source span. For the early positions of the trunk, SRSD consults the cached top-R next-token candidates at the corresponding source-prefix position of that span. Any cached token that differs from the trunk token at that position spawns a one-token branch: it shares the same prefix as the trunk up to the branch point and then diverges locally. In practice, SRSD keeps only a small number of such local alternatives so that joint verification remains efficient. These branches act as lightweight local corrections while preserving the long continuation supplied by semantic retrieval.

Tree Verification. SRSD inserts the primary trunk and its shallow local alternatives into a compact trie. If multiple candidate paths are present, they are verified jointly in a single forward pass using tree attention; otherwise, verification reduces to standard linear draft verification. Implementation-wise, we pack the trie into a single verification sequence and construct a custom causal attention mask in which each node attends only to its ancestors. The resulting mask, together with custom `position_ids`, is passed to the HuggingFace model for one-pass tree verification. The target model produces predictions for all candidate nodes simultaneously. For each root-to-leaf path, SRSD counts how many consecutive drafted tokens match the model's greedy predictions from the start of that path; this count is the *accepted prefix length* of the path. SRSD selects the path with the longest accepted prefix, commits the accepted prefix, updates the KV cache to retain only the accepted path, and resumes decoding from the next position. Under stochastic decoding, committed tokens instead follow the user-specified sampling rule applied to the target-model logits, with strictness diagnostics provided in Appendix E.

4 Experimental Setup

Benchmark Setting. We adopt the task formats and prompt sets of Spec-Bench (Xia et al., 2024). The released Spec-Bench harness supports only a limited set of engines and checkpoints; to ensure a fair comparison, we re-implement all methods in a unified HuggingFace-based runner. All methods use identical decoding settings and the same timing instrumentation.

Task Suite. We evaluate six tasks: MT-Bench, Translation, Summarization, Math, Text Editing, and Code Editing. For Translation, Summarization, and Math, we use prompts constructed from WMT14 De→En (Bojar et al., 2014), CNN/DailyMail (Nallapati et al., 2016; Hermann et al., 2015), and GSM8K (Cobbe et al., 2021), respectively. For the two editing tasks, we draw prompts from `agentlans/text-revision` (Hugging Face Datasets, 2024) and CodeEditorBench (Guo et al., 2024).

Baselines. We compare SRSD with autoregressive decoding (AR), Prompt Lookup Decoding (PLD) (Saxena, 2023), CopySpec (Dumitru et al., 2025), PLD+ (Somasundaram et al., 2025), the standard draft-model speculative decoding (SD) (Leviathan et al., 2023), EAGLE-3 (Li et al., 2025), NEST (Li et al., 2024a), and REST (He et al., 2024). All baselines use default settings, and we perform no additional training and no task- or model-specific tuning. In addition, we also report Lookahead (Fu et al., 2024) in Appendix A. We run Lookahead with `llama.cpp` and treat it as a reference due to backend differences.

Metrics. We report wall-clock decode latency and throughput (tokens/s), measuring decode-only time and excluding KV-cache prefill. We also report speedup over AR under identical decoding settings, and log the average accepted draft length.

Implementation Details. Experiments on Qwen2.5-7B (Q2.5-7B) (Yang et al., 2024), Llama3.1-8B (L3.1-8B) (Team, 2024), and CodeLlama-7B-Instruct (CL-7B) (Rozière et al., 2023) use a single RTX 4090D GPU. Experiments on Qwen2.5-14B (Q2.5-14B) and Mistral-Small-3.1-24B-Instruct-2503 (Mistral-24B) (Mistral AI, 2025) use a single RTX PRO 6000 GPU. All main experiments run in `bfloat16`. We evaluate both greedy and stochastic decoding; for sampling we use temperature 0.8 with top-p=0.9 and top-k=0.

SRSD Hyperparameters. For SRSD, we use one fixed retrieval layer per model across all tasks and fixed drafting hyperparameters (K=16, R=4) for all models. For greedy decoding, we verify exact token-level agreement with autoregressive (AR) decoding under identical prompts. For stochastic decoding, we evaluate *rule-matched* correctness, i.e., every committed token must be sampled from the target model's logits under the user-specified sampling policy. We further perform stricter RNG-aligned and distribution-level diagnostics; we summarize the main results in Section 5.1.

5 Evaluation Results

5.1 Overall Performance

As presented in Table 1 and Table 2, SRSD consistently achieves the highest wall-clock speedup across all models. On Llama3.1-8B, SRSD attains a $1.73\times$ greedy speedup, surpassing PLD ($1.48\times$) and SD ($1.26\times$). This efficiency extends to larger architectures, reaching $1.57\times$ on Qwen2.5-14B and $1.53\times$ on Mistral3.1-24B. PLD relies on lexical span reuse and can be brittle when surface forms diverge. For example, PLD yields only $1.07\times$ on Qwen2.5-7B Translation, whereas SRSD achieves $1.20\times$ by retrieving anchors via semantic similarity. This advantage is also pronounced in structured editing: on Llama3.1-8B Code Editing, SRSD achieves $1.84\times$ versus PLD's $1.67\times$. SD can incur substantial overhead when acceptance is low, even causing slowdown (*e.g.*, $0.88\times$ on Llama3.1-8B Translation). In contrast, SRSD accelerates the same setting by $1.46\times$ without auxiliary models or external datastores. SRSD also remains effective under stochastic decoding (T=0.8). While lexical approaches degrade when generated trajectories diverge from the prompt (*e.g.*, PLD drops to $0.96\times$ on Qwen2.5 Translation), SRSD maintains a $1.53\times$ average speedup on Llama3.1-8B, demonstrating superior stability over lexical baselines. Unless otherwise noted, all main results use batch size 1 under the unified evaluation protocol.

Batched Throughput Analysis. We additionally evaluate batched decode throughput on `code_edit` with Qwen2.5-7B under greedy decoding over 20 prompts. Table 3 shows that SRSD remains faster than autoregressive decoding (AR) at all tested batch sizes. The relative speedup decreases as batch size grows, but remains positive throughout.

Correctness and Strictness. Speculative decoding is only meaningful if correctness is preserved. For greedy decoding, we verify exact token-level agreement with autoregressive (AR) decoding under identical prompts. For stochastic decoding, our goal is *rule-matched* correctness: every committed token must be sampled from the target model's logits under the user-specified sampling policy. To make SRSD and AR directly comparable under sampling, we additionally perform an RNG-aligned

Table 1: The overall performance of SRSD against the baselines under greedy decoding.

Model	Method	Trans.	Multi-turn	Code Edit	Math	Text Edit	Summ.	Avg. T.P.	Speedup
Llama3.1-8B	Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	47.18	1.00×
	SD	0.88×	1.18×	1.57×	1.41×	1.18×	1.08×	59.45	1.26×
	PLD	1.25×	1.26×	1.67×	1.49×	1.60×	1.45×	69.83	1.48×
	REST	1.27×	1.09×	1.37×	1.12×	1.31×	1.22×	58.86	1.25×
	NEST	1.30×	1.06×	1.47×	1.14×	1.45×	1.45×	62.60	1.33×
	PLD+	1.36×	1.38×	**1.84×**	1.61×	1.78×	1.66×	77.71	1.65×
	CopySpec	1.31×	1.34×	1.65×	1.48×	1.67×	1.54×	71.12	1.51×
	SRSD (Ours)	**1.46×**	**1.45×**	**1.84×**	**1.85×**	**1.82×**	**1.73×**	**81.64**	**1.73×**
Qwen2.5-7B	Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	50.57	1.00×
	PLD	1.07×	1.15×	1.59×	1.35×	1.44×	1.23×	68.28	1.37×
	PLD+	1.14×	1.33×	1.91×	1.68×	1.88×	1.48×	84.48	1.67×
	CopySpec	1.18×	1.24×	1.76×	1.35×	1.55×	1.31×	74.27	1.47×
	SRSD (Ours)	**1.20×**	**1.39×**	**2.01×**	**1.69×**	**1.91×**	**1.57×**	**87.99**	**1.74×**
Qwen2.5-14B	Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	36.40	1.00×
	SD	1.05×	1.09×	1.21×	1.22×	1.07×	0.97×	40.54	1.11×
	PLD	1.02×	1.13×	1.31×	1.31×	1.31×	1.12×	45.36	1.25×
	SRSD (Ours)	**1.08×**	**1.30×**	**1.70×**	**1.49×**	**1.85×**	**1.35×**	**57.32**	**1.57×**
Mistral3.1-24B	Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	27.19	1.00×
	PLD	1.22×	1.15×	1.33×	1.29×	1.53×	1.23×	36.20	1.33×
	SRSD (Ours)	**1.40×**	**1.31×**	**1.56×**	**1.50×**	**1.75×**	**1.36×**	**41.55**	**1.53×**

Table 2: Overall speedup under stochastic decoding (T=0.8, top-p=0.9). Per-task are in Appendix B.

Model	Method	Speedup	Throughput
L3.1-8B	Autoregressive	1.00×	46.52
	SD	1.09×	50.63
	PLD	1.37×	63.51
	SRSD	**1.53×**	**71.21**
Q2.5-7B	Autoregressive	1.00×	49.97
	PLD	1.25×	62.27
	SRSD	**1.55×**	**77.23**
Q2.5-14B	Autoregressive	1.00×	36.01
	SD	1.10×	39.74
	PLD	1.19×	42.70
	SRSD	**1.42×**	**51.01**
Mistral-24B	Autoregressive	1.00×	27.24
	PLD	1.34×	36.59
	SRSD	**1.42×**	**38.60**

Table 3: Batched decode throughput on `code_edit` with Qwen2.5-7B.

Batch Size	AR (tok/s)	SRSD (tok/s)	Speedup
2	66.06	125.45	1.899×
4	105.16	195.36	1.858×
8	132.32	224.40	1.696×
16	155.75	237.69	1.526×

check in which both methods use the same per-position random variable. Under this protocol, if SRSD applies the same sampling rule to the same target-model logits at each committed position, it must produce the exact same trajectory as AR. Table 4 summarizes this check on tasks; full definitions and diagnostics are provided in Appendix E.

Stress Test on Open-Ended Generation. To probe generalization beyond structured tasks, we

Table 4: Strictness summary on Qwen2.5-7B under stochastic decoding.

Task	Trajectory	TV (mean/max)	JS (mean/max)
Summ.	100%	1.16e−6 / 4.19e−5	8.7e−12 / 8.8e−10
Code Edit	100%	3.91e−7 / 2.79e−5	3.65e−12 / 4.28e−10

additionally evaluate two small open-ended, high-entropy stress tests. See Appendix D. On these prompts, PLD yields near-zero speedup, while our semantic method still achieves a 1.4∼1.6× decode throughput gain.

5.2 Project-Level Long Code Generation

While the Spec-Bench Code Editing task evaluates short-horizon modifications, real-world deployment often involves *project-level* continuation, where the model must generate thousands of tokens. To evaluate SRSD under these conditions, we introduce a dedicated long-context benchmark.

Setup with StackProject-40. We construct StackProject-40 (SP40) by randomly sampling 40 long code files from The Stack (BigCode) (Kocetkov et al., 2023). Each instance provides a prefix and requires generating 2,048 tokens. We detail dataset construction in Appendix C and compare against PLD, EAGLE-3, SD, and REST (He et al., 2024). As depicted in Table 5, SRSD outperforms all baselines across architectures. On CodeLlama-7B, SRSD achieves **5.42**× speedup, reflecting strong regularities in project-level code where semantic retrieval can propose long drafts. On Llama3.1-8B, SRSD (**3.32**×) surpasses PLD (2.32×) and EAGLE-3 (2.17×), highlighting the

Table 5: Throughput on SP40. Speedup and throughput (tokens/s) when generating 2,048 tokens.

Model	Method	Speedup	Throughput
CL-7B	Autoregressive	1.00×	46.15
	REST	1.19×	54.93
	PLD	3.07×	141.65
	SRSD	**5.42×**	**250.21**
L3.1-8B	Autoregressive	1.00×	45.91
	EAGLE-3	2.17×	99.53
	PLD	2.32×	106.85
	SRSD	**3.32×**	**152.37**
Q2.5-7B	Autoregressive	1.00×	49.90
	PLD	1.69×	84.11
	SRSD	**2.46×**	**122.77**
Q2.5-14B	Autoregressive	1.00×	34.86
	SD	1.52×	52.98
	PLD	1.64×	57.11
	SRSD	**2.05×**	**71.30**

benefit in long-horizon generation.

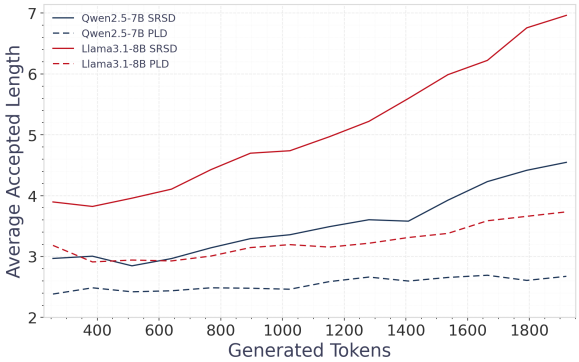

Figure 3: Average accepted length vs. decoding progress on SP40: SRSD vs. PLD.

Accepted Length over Decoding Horizon. Figure 3 compares the average accepted length of SRSD and PLD as a function of decoding progress on SP40. While PLD's acceptance remains roughly flat, SRSD's accepted length grows steadily as more context accumulates, reaching up to 7 tokens per step on Llama3.1-8B. This confirms that the semantic buffer becomes increasingly effective over long-horizon generation, as the growing context provides more and better retrieval candidates.

5.3 Retrieval Overhead

Table 6 breaks down per-step latency on Qwen2.5-7B at varying context lengths. Retrieval (anchor lookup + cosine scoring over the filtered aligned-candidate set) remains under 0.1 ms. Tree construction is similarly lightweight at under 0.5 ms. Both costs are empirically stable over 2K-32K contexts, and the main latency growth comes from target-model verification due to KV-cache attention. This matches SRSD's design: retrieval is restricted to first-token-aligned candidates via a token-to-positions index rather than exhaustive search over the full hidden-state buffer. Overall, SRSD adds negligible overhead relative to model computation within the evaluated range.

Table 6: Per-step latency breakdown (ms) on Q2.5-7B.

Context	Retrieval	Tree Build	Verify	Total
2K	0.10	0.34	26.0	28.7
8K	0.08	0.42	33.7	35.6
16K	0.08	0.48	43.9	45.8
32K	0.09	0.39	64.2	66.2

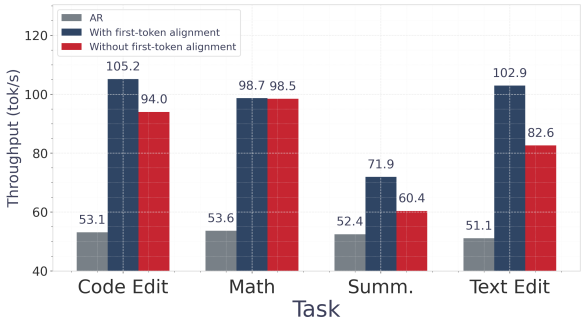

Figure 4: Ablation of first-token alignment on Q2.5-7B.

5.4 Ablation Study

Unless an ablation explicitly varies one hyperparameter, we use greedy decoding with the default setting K=16, and cache top-R=4 logit candidates at each verified position for shallow branching.

First-Token Alignment. A retrieved anchor is only useful if its copied continuation is compatible with the base token selected by the target model at the current step. SRSD therefore restricts history retrieval to anchors that satisfy the first-token alignment condition in Eq. 4. To assess the contribution of this gate, we compare the default aligned variant with an unaligned variant that keeps the same semantic retrieval and verification pipeline but removes the Eq. 4 constraint. Figure 4 shows that enabling first-token alignment consistently improves throughput across all tasks on Qwen2.5-7B. The gain is especially pronounced on text editing and code editing, where semantically related contexts are frequent but many retrieved continuations would otherwise mismatch at the first copied token. These results indicate that first-token alignment is not merely a conservative filter: it materially improves proposal quality and downstream efficiency.

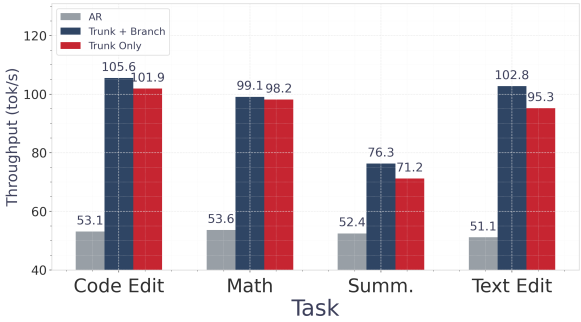

Figure 5: Ablation of shallow branching on Q2.5-7B.

Table 7: Ablation of branching width R on Q2.5-7B.

R	Throughput	Avg. Accept Len.
2	94.8	2.29
4	100.6	2.46
6	97.8	2.48
8	98.6	2.51

Effect of Shallow Branching. Shallow branching augments the primary trunk with rival tokens drawn from cached logit candidates, enabling tree verification to explore multiple plausible continuations in a single forward pass. To more precisely isolate its effect, we compare the default trunk-with-branching configuration against a trunk-only variant that uses the same history retrieval strategy but verifies only a single linear draft sequence. Figure 5 shows throughput across four tasks on Qwen2.5-7B. Branching consistently improves throughput across all tasks. This confirms that shallow branching effectively captures local alternatives missed by the primary trunk alone, and that its benefit compounds over long-horizon generation.

Branching Width R. We vary the number of cached logit candidates $R \in \{2, 4, 6, 8\}$ on Qwen2.5-7B. Table 7 shows that R=2 yields limited branching diversity, while R=4 captures most of the gains with a compact tree. Further increasing R adds more candidate nodes to the tree but does not proportionally improve acceptance. We adopt R=4 as the default.

Retrieval Layer Selection. For each model, we select a single fixed retrieval layer l^* used across all tasks. To avoid data leakage, layer selection is performed on a disjoint development set of 40 prompts that has no overlap with any evaluation benchmark. For each candidate layer, we measure the average accepted draft length on this development set and select the layer with the highest value. Figure 6 shows the average accepted length as a function of layer index on Llama3.1-8B and Qwen2.5-7B. Both models exhibit a broad mid-to-late layer plateau, where performance varies by less than 2% across a wide range of layers. This indicates that layer selection is not sensitive, and a single mid-late layer is a practical default for deployment.

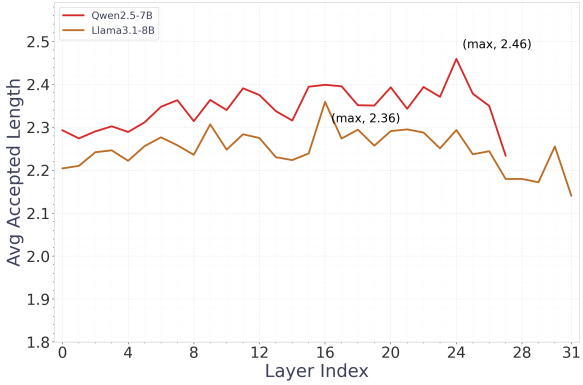

Figure 6: Average accepted length vs. retrieval layer index on Llama3.1-8B and Qwen2.5-7B.

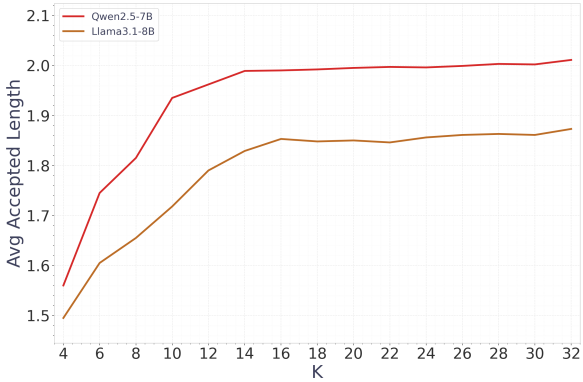

Figure 7: Average accepted length vs. maximum draft length K on Llama3.1-8B and Qwen2.5-7B.

Maximum Draft Length K. We sweep K from 4 to 32 on best layer per model. Figure 7 shows that the mean accepted length grows steadily up to K=16, after which further increases yield only marginal gains in acceptance while adding wasted computation on rejected suffixes. We therefore adopt K=16 as a stable default that balances acceptance rate and verification efficiency.

Normalization Variants for Semantic Retrieval. Our main results use prompt-conditioned mean-centering (Section 3.1.1), ℓ_2 normalization, and cosine scoring (Eq. 6). Appendix F.1 ablates variants (*e.g.*, no centering, diagonal whitening) and shows mean-centering is a strong default, with whitening giving a small additional gain.

6 Conclusion

This paper proposes SRSD, a training-free draft-and-verify method that reuses intermediate-layer

hidden states as an in-context semantic index. SRSD retrieves semantically similar past positions via first-token alignment and cosine similarity, augments the primary draft with shallow branches, and verifies multiple candidates jointly in a single target-model pass with KV-cache reuse. Without auxiliary drafters or external datastores, SRSD consistently outperforms lexical reuse and parametric baselines across five model and six task types, reaching up to $5.42\times$ speedup on project-level long code generation.

Limitations

Dependence on Semantic Redundancy and Copyability. SRSD benefits most from workloads where the committed prefix contains semantically related and lexically copyable continuations. Drafting is activated only when a retrieved anchor passes the first-token alignment gate (Eq. 4) and yields a non-trivial accepted prefix. This condition is common in structured, low-entropy settings (*e.g.*, code editing), but is weaker in high-entropy generation such as open-ended writing or translation, where semantically related contexts may diverge lexically, leading to smaller speedups (Tables 1-2). Exploring softer triggers (*e.g.*, top-k alignment) is a promising direction but may trade off exactness and simplicity.

Retrieval Scalability in Ultra-Long Contexts. SRSD does not perform exhaustive cosine scoring over the full hidden-state buffer at each step; instead, retrieval is restricted to first-token-aligned candidate positions via a token-to-positions index. In our experiments, this makes retrieval and tree construction nearly constant-cost up to 32K context and negligible relative to verification. However, for substantially longer contexts, the aligned candidate set for very frequent tokens may still grow, and more advanced indexing or approximate search could further improve scalability.

Acknowledgments

This research was supported by the Ningbo Yongjiang Talent Programme, by the Zhejiang Provincial Natural Science Foundation of China under Grant No. LQN26F020008, and CAAI-Ant Group Research Fund (2025CAAI-ANT-15).

References

Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Ales Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation, WMT@ACL 2014, June 26-27, 2014, Baltimore, Maryland, USA*, pages 12–58.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 5209–5235.

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

Razvan-Gabriel Dumitru, Minglai Yang, Vikas Yadav, and Mihai Surdeanu. 2025. Copyspec: Accelerating llms with speculative copy-and-paste. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 26301–26332.

Kawin Ethayarajh. 2019. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and GPT-2 embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 55–65.

Angela Fan, Mike Lewis, and Yann N. Dauphin. 2018. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 889–898.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of LLM inference using lookahead decoding. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 14060–14079.

Milan Gritta, Huiyin Xue, and Gerasimos Lampouras. 2025. Dresd: Dense retrieval for speculative decoding. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pages 19822–19832.

Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi Li, Ruibo Liu, Yue Wang, Shuyue Guo, Xingwei Qu, Xiang Yue, Ge Zhang, Wenhu Chen, and Jie Fu. 2024. Codeeditorbench: Evaluating code editing capability of large language models. *CoRR*, abs/2404.03543.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. 2024. REST: retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 1582–1595.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1693–1701.

Hugging Face Datasets. 2024. Text Revision dataset. `https://huggingface.co/datasets/agentlans/text-revision`. Accessed: 2026-01-05.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2023. The stack: 3 TB of permissively licensed source code. *Trans. Mach. Learn. Res.*, 2023.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. 2023. Openassistant conversations - democratizing large language model alignment. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Alan Chi-Man Lee, Wing-Sun Cheng, and Calvin Chun-Kit Chan. 2025. PROMTEC: fast LLM inference decoding using prompt multi-lookup with template database and common sequences. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pages 6830–6842.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286.

Minghan Li, Xilun Chen, Ari Holtzman, Beidi Chen, Jimmy Lin, Scott Yih, and Victoria Lin. 2024a. Nearest neighbor speculative decoding for LLM generation and attribution. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Tingting Li, Ziming Zhao, Zhaoxuan Li, Xiaofei Yue, and Jiongchi Yu. 2026. Fair and carbon-aware LLM routing for web services. In *Proceedings of the ACM Web Conference 2026, WWW 2026, Dubai, United Arab Emirates, originally scheduled for April 13-17, 2026, rescheduled for June 29 - July 3, 2026*, pages 9563–9571.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 28935–28948.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE-3: scaling up inference acceleration of large language models via training-time test. *CoRR*, abs/2503.01840.

Zhipeng Liu, Yifan Zheng, Fanqi Kong, and Ziming Zhao. 2026. QUARTZ: quantile-aware routing and queueing for TTFT slos in LLM serving. In *Findings of the Association for Computational Linguistics, ACL 2026, San Diego, California, United States, July 2-7, 2026*, pages 37886–37896.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. 2025. Turning trash into treasure: Accelerating inference of large language models with token recycling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 6816–6831.

Huidong Ma, Xinyan Shi, Hui Sun, Xiaofei Yue, Xiaoguang Liu, Gang Wang, and Wentong Cai. 2026. Efficient learned data compression via dual-stream feature decoupling. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pages 7151–7164.

Mistral AI. 2025. Mistral-small-3.1-24b-instruct-2503. `https://huggingface.co/mistralai/Mistral-Small-3.1-24B-Instruct-2503`. Accessed: 2026-01-05.

Ramesh Nallapati, Bowen Zhou, Cícero Nogueira dos Santos, Çaglar Gülçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural*

Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016, pages 280–290.

Gabriele Oliaro, Zhihao Jia, Daniel F. Campos, and Aurick Qiao. 2025. Suffixdecoding: Extreme speculative decoding for emerging AI applications. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, and 6 others. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Apoorv Saxena. 2023. Prompt lookup decoding. https://github.com/apoorvumang/prompt-lookup-decoding. Accessed: 2025-12-09.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. 2025. Layer by layer: Uncovering hidden representations in language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*.

Shwetha Somasundaram, Anirudh Phukan, and Apoorv Saxena. 2025. PLD+: accelerating LLM inference by leveraging language model artifacts. In *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, volume NAACL 2025 of *Findings of ACL*, pages 6075–6089.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10107–10116.

Llama Team. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Zhaofeng Wu, Xinyan Velocity Yu, Dani Yogatama, Jiasen Lu, and Yoon Kim. 2025. The semantic hub hypothesis: Language models share semantic representations across languages and modalities. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, volume EMNLP 2023 of *Findings of ACL*, pages 3909–3925.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, volume ACL 2024 of *Findings of ACL*, pages 7655–7671.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Inference with reference: Lossless acceleration of large language models. *CoRR*, abs/2304.04487.

Jiongchi Yu, Xiaofei Xie, Qiang Hu, Yuhan Ma, and Ziming Zhao. 2026. Chimera: Harnessing multi-agent llms for automatic insider threat simulation. In *33rd Annual Network and Distributed System Security Symposium, NDSS 2026, San Diego, California, USA, February 23-27, 2026*.

Xiaofei Yue, Song Yang, Fan Li, Youqi Li, and Yu Wang. 2026a. Rocket: Warming serverless inference via hierarchical ML artifact pre-loading and sharing. In *IEEE INFOCOM 2026 - IEEE Conference on Computer Communications, Tokyo, Japan, May 18-21, 2026*, pages 1–10.

Xiaofei Yue, Fangming Zhao, Fulun Ye, Jiongchi Yu, Zhaoxuan Li, Tingting Li, Ziming Zhao, and Jianwei Yin. 2026b. Heterosim: Towards high-fidelity heterogeneous LLM training simulation on gpus. In *Proceedings of the ACM Web Conference 2026, WWW 2026, Dubai, United Arab Emirates, originally scheduled for April 13-17, 2026, rescheduled for June 29 - July 3, 2026*, pages 5189–5197.

Xiaofei Yue, Ziming Zhao, Jiongchi Yu, Huidong Ma, Zhaoxuan Li, Tingting Li, and Jianwei Yin. 2026c. sMoE: Elastic MoE-based inference with serverless computing via fine-grained expert scaling. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference (DAC)*. ACM.

Ziyin Zhang, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Rui Wang, and Zhaopeng Tu. 2025. Draft model knows when to stop: Self-verification speculative decoding for long-form generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 16685–16697.

Qianhui Zhao, Li Zhang, Fang Liu, Xiaoli Lian, Qiaoyuanhe Meng, Ziqian Jiao, Zetong Zhou, Jia Li, and Lin Shi. 2025a. Fastcoder: Accelerating repository-level code generation via efficient retrieval and verification. In *40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16-20, 2025*, pages 2299–2311.

Ziming Zhao, Zhaoxuan Li, Tingting Li, and Fan Zhang. 2025b. Cyberllm: Enable mapping CVE to tactics and techniques of cyber threats via LLM. In *Database Systems for Advanced Applications - 30th International Conference, DASFAA 2025, Singapore, Singapore, May 26-29, 2025, Proceedings, Part V*, volume 15990 of *Lecture Notes in Computer Science*, pages 473–488.

A Baseline Hyperparameters and Draft Models

Unified Evaluation Protocol. We evaluate all baselines under a unified HuggingFace-based runner with identical tokenizers, stop-token handling, and batch size 1, using `max_new_tokens=2048`. We report *decode-only* latency and throughput, excluding prefill.

Autoregressive (AR). Standard autoregressive decoding under the same decoding rule.

PLD (Saxena, 2023). We use $n = 4$ and maximum draft length $K = 16$ for all models.

Speculative Decoding (SD) (Leviathan et al., 2023). We use off-the-shelf draft models without additional finetuning. For Llama3.1-8B, the drafter is Llama3.2-1B; for Qwen2.5-14B, the drafter is Qwen2.5-0.5B. Verification uses the same decoding rule as the target model.

Lookahead (Fu et al., 2024). We evaluate Lookahead using the official `llama.cpp` implementation (CUDA backend). Due to substantial backend differences (*e.g.*, kernels, KV-cache implementation, and runtime optimizations) relative to HuggingFace, we treat these results as reference rather than strict ranking (Table 8).

REST / CopySpec / NEST / PLD+. For REST (He et al., 2024), CopySpec (Dumitru et al., 2025), NEST, and PLD+ (Somasundaram et al., 2025), we use the corresponding public implementations or faithful re-implementations under the same unified evaluation protocol whenever possible. We follow the default or recommended settings of each method and avoid task- or model-specific tuning. When a maximum draft length is applicable, we keep it aligned with our main training-free baselines for fair comparison.

Table 8: Throughput comparison across backends.

Method	Engine	Throughput (tok/s)
AR	HuggingFace	44.53
SRSD	HuggingFace	78.18
Lookahead	`llama.cpp`	54.80

Licenses and terms. We use publicly available models, datasets, and benchmarks under their respective licenses and terms of use. Our released code is for research purposes; we do not redistribute third-party model weights or dataset contents, and we refer users to the original sources for access and licensing details.

B Per-Task Stochastic Decoding Results

Table 9 reports per-task speedup under stochastic decoding (T=0.8, top-p=0.9), complementing the summary in Table 2.

C SP40 Construction Details

This appendix describes SP40, a project-level code-continuation benchmark designed to stress long-context, memory-bandwidth-limited decoding and measure long-horizon throughput.

Data Source. We use The Stack (Kocetkov et al., 2023) via the HuggingFace dataset interface. We stream the deduplicated corpus and treat each entry as a file, preserving repository metadata such as repository name and file path.

Filtering Criteria. We enforce length constraints via a two-stage pipeline: (i) a heuristic character-level prefilter to remove files far below the target length; and (ii) an exact token-count filter. For (ii), we tokenize files with the **Llama3.1-8B-Instruct** tokenizer and retain only files with at least 3,000 tokens.

Sampling Strategy. We apply reservoir sampling over the streaming candidates to select 40 qualifying files.

Prompt Engineering. For each instance, we truncate the file to a fixed prompt budget of 800 tokens under the target tokenizer, then wrap the prefix with repository metadata (name and file path) and a continuation instruction.

Table 9: Per-task performance of SRSD against the baselines under stochastic decoding (T=0.8, top-p=0.9).

Model	Method	Trans.	Multi-turn	Code Edit	Math	Text Edit	Summ.	Avg. T.P.	Speedup
Llama3.1-8B	Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	46.52	1.00×
	SD	0.83×	1.10×	1.15×	1.13×	1.10×	0.96×	50.63	1.09×
	PLD	1.23×	1.22×	1.44×	1.28×	1.48×	1.33×	63.51	1.37×
	SRSD (Ours)	**1.41×**	**1.32×**	**1.66×**	**1.44×**	**1.62×**	**1.51×**	**71.21**	**1.53×**
Qwen2.5-7B	Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	49.97	1.00×
	PLD	0.96×	1.07×	1.43×	1.27×	1.29×	1.10×	62.27	1.25×
	SRSD (Ours)	**1.17×**	**1.28×**	**1.81×**	**1.45×**	**1.66×**	**1.37×**	**77.23**	**1.55×**
Qwen2.5-14B	Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	36.01	1.00×
	SD	0.97×	1.03×	1.26×	1.31×	1.04×	0.94×	39.74	1.10×
	PLD	1.03×	1.10×	1.25×	1.29×	**1.21×**	1.09×	42.70	1.19×
	SRSD (Ours)	**1.07×**	**1.24×**	**1.55×**	**1.34×**	1.07×	**1.28×**	**51.01**	**1.42×**
Mistral3.1-24B	Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	27.24	1.00×
	PLD	1.25×	1.18×	1.36×	1.33×	1.51×	1.22×	36.59	1.34×
	SRSD (Ours)	**1.28×**	**1.23×**	**1.44×**	**1.36×**	**1.62×**	**1.27×**	**38.60**	**1.42×**

Generation Horizon and Metrics. SP40 is designed for throughput measurement. For each prompt, we generate up to 2,048 new tokens and report decode-only throughput (excluding prefill), following the timing protocol described in Section 4.

Reproducibility Metadata. We record per-instance metadata, including repository name, file path, total token count, and prefix length. We additionally store a content hash (*e.g.*, SHA1) for each selected file to support exact regeneration under the same corpus snapshot.

Representativeness. Although SP40 is author-constructed, it is not manually written or synthetic. Each instance is sampled from a real open-source file in The Stack and preserves the original repository name, file path, file prefix, and in-file continuation target. As a result, SP40 reflects repository-level code continuation with naturally occurring long prefixes, rather than short standalone programming puzzles or manually curated completion prompts. This construction makes SP40 particularly suitable for studying long-context decoding behavior in realistic project-style continuation settings.

Scope and External Validity. We emphasize that SP40 is intended as a diagnostic stress test for long-context project completion, rather than a universal benchmark for all code-generation workloads. Its primary role is to isolate settings where long prefixes, file-level continuity, and long-horizon decoding throughput are central. We therefore use SP40 to analyze mechanism-level behavior under realistic repository continuations, while interpreting its conclusions together with results on broader and more heterogeneous evaluation sets reported in the main paper.

D High-Entropy Open-Ended Generation Stress Tests

Benchmarks. To test open-ended, high-entropy generation (where lexical reuse is limited), we construct two stress-test sets: (i) **Writing**: creative writing prompts from WRITINGPROMPTS (Fan et al., 2018); (ii) **Chat**: assistant-style user requests from OPENASSISTANT conversations (Köpf et al., 2023). For each set, we sample 40 English prompts and require at least 50 prompt tokens (under the target tokenizer) to avoid overly short inputs.

Evaluation Protocol. We follow the unified protocol in Appendix A (batch size 1, `max_new_tokens`=2048) and report *decode-only* throughput (tokens/s), excluding prefill.

Table 10 reports throughput and speedup over AR. On these open-ended tasks, PLD provides little to no speedup, while SRSD maintains a consistent 1.4∼1.6× throughput gain.

Table 10: *High-entropy open-ended generation stress tests.* Throughput (with speedup) on Qwen2.5-7B.

Method	Writing	Chat
AR	50.18 (1.00×)	49.33 (1.00×)
PLD	50.11 (1.00×)	50.89 (1.03×)
SRSD	77.56 (1.55×)	71.64 (1.45×)

E Stochastic Decoding Strictness

Speculative decoding need not preserve per-seed sequence identity, since it may consume randomness

along a different execution path than autoregressive (AR) sampling. Our goal is *rule-matched* correctness: each committed token must be produced by applying the user-specified sampling rule to the target model's logits. We therefore report (i) RNG-aligned trajectory identity and (ii) per-position distribution agreement.

Sampling Rule. Let $\boldsymbol{\ell}_t$ be the target-model logits at position t. A sampling policy $\pi_{T,p,k}$ (temperature T, nucleus top-p, optional top-k) defines the truncated distribution:

$$p_t(\cdot) \;=\; \pi_{T,p,k}(\boldsymbol{\ell}_t). \tag{9}$$

Sampling draws a token using a single uniform random variable $u_t \sim \mathrm{Unif}(0,1)$ via inverse-CDF sampling on p_t.

E.1 RNG-Aligned Sampling Trajectories

To make AR and SRSD comparable under sampling, we use a counter-based RNG keyed by (global seed, sample id s, position t) so both methods draw the same uniform variable $u_{s,t}$ when sampling token t, regardless of how many speculative verification calls are made:

$$\begin{aligned} h_{s,t} &= \mathrm{hash}(g, s, t), \\ u_{s,t} &= (h_{s,t} + 0.5) \cdot 2^{-64}. \end{aligned} \tag{10}$$

Here, $\mathrm{hash}(\cdot)$ outputs a 64-bit integer, yielding $u_{s,t} \in (0,1)$.

Under RNG alignment, if SRSD applies $\pi_{T,p,k}$ to the same target-model logits at each committed position, it must produce the *exact* same trajectory as AR. We check full-sequence identity and record any divergence.

E.2 Per-Position Distribution Agreement

Trajectory identity is a strong end-to-end check. To localize discrepancies, we additionally compare the distributions used by AR (p_t) and SRSD (q_t) position-by-position under the same prefix.

Distance Metrics. We compare p_t and q_t using Total Variation (TV) and Jensen-Shannon (JS) divergence:

$$\begin{aligned} \mathrm{TV}(p_t, q_t) &= \tfrac{1}{2} \sum_{v \in \mathcal{V}} |p_t(v) - q_t(v)|, \\ \mathrm{JS}(p_t, q_t) &= \tfrac{1}{2}\mathrm{KL}(p_t \| m_t) + \tfrac{1}{2}\mathrm{KL}(q_t \| m_t), \end{aligned} \tag{11}$$

where $m_t = \frac{1}{2}(p_t + q_t)$. Distances are computed over the full vocabulary by treating truncated-out tokens as probability 0.

Implementation Notes. All diagnostics are computed in `float32`. We report distances only up to (and excluding) the first divergence position.

On Qwen2.5-7B-Instruct with stochastic decoding (T=0.8, top-p=0.9), AR and SRSD produce identical trajectories on summarization prompts, and also on code-editing prompts (512 tokens per prompt under RNG alignment). As shown in Table 11, distribution gaps are negligible (near machine epsilon), confirming that SRSD follows the AR sampling distribution at committed positions.

Table 11: *RNG-aligned strictness check.* AR vs. SRSD on Qwen2.5-7B-Instruct (T=0.8, p=0.9). Metrics are averaged over 1,175 positions (summarization) and 5,120 positions (code edit).

Setting	Match	TV (mean/max)	JS (mean/max)
RNG-aligned (Summ.)	100%	1.16e-6 / 4.19e-5	8.7e-12 / 8.8e-10
RNG-aligned (Code edit)	100%	3.91e-7 / 2.79e-5	3.65e-12 / 4.28e-10

F Additional Ablation Studies

F.1 Normalization Variants

We compare alternative normalization choices for semantic retrieval using the following Variants.

Variants. Let $\mathbf{h}_i \in \mathbb{R}^{d_{\mathrm{hid}}}$ be the hidden state at position i from the retrieval layer, and let ϵ be a small constant.

(1) No Centering (ℓ_2 only).

$$\tilde{\mathbf{h}}_i \;=\; \frac{\mathbf{h}_i}{\|\mathbf{h}_i\|_2 + \epsilon}. \tag{12}$$

(2) Prompt-Conditioned Mean-Centering (default). We compute the prompt centroid over $\mathcal{P} = \{1, \ldots, T_p\}$:

$$\bar{\mathbf{u}} \;=\; \frac{1}{T_p} \sum_{i \in \mathcal{P}} \mathbf{h}_i, \tag{13}$$

then center and normalize:

$$\tilde{\mathbf{h}}_i \;=\; \frac{\mathbf{h}_i - \bar{\mathbf{u}}}{\|\mathbf{h}_i - \bar{\mathbf{u}}\|_2 + \epsilon}. \tag{14}$$

(3) Mean-Centering + Diagonal Whitening. We compute a prompt-only diagonal standard deviation $\boldsymbol{\sigma} \in \mathbb{R}^{d_{\mathrm{hid}}}$:

$$\boldsymbol{\sigma}^2 \;=\; \frac{1}{T_p} \sum_{i \in \mathcal{P}} (\mathbf{h}_i - \bar{\mathbf{u}})^{\odot 2}, \tag{15}$$

then whiten and normalize:

$$\tilde{\mathbf{h}}_i \;=\; \frac{(\mathbf{h}_i - \bar{\mathbf{u}}) \oslash (\boldsymbol{\sigma} + \epsilon)}{\|(\mathbf{h}_i - \bar{\mathbf{u}}) \oslash (\boldsymbol{\sigma} + \epsilon)\|_2 + \epsilon}. \tag{16}$$

Table 12: *Ablation of normalization variants* on Qwen2.5-7B `code_edit`.

Variant	Throughput (tok/s)
No centering (ℓ_2 only)	99.91
Mean-centering (default)	100.64
Mean-centering + diag whitening	101.01

where $\odot$ and $\oslash$ denote elementwise operations.

All variants use cosine scoring via dot product:

$$s_t(a) \;=\; \left\langle \tilde{\mathbf{h}}_{t-1}, \tilde{\mathbf{h}}_a \right\rangle, \tag{17}$$

and Table 12 shows that mean-centering improves retrieval quality over raw ℓ_2 normalization. Diagonal whitening yields a further but modest gain.

F.2 Two-layer Retrieval Fusion

We test a two-layer fusion variant that keeps drafting, the first-token gate, and verification unchanged, and only modifies retrieval scoring.

Two-layer fusion rule. Let ℓ_1 and ℓ_2 be two retrieval layers. We compute cosine scores at each layer, retrieve Top-R anchors

$$A_k \;=\; \mathrm{TopR}\big(\mathrm{sim}_{\ell_k}(t)\big), \quad k \in \{1, 2\}, \tag{18}$$

take the intersection $A = A_1 \cap A_2$, and rank by the summed score

$$s(a) \;=\; \mathrm{sim}_{\ell_1}(a) + \mathrm{sim}_{\ell_2}(a), \quad a \in A. \tag{19}$$

Anchor selection is followed by the same first-token gate and one-pass verifier as in the main method.

On `code_edit` with Qwen2.5-7B-Instruct, two-layer fusion does not improve throughput: single-layer retrieval achieves 101.2 tok/s, while fusion achieves 95.8 tok/s (Table 13).

Table 13: Two-layer retrieval fusion ablation on `code_edit` (Qwen2.5-7B-Instruct).

Method	Throughput (tok/s)
Single-layer retrieval	101.2
Two-layer fusion	95.8

F.3 Scalability and Windowed Retrieval

We ablate windowed retrieval, restricting the search space at step t to the most recent W tokens:

$$\mathcal{S}_t \;=\; \{\max(1, t - W), \ldots, t - 1\}. \tag{20}$$

Table 14: *Windowed retrieval ablation on Qwen2.5-7B (`code_edit`).* Throughput (tok/s) at long contexts.

Method	16K	32K
Global retrieval (no window)	310.65	272.85
Windowed retrieval ($2K$ tokens)	148.45	110.64

All other components remain unchanged.

Windowed retrieval substantially degrades end-to-end throughput. On `code_edit`, global retrieval reaches 310.65 tok/s at 16K context and 272.85 tok/s at 32K, while a 2K-token window drops throughput to 148.45 and 110.64 tok/s, respectively (Table 14). Since the global retrieval kernel remains sub-millisecond even at 32K, the reduced similarity-search cost from windowing does not compensate for the loss in proposal quality.