SUFFIXSPEC: HIERARCHICAL CONTEXT-AWARE MEMORY FOR TRAINING-FREE SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates large language model (LLM) inference by drafting candidate tokens and verifying them in one forward pass. Training-free drafters avoid training a draft model for every target model, but they either copy text from the prompt, the generated history, or an external corpus, or index the target model's own predictions by a single token, and none of them uses the predictions that the target model already computes at every prompt position during prefill. We cast drafting as retrieval from a memory of these predictions, organized along two axes: the granularity at which contexts are keyed and the lifetime of stored evidence. Along these axes we build **SuffixSpec**, a hierarchical memory whose suffix-local, token-local, and token-global layers are queried in order of decreasing specificity; a *prefill bootstrap* populates it before the first decoding step, and a *dual-timescale commit* updates the persistent layer once per request with decode-time predictions only. On seven tasks and three models from 8B to 32B parameters, SuffixSpec achieves up to $2.37\times$ speedup over autoregressive decoding and improves over the strongest training-free baseline by up to $0.36\times$. In a controlled comparison inside the SGLang serving engine on Qwen3-8B with 8K-token contexts, it outperforms the trained drafters EAGLE-3 and DFlash under greedy ($2.45\times$ vs. $2.08\times$ and $2.24\times$) and stochastic decoding, while keeping about 10 MB of draft state in host memory instead of 2 to 5 GiB of extra GPU memory for the drafters. The code is available at `https://anonymous.4open.science/r/suffixspec-8075`.

1 INTRODUCTION

Large language models (LLMs) generate text autoregressively, and each decoding step reads all model weights to emit one token, so decoding is memory-bandwidth bound and dominates the cost of long generations. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) addresses this bottleneck: a lightweight drafter proposes several tokens, and the target model verifies all of them in one forward pass, reproducing its greedy output exactly. The speedup therefore hinges on the drafter, which must propose tokens that the target model accepts at a fraction of the cost of a forward pass.

Existing drafters fall into two families, depending on whether they rely on a trained draft model. *Trained drafters*, which include the strongest drafters to date such as EAGLE-3 (Li et al., 2025) and DFlash (Chen et al., 2026), are small models trained on the target model's hidden features; they must be trained for every target model, retrained when it changes, and served alongside it in GPU memory. *Training-free drafters* need no draft model and thus avoid these costs. Retrieval-based methods copy the continuation of a matching n-gram from the prompt, the generated history, or an external datastore (Saxena, 2023; Dumitru et al., 2025; He et al., 2024; Hu et al., 2025; Liu et al., 2024b), and Token Recycling (TR) (Luo et al., 2025) reuses the target model's own top-K predictions from earlier verification steps as drafts for later steps, indexed by the current token alone.

Training-free drafters, however, leave much of the information they could draft from unused. Copied text is only a proxy for what the target model would predict, a single-token index such as TR's cannot separate contexts that share their last token but not their continuation, and all of these methods discard the predictions that the target model computes at every prompt position during prefill. Figure 1 quantifies these gaps by replaying a simple draft memory along greedy generations of Llama-3.1-8B.

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.

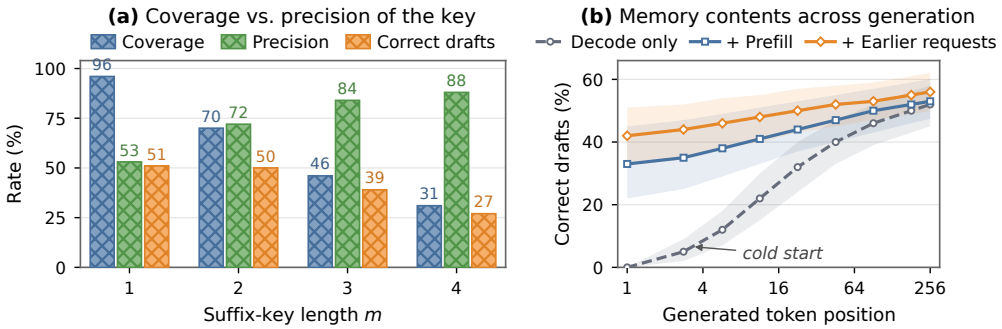


Figure 1: Replaying a memory of the target model's top-1 predictions along greedy generations of Llama-3.1-8B. (a) Keys of the last m tokens (coverage: the lookup hits; precision: the hit is correct). (b) A single-token key with three memory contents; bands span the seven tasks.

A single-token key almost always finds a stored prediction, but that prediction is often wrong because the same token is followed by different continuations in different contexts, whereas a three-token key is precise but available far less often (Figure 1a). A memory filled only during decoding also starts empty, although prefill has already computed a prediction at every prompt position and earlier requests have left reusable predictions behind (Figure 1b); both sources go unused.

Our key insight is that the target model already produces predictive signals at multiple granularities: prefill logits encode prompt-conditioned predictions at every position, and each verification step reveals context-dependent continuations at every node of the draft tree. These signals are naturally hierarchical, since short suffixes carry context-specific information while individual tokens capture broader patterns, and they span two timescales, the current prompt and the stream of requests. No existing method exploits this structure, although doing so costs no extra forward pass.

We propose **SuffixSpec**, a hierarchical context-aware memory for training-free speculative decoding built around this structure. Because no single key length is both precise and available, it keys the model's predictions at two granularities and queries them from the most to the least specific; because prefill and earlier requests hold predictions that a decode-only memory lacks, it draws on all three sources and keeps each at the timescale where it stays valid, through a *prefill bootstrap* for the current request and a *dual-timescale commit* across requests. It requires no draft model, training, or external datastore, and leaves the tree template and the verification rule unchanged. On seven tasks across three models from 8B to 32B parameters, SuffixSpec achieves up to $2.37\times$ speedup over autoregressive decoding and outperforms every training-free baseline, by up to $0.36\times$ over the strongest one. In a controlled comparison inside the SGLang serving engine, it also outperforms the trained drafters EAGLE-3 and DFlash on 8K-token contexts with about 10 MB of host memory instead of 2 to 5 GiB of extra GPU memory. Our main contributions are as follows:

- An analysis showing that the target model's own predictions carry complementary information at different key granularities and from three sources, prefill, decoding, and earlier requests, which existing training-free drafters exploit only in part, never all together (Figure 1).
- SuffixSpec, a single memory design derived from this analysis, with hierarchical keys (§3.2), evidence from prefill and verification (§3.3), and a dual-timescale commit (§3.4).
- State-of-the-art speedups among training-free methods on three models, and higher speedups than EAGLE-3 and DFlash on long contexts at a small fraction of their memory (§4).

2 Related Work

Speculative Decoding and Trained Drafters. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) verifies drafted tokens with the target model and preserves its output distribution exactly; SpecInfer (Miao et al., 2024) and Medusa (Cai et al., 2024) verify a tree of candidates in one pass, whose shape EAGLE-2 (Li et al., 2024b) adapts to draft confidence and Sequoia (Chen et al., 2024) optimizes by dynamic programming. Most drafters are trained for a specific target model: Medusa adds decoding heads, EAGLE (Li et al., 2024a; 2025) drafts at the feature level, DFlash (Chen et al., 2026) drafts a block with a block-diffusion model, and online speculative decoding (Liu et al., 2024a) keeps distilling the drafter on served queries. All of them pay the training and memory costs of §1; §4.3 and Appendix C.4 compare them with SuffixSpec, which changes only the drafter.

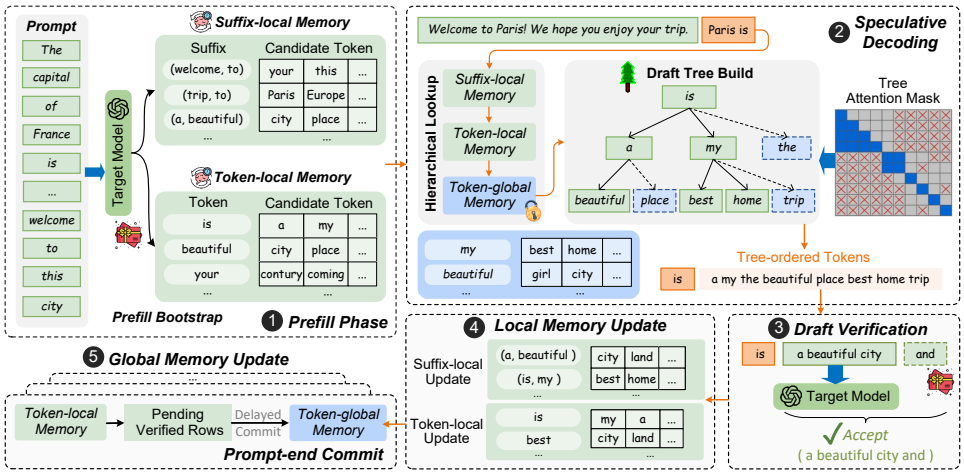


Figure 2: Overview of SuffixSpec for one request: prefill bootstrap, hierarchical lookup of every draft-tree node, write-back after verification, and verified-only commit when the request ends.

Training-Free Drafters. Self-speculative decoding (Zhang et al., 2024) drafts with part of the target model, so each draft token still costs compute, whereas retrieval-based drafters cost only lookups and differ in the three choices of §3.1. As *evidence*, PLD (Saxena, 2023), PLD+ (Somasundaram et al., 2025), and CopySpec (Dumitru et al., 2025) copy from the current context, REST (He et al., 2024) from an external datastore, SAM (Hu et al., 2025) from suffix automata over the generated text and optionally a corpus, and Suffix Decoding (Liu et al., 2024b) from suffix trees over the prompt and past outputs; Lookahead (Fu et al., 2024) instead uses n-grams that the target model generates in Jacobi iterations, and Token Recycling (Luo et al., 2025), the closest work, stores its top-K verification predictions. As *keys*, text-matching methods use variable-length matches and TR a single token; in *lifetime*, PLD-style methods and dynamic SAM use only the current request, REST and static SAM a fixed corpus, and Suffix Decoding and TR accumulate evidence across requests. Predicting from stored contexts goes back to back-off n-gram models (Katz, 1987) and nearest-neighbor language models (Khandelwal et al., 2020); SuffixSpec borrows the backoff structure but stores the target model's own predictions under token-suffix keys, so a lookup costs one hash. It is the only one of these drafters that keys target predictions hierarchically, draws them from both prefill and verification, and separates request-local from persistent evidence; §4.2 compares it with all of them.

Relaxed Verification. Typical acceptance (Cai et al., 2024), BiLD (Kim et al., 2023), judge decoding (Bachmann et al., 2025), and other relaxed or adaptive criteria (Fang et al., 2025; Sun et al., 2025) accept drafts that exact verification would reject, trading equivalence with the target model for longer accepted runs. We do not compare with them for two reasons: their outputs differ from those of the target model, so a comparison would have to weigh speed against output quality, whereas all methods in §4 are lossless and compared by speed alone; and they modify the verifier rather than the drafter, so they are orthogonal to SuffixSpec and can directly consume its drafts.

3 Method

SuffixSpec replaces the draft model with a memory of the target model's own predictions, organized in three tables that map a context to K candidate next tokens: a *suffix-local* table keyed by the last few tokens, a *token-local* table keyed by the last token, and a *token-global* table that is also keyed by the last token but persists across requests. The two local tables span the granularity axis within a request, and the global table spans the timescale axis. Figure 2 traces one request. After the preliminaries (§3.1), we explain how contexts are keyed (§3.2), how the tables are filled (§3.3), and which entries persist across requests (§3.4); §3.5 then gives the complete algorithm for one request.

3.1 Preliminaries

Tree-Based Speculative Decoding. Let $\mathcal{M}$ be the target model with vocabulary $\mathcal{V}$, and $p_{\mathcal{M}}(\cdot \mid c)$ its next-token distribution after a context c. For a prompt $x_{1:n}$, the *prefill* pass evaluates $p_{\mathcal{M}}(\cdot \mid x_{1:i})$

at every position i in parallel, after which autoregressive (AR) decoding emits one token per forward pass. Tree-based speculative decoding (Cai et al., 2024; Li et al., 2024b) instead drafts a small tree rooted at the last accepted token x_t, in which each node v carries a drafted token $x(v)$ and each root-to-leaf path is one guess of the next tokens. One *verification* pass of $\mathcal{M}$ with tree attention evaluates all nodes and yields at every node the target prediction $y(v)$, i.e., the token that $\mathcal{M}$ would emit after the path from the root to v. A path $(v_0, \ldots, v_L)$ is accepted up to depth a if each drafted token matches the prediction at its parent, and the step emits

$$l^\star = 1 + \max_{\text{paths}} \max\{a \le L : \; x(v_j) = y(v_{j-1}) \;\; \forall j \le a\} \tag{1}$$

tokens: the accepted drafts plus the target prediction at the last accepted node. Greedy outputs are thus identical to those of AR decoding, and sampling uses the standard verifier-side tree sampling (Li et al., 2024b; 2025). The drafter must therefore fill the tree cheaply with tokens that $\mathcal{M}$ will predict.

Draft Memory. The ideal draft at a context c is $\mathcal{M}$'s own prediction top-K $p_{\mathcal{M}}(\cdot \mid c)$, which would require the very forward pass that speculation aims to save. As a by-product of inference, however, $\mathcal{M}$ has already made such predictions at many *other* contexts: every prompt position during prefill and every tree node during verification. A draft memory $\mathbf{M}$ keeps them: under a key $\phi(c')$ that summarizes a past context c', it stores the top-K tokens predicted at c', and it drafts for a new context c by returning the entry with the same key,

$$D(c) = \mathbf{M}\big[\phi(c)\big] \in \mathcal{V}^K. \tag{2}$$

A static tree template $\mathcal{T}$ fixes only the *shape* of the tree: at run time, each node is looked up, and its children of rank $1, 2, \ldots$ receive the first, second, and later candidates returned. We use $K{=}8$ and a single template with 81 nodes and depth 8 throughout (Appendix A.5). Equation (2) leaves three design questions, answered in some way by every training-free drafter (§2): what forms the key (§3.2), which predictions are stored (§3.3), and how long they are kept (§3.4). SuffixSpec answers them with suffix-to-token keys, prefill and verification evidence, and two lifetimes.

3.2 Hierarchical Context Keys

Specificity versus Coverage. A natural key is a suffix of the context, i.e., its last few tokens. The last token alone recurs constantly but conflates contexts with different continuations; in code, for example, the token after an opening parenthesis depends on whether it follows `print` or `range`. A longer suffix such as "`range (`" separates these contexts but recurs far less often, especially early in a request. Short keys thus almost always hit but are imprecise, whereas long keys are precise but often miss, so no single key length is both precise and available (Figure 1a).

Suffix-to-Token Backoff. SuffixSpec therefore keeps two tables and tries the precise one first, as back-off n-gram language models do (Katz, 1987). The suffix-local table $\mathbf{M}^{\text{loc}}_{\text{suf}}$ is keyed by the suffix state $s_t = (x_{\max(1,\, t-m+1)}, \ldots, x_t)$, i.e., the last m tokens, and the token-local table $\mathbf{M}^{\text{loc}}_{\text{tok}}$ by the current token x_t. A lookup returns $\mathbf{M}^{\text{loc}}_{\text{suf}}[s_t]$ if this suffix has been observed and otherwise falls back to $\mathbf{M}^{\text{loc}}_{\text{tok}}[x_t]$. We use $m{=}3$ and fall back directly to a single token (§4.5), and we hash suffix states into $65{,}536$ buckets per final token, which keeps collisions below 2% (Appendix D).

Suffix States on the Draft Tree. Each draft-tree node v keeps its own suffix state $s(v)$: a child appends its token to its parent's state and drops the oldest token once the state holds m tokens. Each node is looked up with $s(v)$ and $x(v)$, so two nodes with the same token on different paths, such as the two parentheses above, can receive different candidates, which a token key cannot.

3.3 Filling the Memory

Verification Write-Back. Each verification pass evaluates $p_{\mathcal{M}}$ at every tree node, and these predictions are exactly what the verifier would accept at those contexts. SuffixSpec writes the top-K predictions at each node v, including nodes on rejected branches, into $\mathbf{M}^{\text{loc}}_{\text{suf}}[s(v)]$ and $\mathbf{M}^{\text{loc}}_{\text{tok}}[x(v)]$, overwriting older entries. Because the tree explores likely continuations of the current text, these writes fill the entries that upcoming steps are likely to query, with no extra forward pass.

Prefill Bootstrap. Write-back yields entries only once decoding has started, so the first steps of every request would draft from an empty memory (Figure 1b). Yet prefill has already computed $p_{\mathcal{M}}(\cdot \mid x_{1:i})$ at every prompt position, of which standard decoding keeps only the last. SuffixSpec writes top-K $p_{\mathcal{M}}(\cdot \mid x_{1:i})$ into $\mathbf{M}^{\mathrm{loc}}_{\mathrm{suf}}[s_i]$ and $\mathbf{M}^{\mathrm{loc}}_{\mathrm{tok}}[x_i]$ for every $i \leq n$, filling the memory before the first decoding step at the cost of one top-K over the prefill logits. The stored values are the model's predictions rather than prompt text, so they help even when the response does not copy the prompt; the gain is largest for short outputs, where the cold start is a large share of generation (§4.5).

3.4 Dual-Timescale Memory

Transferable and Request-Specific Entries. Entries differ in how well they carry over to a new request. A suffix entry records what followed one specific short context, such as a variable name and an operator: in a later request the same suffix rarely recurs, and when it does, it often sits in a different context and is followed by something else. A token entry summarizes the continuation of a single token, which depends far less on the request, since syntax, formatting, and function words behave alike across requests. The source matters in the same way: prefill predictions are made while reading one particular prompt, whereas decode-time predictions are made while writing a response, whose style and format recur across requests. Table 1 makes the first point concrete: suffix entries from earlier requests resolve only 17.0% of the lookups that miss the local tables, against 89.9% for token entries from earlier requests, and keeping them next to the token table lowers accepted length from 3.07 to 2.95 (§4.5), so their extra hits hurt more than they help. Suffix entries from the current request, by contrast, are correct 83.9% of the time (Appendix D), so they are worth keeping within the request.

Table 1: Entries from the current request versus earlier requests. *Hit*: share of the lookups reaching a table that it resolves.

Key	Entries from	Hit
Suffix	current request	23.8%
Suffix	earlier requests	17.0%
Token	earlier requests	89.9%

Local and Global Tables. SuffixSpec therefore gives each kind of entry its own timescale. The two local tables are cleared at the start of every request, so suffix and prefill entries never leak into other requests, and a token-global table $\mathbf{M}^{\mathrm{glo}}_{\mathrm{tok}} : \mathcal{V} \to \mathcal{V}^K$ that is never cleared keeps the transferable token-level entries. It completes the backoff chain:

$$\mathrm{Lookup}(s_t, x_t) = \begin{cases} \mathbf{M}^{\mathrm{loc}}_{\mathrm{suf}}[s_t], & \text{if } s_t \text{ has been observed in the current request,} \\ \mathbf{M}^{\mathrm{loc}}_{\mathrm{tok}}[x_t], & \text{else if } x_t \text{ has a local entry,} \\ \mathbf{M}^{\mathrm{glo}}_{\mathrm{tok}}[x_t], & \text{otherwise,} \end{cases} \tag{3}$$

with a fixed default entry if all three miss, which happens only for tokens never seen before. The global table comes last because its entries were made in other requests: whenever the current request already has an entry for the same token, that entry reflects the current context better. Because $\mathbf{M}^{\mathrm{glo}}_{\mathrm{tok}}$ stores one row of K tokens per vocabulary entry, its size is fixed and does not grow with traffic.

Verified-Only Commit. The global table is updated once, when a request ends: every entry of $\mathbf{M}^{\mathrm{loc}}_{\mathrm{tok}}$ that received a write-back during decoding is copied into $\mathbf{M}^{\mathrm{glo}}_{\mathrm{tok}}$, whereas entries written only by the prefill bootstrap are dropped. Dropping them keeps long prompts, which fill up to thousands of token entries with predictions about their own text, from overwriting the transferable entries left by earlier requests. To tell the two kinds apart, write-back adds the token to a pending set P and the bootstrap does not. The commit can wait until the request ends, since during decoding each written-back token already has a local entry that the lookup always consults first. Committing once per request also keeps $\mathbf{M}^{\mathrm{glo}}_{\mathrm{tok}}$ frozen while a request is being decoded (Algorithm 1).

3.5 Complete Procedure

Algorithm 1 puts the three parts together for one request; only the drafter differs from standard tree-based speculative decoding. Drafting and table updates take 0.22 ms per step versus 27.8 ms for a Llama-3.1-8B forward pass, and the bootstrap adds under 0.5% to prefill time (Appendix A.1).

Algorithm 1 SuffixSpec decoding for one request

Input: prompt $x_{1:n}$, target model $\mathcal{M}$, tree template $\mathcal{T}$, suffix length m, token-global table $\mathbf{M}_{\text{tok}}^{\text{glo}}$
// Before decoding: clear and bootstrap the local tables (§3.3)
$\mathbf{M}_{\text{suf}}^{\text{loc}},\ \mathbf{M}_{\text{tok}}^{\text{loc}} \leftarrow \varnothing$; pending set $P \leftarrow \varnothing$; $\text{logits}_{1:n} \leftarrow \mathcal{M}.\text{prefill}(x_{1:n})$
for $i = 1, \ldots, n$ **do** $\mathbf{M}_{\text{suf}}^{\text{loc}}[s_i],\ \mathbf{M}_{\text{tok}}^{\text{loc}}[x_i] \leftarrow \text{top-}K(\text{logits}_i)$ ▷ not added to P
// Decoding: draft, verify, accept, write back (§3.2, §3.3)
while not finished **do**
  Draft: expand $\mathcal{T}$ from x_t; each node v queries $\text{Lookup}(s(v), x(v))$ ▷ Eq. (3)
  Verify: one forward pass of $\mathcal{M}$ yields the top-K predictions at every node
  Accept: advance by $l^\star$ tokens ▷ Eq. (1)
  Write back: each node's top-K into $\mathbf{M}_{\text{suf}}^{\text{loc}}[s(v)]$ and $\mathbf{M}_{\text{tok}}^{\text{loc}}[x(v)]$; $P \leftarrow P \cup \{x(v)\}$ ▷ $\mathbf{M}_{\text{tok}}^{\text{glo}}$ frozen
end while
// After decoding: verified-only commit (§3.4)
for each token $x \in P$ **do** $\mathbf{M}_{\text{tok}}^{\text{glo}}[x] \leftarrow \mathbf{M}_{\text{tok}}^{\text{loc}}[x]$ ▷ prefill-only entries are dropped

4 Experiments

4.1 Evaluation Protocols

We use two evaluation protocols, each matching the setting that its baselines are designed for: running trained drafters outside their optimized engines would understate them, while porting every training-free baseline into a serving engine is beyond the scope of this paper.

Protocol I: Training-Free Comparison. Training-free drafters are mostly released as HuggingFace Transformers implementations, so all of them run on one shared stack (Transformers 4.46.2 (Wolf et al., 2020), `bfloat16`, batch size one) in which only the draft module differs. We use Llama-3.1-8B-Instruct (Team, 2024) and Qwen3-8B (Team, 2025) on one RTX 4090 (24 GB), and Qwen3-32B on one H20 (96 GB). The benchmark has 610 prompts (49 to 2,035 tokens, median 379) from seven tasks: code editing, text editing, math (GSM8K), summarization, translation, multi-turn dialogue (MT-Bench), and SPEED-Bench; Appendix B lists the sources. We generate up to 512 tokens with greedy decoding and with sampling (T=1.0, top-p=0.9). Baselines are PLD (Saxena, 2023), PLD+ (Somasundaram et al., 2025), Lookahead (Fu et al., 2024), REST (He et al., 2024), SAM (dynamic and static) (Hu et al., 2025), Suffix Decoding (Liu et al., 2024b), CopySpec (Dumitru et al., 2025), and TR (Luo et al., 2025), with their authors' recommended settings (Appendix A.2).

Protocol II: Comparison with Trained Drafters. We compare with EAGLE-3 (Li et al., 2025), DFlash (Chen et al., 2026), and DSpark (Cheng et al., 2026) on Qwen3-8B (Team, 2025). EAGLE-3, DFlash and DSpark use their public Qwen3-8B checkpoints. This comparison covers three trained draft-model families with different draft-generation mechanisms while keeping the target model and serving environment fixed. All methods, including AR, run in SGLang v0.5.14 (Zheng et al., 2024) with `bfloat16`, FlashInfer attention, and CUDA graphs on one H20 (96 GB). The workload is the throughput split of SPEED-Bench (Abramovich et al., 2026), built for system-level evaluation with fixed input-length buckets because serving inputs are often far longer than those of speculative-decoding benchmarks. We use the 2K and 8K buckets to contrast medium and long contexts, generate up to 512 tokens, and report Code, Creative writing, and Open-book QA.

Metrics and Output Consistency. We report throughput speedup over AR in the same stack: averaged over three runs under Protocol I (CV $< 0.5\%$), and cumulative decoded tokens over cumulative decode time under Protocol II. Greedy outputs match AR decoding up to floating-point nondeterminism; under sampling, SuffixSpec and AR reach comparable GSM8K solve rates (Llama-3.1-8B: 83.75% vs. 81.25%; Qwen3-8B: 90.00% vs. 91.25%). Since the global table persists across requests, we also shuffled the prompts: throughput changes by 0.4% (Appendix C.1).

4.2 Comparison with Training-Free Drafters

Greedy Decoding. Table 2 shows that SuffixSpec is the fastest method on every model and task, reaching $2.37\times$, $2.31\times$, and $1.96\times$ on Llama-3.1-8B, Qwen3-8B, and Qwen3-32B and improving

Table 2: Protocol I results under greedy decoding: speedup over AR per task (**SPEED**: SPEED-Bench long-generation subset), overall throughput and speedup, and average accepted length τ.

Model	Method	Speedup per task (×)							Overall		
		Code	Text	Math	Sum	Trans	MT	SPEED	Tok/s	Speedup	τ
Llama-3.1-8B	AR	1.00	1.00	1.00	1.00	1.00	1.00	1.00	46.13	1.00×	1.00
	PLD	1.59	1.56	1.32	1.41	1.23	1.22	1.30	64.72	1.40×	1.52
	PLD+	1.81	1.73	1.55	1.53	1.45	1.37	1.50	73.09	1.58×	1.89
	Lookahead	1.21	1.14	1.27	1.09	1.04	1.18	1.13	53.44	1.16×	1.35
	REST	1.27	1.15	1.21	1.18	1.32	1.30	1.21	56.15	1.26×	1.42
	SAM (dynamic)	1.86	1.76	1.57	1.66	1.48	1.44	1.57	75.84	1.64×	1.71
	SAM (static)	1.75	1.76	1.57	1.65	1.45	1.36	1.51	73.48	1.59×	1.73
	SuffixDecoding	1.82	1.53	1.51	1.38	1.53	1.32	1.38	68.67	1.49×	1.74
	CopySpec	1.59	1.52	1.29	1.41	1.38	1.29	1.41	66.45	1.44×	1.43
	TR	2.14	2.05	2.39	1.94	1.96	2.05	1.85	92.92	2.01×	2.77
	SuffixSpec	**2.63**	**2.40**	**2.63**	**2.29**	**2.14**	**2.27**	**2.23**	**109.32**	**2.37×**	**3.15**
Qwen3-8B	AR	1.00	1.00	1.00	1.00	1.00	1.00	1.00	49.98	1.00×	1.00
	PLD	1.66	1.62	1.37	1.30	1.03	1.18	1.25	69.35	1.39×	1.48
	PLD+	1.87	1.85	1.57	1.41	1.12	1.31	1.37	77.37	1.55×	1.81
	REST	1.30	1.14	1.24	1.20	1.27	1.27	1.21	60.94	1.25×	1.38
	SAM (dynamic)	1.85	1.79	1.58	1.47	1.16	1.33	1.45	78.78	1.58×	1.68
	SAM (static)	1.75	1.77	1.58	1.45	1.15	1.24	1.38	75.71	1.51×	1.69
	SuffixDecoding	1.93	1.50	1.50	1.26	1.06	1.19	1.30	71.12	1.42×	1.71
	CopySpec	1.67	1.54	1.30	1.30	1.11	1.21	1.37	70.72	1.41×	1.41
	TR	2.16	2.04	2.41	1.85	1.65	1.98	1.78	99.15	1.98×	2.69
	SuffixSpec	**2.61**	**2.38**	**2.68**	**2.16**	**1.75**	**2.20**	**2.10**	**115.31**	**2.31×**	**3.03**
Qwen3-32B	AR	1.00	1.00	1.00	1.00	1.00	1.00	1.00	18.40	1.00×	1.00
	PLD	1.45	1.58	1.40	1.26	1.06	1.18	1.11	24.16	1.31×	1.42
	PLD+	1.50	1.61	1.43	1.22	1.01	1.19	1.10	24.30	1.32×	1.72
	SAM (dynamic)	1.59	1.70	1.51	1.37	1.06	1.27	1.22	26.16	1.42×	1.58
	SuffixDecoding	1.74	1.49	1.47	1.23	1.05	1.22	1.17	25.10	1.37×	1.67
	CopySpec	1.44	1.48	1.25	1.20	1.03	1.17	1.18	23.81	1.29×	1.38
	TR	1.92	1.85	2.07	1.57	1.35	1.72	1.54	32.17	1.75×	2.33
	SuffixSpec	**2.24**	**2.12**	**2.32**	**1.87**	**1.44**	**1.89**	**1.72**	**36.11**	**1.96×**	**2.78**

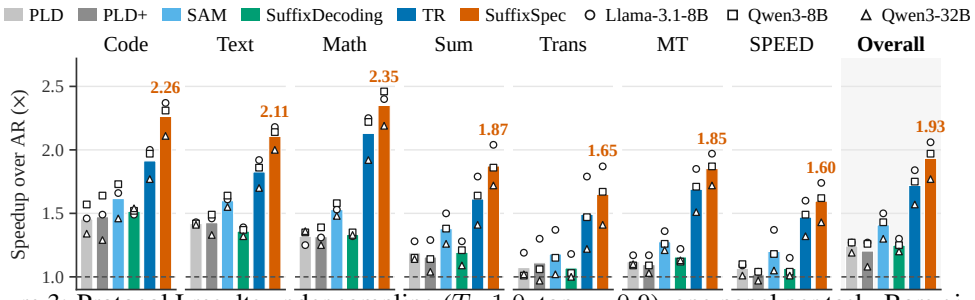


Figure 3: Protocol I results under sampling (T=1.0, top-p=0.9), one panel per task. Bars give the mean speedup over the three models and markers the individual models; the dashed line marks AR.

over the strongest baseline, TR, by 0.36×, 0.33×, and 0.21×. Its speedups peak on Code and Math (2.63× each on Llama-3.1-8B), and its largest margin over TR is on Code (+0.49×). Copy-based methods trail most on Translation and MT, where the output shares little text with the prompt, whereas drafting from target predictions keeps SuffixSpec's gains consistent across tasks. Its accepted length exceeds TR's by 12.6% to 19.3%, which accounts for most of the speedup gap.

Sampling. Figure 3 breaks the sampling results down by task. SuffixSpec remains the fastest method on every task and model, with overall speedups of 2.06×, 1.97×, and 1.77× on the three models. Its margin over TR is largest on Code and Text and smallest on Translation, MT, and SPEED, the tasks whose outputs depend least on the prompt; Appendix C.2 varies T.

Table 3: Comparison with trained drafters on Qwen3-8B. *Top:* greedy decode speedup over AR; *All* uses cumulative tokens and time. *Bottom:* deployment cost (extra peak GPU memory, or host memory for SuffixSpec) and sampling results at 8K-token input (T=1.0, top-p=0.9).

	Greedy, 2K-token input				Greedy, 8K-token input			
Method	Code	Creative	Open-book	All	Code	Creative	Open-book	All
EAGLE-3	2.05×	2.37×	1.82×	2.04×	2.14×	2.62×	1.77×	2.08×
DFlash	**4.23**×	**2.65**×	1.92×	**2.49**×	**3.75**×	2.39×	1.67×	2.24×
DSpark	3.63×	**2.65**×	1.73×	2.48×	2.83×	2.26×	1.64×	2.13×
SuffixSpec	3.14×	2.50×	**1.96**×	2.32×	3.28×	**2.95**×	**1.86**×	**2.45**×

	Deployment		Sampling, 8K-token input		
	Training	Extra memory	Acc. len.	Decode	End-to-end
EAGLE-3	✓	+1.98 GiB (GPU)	2.33	1.64×	1.56×
DFlash	✓	+4.91 GiB (GPU)	2.28	1.67×	1.59×
DSpark	✓	+2.94 GiB (GPU)	2.42	1.73×	1.64×
SuffixSpec	✗	≈10 MB (host)	**2.70**	**1.85**×	**1.71**×

4.3 Comparison with Trained Drafters

Context Length. On 8K-token inputs (Table 3), SuffixSpec achieves the highest overall greedy speedup (2.45×), ahead of DFlash (2.24×), DSpark (2.13×), and EAGLE-3 (2.08×). SuffixSpec is also fastest on Creative writing and Open-book QA, while DFlash remains fastest on Code. On 2K-token inputs, DFlash and DSpark are nearly tied overall at 2.49× and 2.48×, respectively; DFlash leads Code, DFlash and DSpark tie on Creative writing at 2.65×, and SuffixSpec is fastest on Open-book QA. From 2K to 8K, SuffixSpec's overall speedup rises from 2.32× to 2.45×, whereas EAGLE-3 stays nearly flat (2.04× to 2.08×), DFlash drops from 2.49× to 2.24×, and DSpark drops from 2.48× to 2.13×. This trend is consistent with longer prompts providing more target-conditioned evidence for SuffixSpec. SuffixSpec is thus most effective on long contexts while remaining competitive at 2K, without the extra GPU memory or per-target training of trained drafters. Medusa heads are public only for older models; on Mistral-7B-Instruct under Protocol I, SuffixSpec is faster than Medusa on every task (2.15× vs. 1.33× overall speedup; Appendix C.4).

Sampling. In Table 3, SuffixSpec achieves the highest decode speedup at 8K (1.85×, vs. 1.67× for DFlash and 1.64× for EAGLE-3), the highest end-to-end speedup (1.71×, vs. 1.59× and 1.56×), and the longest average accepted length (2.70 vs. 2.28 and 2.33). A verification step costs about 1.49 AR steps in SGLang, far below this length (Appendix C.3).

4.4 Concurrency and Model Scale

Concurrent Serving. Table 4 compares SuffixSpec with EAGLE-3, DFlash, and DSpark on Qwen3-8B as concurrency increases from 1 to 16. SuffixSpec leads at concurrency 1, 2, and 4, with speedups of 2.45×, 2.13×, and 1.96×, respectively. At concurrency 8, DFlash is slightly ahead at 1.79×, followed by SuffixSpec at 1.78×, EAGLE-3 at 1.76×, and DSpark at 1.52×. At concurrency 16, DFlash reaches 1.62×, compared with 1.53× for SuffixSpec, 1.49× for EAGLE-3, and 1.39× for DSpark. The decreasing speedups at higher concurrency are consistent with batching improving AR utilization and leaving less headroom for speculative decoding.

Table 4: Decode speedup over AR on Qwen3-8B under increasing concurrency in SGLang.

	Concurrency				
Method	1	2	4	8	16
AR	1.00×	1.00×	1.00×	1.00×	1.00×
SuffixSpec	**2.45**×	**2.13**×	**1.96**×	1.78×	1.53×
DFlash	2.24×	2.06×	1.92×	**1.79**×	**1.62**×
DSpark	2.13×	1.99×	1.77×	1.52×	1.39×
EAGLE-3	2.08×	1.94×	1.86×	1.76×	1.49×

Trained drafters also incur extra GPU-memory cost. The measured extra peak memory is +1.98 GiB for EAGLE-3, +4.91 GiB for DFlash, and +2.94 GiB for DSpark, whereas SuffixSpec keeps about 10 MB of draft state in host memory. EAGLE-3 and DFlash additionally maintain draft-side KV caches that grow with context length and concurrency. With Qwen3-8B's 8 KV heads of dimension 128, a draft layer stores $2 \times 8 \times 128 \times 2 = 4096$ bytes per token in BF16, so the one-layer EAGLE-3

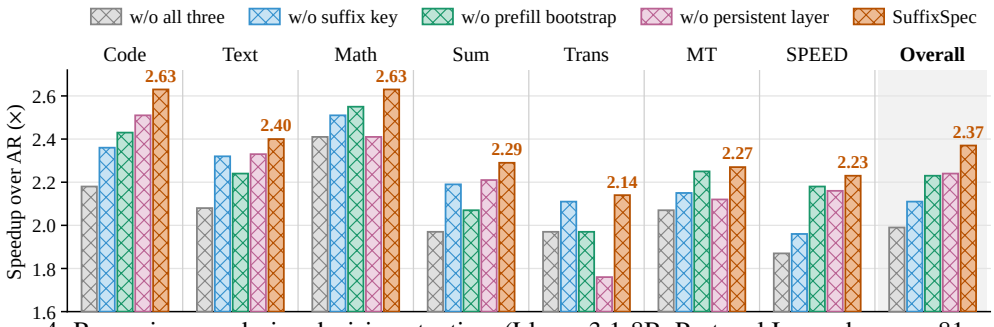


Figure 4: Removing one design decision at a time (Llama-3.1-8B, Protocol I, greedy, same 81-node tree); *Overall* is the end-to-end speedup, and the labels give SuffixSpec's values.

and five-layer DFlash drafters add about 4 and 20 KiB per token (Li et al., 2025; Chen et al., 2026): roughly 32 and 160 MiB per 8K-token request, or 0.5 and 2.5 GiB at concurrency 16 without prefix sharing, excluding allocator overhead and generated tokens. SuffixSpec needs no draft-side KV cache and thus avoids this overhead entirely.

Model Scale. On Llama-3.1-70B-Instruct with tensor parallelism over four H20 GPUs, SuffixSpec reaches a $2.42\times$ decode speedup over AR (Code $2.97\times$, Creative writing $2.22\times$, Open-book QA $1.88\times$, Sorting $2.98\times$) with the same tree and hyperparameters as for the 8B models.

4.5 Analysis

Design Decisions. In Figure 4, the suffix key matters most ($2.37\times \rightarrow 2.11\times$ overall), especially on Code and SPEED; the prefill bootstrap matters most on Summarization, Code, and Translation ($2.23\times$ without it); and the persistent layer matters most on Translation and Math ($2.24\times$ without it). Without all three, the memory reduces to a single token-keyed table filled during decoding ($1.99\times$); the individual drops sum to $0.53\times$, more than this $0.38\times$, so the decisions reinforce each other. Across about 8M lookups, the suffix-local, token-local, and token-global tables resolve 23.8%, 63.1%, and 12.0% of expansions, and only 1.2% reach the default entry (Qwen2.5-7B: $22.9/63.1/12.9/1.2\%$).

Granularity of Keys and Persistence. Among suffix lengths, $m{=}3$ is best ($2.37\times$, $\tau{=}3.07$): with $m{=}2$ the suffix loses specificity ($2.21\times$, $\tau{=}2.95$), and with $m{=}4$ it hits so rarely that fallback erases its precision ($2.05\times$, $\tau{=}2.97$). A three-level ladder ($3 \rightarrow 2 \rightarrow$ token) is slower than direct backoff (101.74 vs. 109.32 tok/s) with unchanged accepted length, because a two-token suffix resolves only 29.1% of three-token misses versus 97.7% for the token table. Keeping suffix entries across requests, in a persistent suffix table next to the token table, lowers accepted length from 3.07 to 2.95 and throughput by 7.9%, although it adds hits for 17.0% of the lookups that reach it (Table 1): its entries encode prompt-specific contexts such as variable names, so we keep only a token-level global table.

5 Conclusion

SuffixSpec treats the target model's own predictions as a hierarchical, dual-timescale memory: it keys them from suffixes down to tokens, fills them from prefill and verification, and keeps only decode-time predictions across requests. It is the fastest training-free drafter on seven tasks and three models and, without any training, outperforms EAGLE-3 and DFlash on long contexts. Because it only reuses predictions that inference already computes, it needs no draft model, adds only about 10 MB of host memory, and applies to any new target model without retraining.

Limitations. SuffixSpec is strongest at small batch sizes and on long contexts; DFlash is faster on 2K-token contexts and at concurrency 16, and all speculative methods lose ground as batching raises AR utilization. The memory offers no confidence signal and the tree is static, so some steps accept too few tokens to pay for verification; confidence-aware or load-aware tree budgets are a natural extension. The shared global table should be kept per tenant when requests carry sensitive data.

AI USE STATEMENT

In this work, we used generative AI tools solely to assist in searching for relevant baseline methods. We did not use generative AI tools for any other tasks. We manually reviewed the search results and checked them against the original sources. We take responsibility for the final content of this work, including any content informed by AI-assisted searches.

ETHICS STATEMENT

SuffixSpec changes only how draft tokens are proposed: every draft is checked by the target model, so greedy outputs are identical to autoregressive decoding and sampling preserves the target distribution. The token-global table is shared across requests; as noted in Section 5, deployments that serve sensitive data should keep it per tenant or per session so that no request can observe predictions made for another user. All benchmarks are public datasets, and no human subjects were involved.

REPRODUCIBILITY STATEMENT

Anonymized code is available at `https://anonymous.4open.science/r/suffixspec-8075`. Section 3 and Algorithm 1 specify the complete method. Appendix A gives the implementation details, the hyperparameters of every baseline, the construction of the SAM and REST datastores, the draft-tree search, and the Protocol II configuration; Appendix B describes the benchmark composition, and Appendix D the suffix hashing and its collision rate.

REFERENCES

Talor Abramovich, Maor Ashkenazi, Carl Putterman, Benjamin Chislett, Tiyasa Mitra, Bita Darvish Rouhani, Ran Zilberstein, and Yonatan Geifman. SPEED-Bench: A unified and diverse benchmark for speculative decoding. Technical report, NVIDIA, February 2026. URL `https://research.nvidia.com/publication/2026-02_speed-bench-unified-and-diverse-benchmark-speculative-decoding`. NVIDIA Research Publication.

agentlans. text-revision. `https://huggingface.co/datasets/agentlans/text-revision`, 2024. Hugging Face dataset.

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali K. Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL `https://openreview.net/forum?id=mtSSFiqW6y`.

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. 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*, pp. 12–58. The Association for Computer Linguistics, 2014. doi: 10.3115/V1/W14-3302. URL `https://doi.org/10.3115/v1/w14-3302`.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 5209–5235. PMLR / OpenReview.net, 2024. URL `https://proceedings.mlr.press/v235/cai24b.html`.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *CoRR*, abs/2302.01318, 2023. doi: 10.48550/ARXIV.2302.01318. URL `https://doi.org/10.48550/arXiv.2302.01318`.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *CoRR*, abs/2602.06036, 2026. doi: 10.48550/ARXIV.2602.06036. URL https://doi.org/10.48550/arXiv.2602.06036.

Yulong Chen, Yang Liu, Liang Chen, and Yue Zhang. Dialogsum: A real-life scenario dialogue summarization dataset. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, Findings of ACL, pp. 5062–5074. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.FINDINGS-ACL.449. URL https://doi.org/10.18653/v1/2021.findings-acl.449.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *CoRR*, abs/2607.05147, 2026. doi: 10.48550/ARXIV.2607.05147. URL https://doi.org/10.48550/arXiv.2607.05147.

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

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. *CoRR*, abs/2305.14233, 2023. doi: 10.48550/ARXIV.2305.14233. URL https://doi.org/10.48550/arXiv.2305.14233.

Zican Dong, Tianyi Tang, Junyi Li, Wayne Xin Zhao, and Ji-Rong Wen. BAMBOO: A comprehensive benchmark for evaluating long text modeling capacities of large language models. In Nicoletta Calzolari, Min-Yen Kan, Véronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pp. 2086–2099. ELRA and ICCL, 2024. URL https://aclanthology.org/2024.lrec-main.188.

Razvan-Gabriel Dumitru, Minglai Yang, Vikas Yadav, and Mihai Surdeanu. Copyspec: Accelerating llms with speculative copy-and-paste. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 26301–26332. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1337. URL https://doi.org/10.18653/v1/2025.emnlp-main.1337.

Min Fang, Zhihui Fu, Qibin Zhao, and Jun Wang. When, what, and how: Rethinking retrieval-enhanced speculative decoding. *CoRR*, abs/2511.01282, 2025. doi: 10.48550/ARXIV.2511.01282. URL https://doi.org/10.48550/arXiv.2511.01282.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 14060–14079. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

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. Codeeditorbench: Evaluating code editing capability of large language models. *CoRR*, abs/2404.03543, 2024. doi: 10.48550/ARXIV.2404.03543. URL https://doi.org/10.48550/arXiv.2404.03543.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. REST: retrieval-based speculative decoding. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (eds.), *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*, pp. 1582–1595. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.NAACL-LONG.88. URL https://doi.org/10.18653/v1/2024.naacl-long.88.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. SAM decoding: Speculative decoding via suffix automaton. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *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*, pp. 12187–12204. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.595/.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023. doi: 10.48550/ARXIV.2310.06825. URL https://doi.org/10.48550/arXiv.2310.06825.

Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoust. Speech Signal Process.*, 35(3):400–401, 1987. doi: 10.1109/TASSP.1987.1165125. URL https://doi.org/10.1109/TASSP.1987.1165125.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HklBjCEKvH.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *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*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/7b97adeafa1c51cf65263459ca9d0d7c-Abstract-Conference.html.

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. The stack: 3 TB of permissively licensed source code. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=pxpbTdUEpD.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, Proceedings of Machine Learning Research, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: speculative sampling requires rethinking feature uncertainty. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 28935–28948. PMLR / OpenReview.net, 2024a. URL https://proceedings.mlr.press/v235/li24bt.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: faster inference of language models with dynamic draft trees. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language*

Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024, pp. 7421–7432. Association for Computational Linguistics, 2024b. doi: 10.18653/V1/2024.EMNLP-MAIN.422. URL https://doi.org/10.18653/v1/2024.emnlp-main.422.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: scaling up inference acceleration of large language models via training-time test. *CoRR*, abs/2503.01840, 2025. doi: 10.48550/ARXIV.2503.01840. URL https://doi.org/10.48550/arXiv.2503.01840.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 31131–31146. PMLR / OpenReview.net, 2024a. URL https://proceedings.mlr.press/v235/liu24y.html.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 31131–31146. PMLR / OpenReview.net, 2024b. URL https://proceedings.mlr.press/v235/liu24y.html.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c16a5320fa475530d9583c34fd356ef5-Abstract-round1.html.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. Turning trash into treasure: Accelerating inference of large language models with token recycling. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *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*, pp. 6816–6831. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.338/.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 932–949, 2024.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 1797–1807. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1206. URL https://doi.org/10.18653/v1/d18-1206.

Project Gutenberg. Project Gutenberg. https://www.gutenberg.org, 1971. Accessed 2025.

Vipul Raheja, Dhruv Kumar, Ryan Koo, and Dongyeop Kang. Coedit: Text editing by task-specific instruction tuning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Findings of ACL, pp. 5274–5291. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.350. URL https://doi.org/10.18653/v1/2023.findings-emnlp.350.

Apoorv Saxena. Prompt lookup decoding. https://github.com/apoorvumang/prompt-lookup-decoding, 2023. GitHub repository.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pp. 1073–1083. Association for Computational Linguistics, 2017. doi: 10.18653/V1/P17-1099. URL https://doi.org/10.18653/v1/P17-1099.

Shwetha Somasundaram, Anirudh Phukan, and Apoorv Saxena. PLD+: accelerating LLM inference by leveraging language model artifacts. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, Findings of ACL, pp. 6075–6089. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-NAACL.338. URL https://doi.org/10.18653/v1/2025.findings-naacl.338.

Chendong Sun, Ali Mao, Lei Xu, and Mingmin Chen. Efficient adaptive rejection sampling for accelerating speculative decoding in large language models. *CoRR*, abs/2512.13194, 2025. doi: 10.48550/ARXIV.2512.13194. URL https://doi.org/10.48550/arXiv.2512.13194.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Jörg Tiedemann. Parallel data, tools and interfaces in OPUS. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis (eds.), *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, pp. 2214–2218. European Language Resources Association (ELRA), 2012. URL http://www.lrec-conf.org/proceedings/lrec2012/summaries/463.html.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pp. 38–45. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.EMNLP-DEMOS.6. URL https://doi.org/10.18653/v1/2020.emnlp-demos.6.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, and Fei Huang. Writingbench: A comprehensive benchmark for generative writing. *CoRR*, abs/2503.05244, 2025. doi: 10.48550/ARXIV.2503.05244. URL https://doi.org/10.48550/arXiv.2503.05244.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=N8N0hgNDRt.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 11263–11282, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh,

Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *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*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. SGLang: efficient execution of structured language model programs. 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*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

A Implementation Details

A.1 Memory and Time Overhead

For Llama-3.1-8B, the memory takes about 17 MB (two dense $|\mathcal{V}| \times K$ tables of 8.2 MB and 1.1 MB of buffers) plus a sparse suffix dictionary below 14 MiB even for 32K-token prompts; under Protocol I, drafting and table updates take 0.22 ms per step versus 27.8 ms for a target forward pass, and the bootstrap adds less than 0.5% to prefill time because it reuses the prefill logits.

A.2 Baseline Hyperparameters

Table 5 lists the hyperparameters of every Protocol I baseline. All methods share the same HuggingFace Transformers 4.46.2 inference stack; only the draft module differs between methods.

Table 5: Hyperparameters of the Protocol I baselines.

Method	**Configuration**
AR	HuggingFace `generate`, `do_sample=False`
PLD	n-gram $= 4$, $K = 16$
PLD+	`layer_idx` $= 9$, $K = 32$, `cos_threshold` $=$ None
REST	longest-suffix retrieval, $n \in [2, 16]$, tree-path verification (Appendix A.4)
SAM (dynamic)	`max_predicts` $= 40$, $\alpha = 4.0$, `len_bias` $= 5$
SAM (static)	same as dynamic, with a prebuilt static library (Appendix A.3)
Suffix Decoding	`max_depth` $= 64$, `min_token_prob` $= 0.1$, `use_cached_prompt` $=$ True
CopySpec	$\gamma = 3$, `speculate_k` $= 10$
TR	$K = 8$, `tree_v2.2.2`, `reset_per_prompt` $=$ False

A.3 SAM Static Library

The SAM static library follows a task-aligned but source-disjoint protocol: its data come from tasks that match our benchmark categories but are never drawn from the evaluation sets. The library is built from machine translation (Opus Books (Tiedemann, 2012), German to English, 20,000 examples), summarization (DialogSum (Chen et al., 2021), 10,000; XSum (Narayan et al., 2018), 10,000), mathematical reasoning (MetaMathQA (Yu et al., 2024), 20,000), text editing (CoEdIT (Raheja et al., 2023), 20,000), and code refinement (CodeXGLUE (Lu et al., 2021) Code Refinement small, 20,000), 100,000 task-aligned sequences in total. Every vocabulary token is also appended as a single-token sequence to guarantee full coverage. For Qwen3-8B (vocabulary size 151,665), this yields about 251,665 input sequences, a library with 17,938,921 suffix-automaton states, and 986 MB on disk.

A.4 REST Datastore

REST (He et al., 2024) uses a single datastore per model across all tasks, reflecting a deployment that does not switch retrieval corpora by task. To avoid contamination, the datastore sources are

disjoint from all evaluation sets: 80% of the tokens come from UltraChat 200K (Ding et al., 2023) and 20% from a Python subset of The Stack (Kocetkov et al., 2023). Separate datastores are built for Llama-3.1-8B and Qwen3-8B because their tokenizers differ; each takes about 6.7 GB on disk. Each datastore stores a shared token array, document-boundary arrays, and exact-match indices for spans of length $n \in \{2, \ldots, 16\}$. Retrieval uses longest-suffix fallback ($n_{\max}$=16, $n_{\min}$=2), and candidates are pruned by a trie (max_branching=4, max_leaves=16) before tree-path verification.

A.5 Draft Tree Search

We construct a 287-node mother tree with maximum depth 8 and collect SuffixSpec verification traces on a tree-search dataset that does not overlap with any evaluation benchmark. Because the accepted length of any connected subtree can be replayed from these traces without rerunning inference, we search offline for the connected 81-node subtree with the highest replayed mean accepted length, and derive nested connected subtrees for smaller verification budgets from this subtree.

A.6 Protocol II Configuration

All Protocol II runs use one NVIDIA H20 (96 GB) and SGLang v0.5.14 with `bfloat16`, TP=1, FlashInfer attention, decode CUDA graphs, and concurrency one unless stated otherwise; AR runs in the same engine with the same settings. EAGLE-3 and DFlash use their public Qwen3-8B drafters with default inference settings. Throughput is always cumulative tokens over cumulative latency.

A.7 Baseline Coverage

Lookahead, REST, and SAM (static) are omitted from the Qwen3-32B rows of Table 2. Preliminary runs on a subset of tasks show that they follow the same relative ranking as on the 8B models and trail stronger baselines already in the table, so including them would not change any conclusion.

B Benchmark Composition

Protocol I. The benchmark contains 610 prompts from seven tasks, each converted into an instruction-style prompt for chat models. *Multi-turn dialogue*: MT-Bench (Zheng et al., 2023) prompts evaluate open-ended instruction following. *Translation*: sentences from WMT14 De→En (Bojar et al., 2014) are translated into English. *Summarization*: CNN/DailyMail (See et al., 2017) articles are summarized. *Math reasoning*: GSM8K (Cobbe et al., 2021) word problems test multi-step reasoning. *Text editing*: agentlans/text-revision (agentlans, 2024) prompts ask the model to revise a text according to an instruction. *Code editing*: the Code Debug, Code Translate, and Code Polish subsets of CodeEditorBench (Guo et al., 2024). *Long-form generation*: a source-balanced subset of SPEED-Bench (Abramovich et al., 2026) with about 2K input tokens per prompt, drawn equally from Project Gutenberg (Project Gutenberg, 1971), WritingBench (Wu et al., 2025), BAMBOO MeetingPred, and BAMBOO ShowsPred (Dong et al., 2024). Input lengths range from 49 to 2,035 tokens, median 379, mean 481, and 95th percentile 1,187; per-task medians span 73 to 1,045.5.

Protocol II. The throughput split of SPEED-Bench (Abramovich et al., 2026) provides prompts in fixed input-length buckets from 1K to 32K tokens, grouped into low-, mixed-, and high-entropy domains, and targets system-level speedups in production engines. We use the 2K and 8K buckets and the Code, Creative-writing, and Open-book QA categories. Prompts whose greedy output degenerates into repetition are excluded, leaving the same 180 prompts for all methods.

C Additional Results

C.1 Prompt Order

Because the token-global table accumulates entries across requests, the order in which prompts arrive could affect the results. We fully shuffle all prompts across the seven tasks and rerun SuffixSpec on Llama-3.1-8B under Protocol I: throughput changes from 109.32 to 108.92 tok/s (0.4%), so the fixed task order of the benchmark has a negligible effect on all reported speedups.

C.2 Temperature Sensitivity

Table 6 varies the sampling temperature on Qwen3-8B under Protocol II (8K-token input). Accepted length declines gradually rather than collapsing: from $T{=}0$ to $T{=}1.3$ it decreases by about 15% ($2.85 \rightarrow 2.43$), and the decode speedup remains $1.66\times$ at $T{=}1.3$. The decode speedup nevertheless drops from $T{=}0$ to $T{=}0.5$ while accepted length is nearly unchanged. For every $T{>}0$, the ratio of decode speedup to accepted length stays nearly constant (0.68 to 0.69, versus 0.87 at $T{=}0$), which points to a fixed per-step cost of sampling-based verification rather than to worse drafts.

Table 6: Temperature sensitivity of SuffixSpec on Qwen3-8B (Protocol II, 8K-token input).

Temperature	Accepted length	Decode speedup
0.0	2.85	2.49×
0.5	2.84	1.96×
1.0	2.70	1.85×
1.3	2.43	1.66×

C.3 Verification-Step Profile in SGLang

NVTX profiling on Qwen3-8B in SGLang on one H20 (81-node tree, batch size one) measures an AR decode step at 12.4 to 12.5 ms and a SuffixSpec verification step at 18.57 ms, a ratio of about 1.49 that equals the breakeven accepted length. Target GEMMs and attention take 14.54 ms of the verification step, whereas draft lookup, preparation, and table update take 0.10, 0.31, and 0.06 ms; the GPU is busy about 87% of the decode time, so host-side drafting is not a bottleneck. Under sampling at 8K, the measured accepted length (2.70) is well above this breakeven point.

C.4 Comparison with Medusa

Public Medusa (Cai et al., 2024) heads are not available for our Protocol I models, so we compare on Mistral-7B-Instruct-v0.2 (Jiang et al., 2023) under the Protocol I stack (one RTX 4090, greedy, up to 1,024 new tokens). Medusa uses `posterior_threshold` $=0.09$, `posterior_alpha` $=0.3$, three heads with choice widths $[1, 1, 7, 6]$, and a KV cache length of 4096; SuffixSpec uses the same hyperparameters as in the main experiments ($K{=}8$, $m{=}3$). SuffixSpec is faster on every task, with an overall speedup of $2.15\times$ versus $1.33\times$ (Table 7). Medusa's gains concentrate on Code ($1.47\times$) and MT ($1.49\times$), whereas SuffixSpec improves all task types by at least $1.64\times$.

Table 7: Comparison with Medusa on Mistral-7B-Instruct, Protocol I stack (speedup over AR).

Method	Task speedup over AR							Overall	
	Code	Text	Math	Sum	Trans	MT	SPEED	Tok/s	Speedup
AR	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	46.35	1.00×
Medusa	1.47×	1.26×	1.47×	1.16×	1.17×	1.49×	1.24×	61.64	1.33×
SuffixSpec	**2.41**×	**2.26**×	**2.34**×	**2.22**×	**1.64**×	**2.03**×	**1.90**×	**99.57**	**2.15**×

D Suffix Hash Collision Analysis

Table 8: Suffix hash collision analysis ($m{=}3$, $B{=}65{,}536$).

Model	State coll. rate	Occ. coll. rate	Exact top-1	Hashed top-1
Llama-3.1-8B	1.90%	1.91%	84.49%	83.94%
Qwen3-8B	1.88%	1.90%	83.18%	82.64%

The suffix-local table is indexed by the composite key $\text{key} = x_t \times B + h(x_{t-m+1}, \ldots, x_t)$, where x_t is the current token, $B = 65{,}536$ is the bucket count, and h is a 64-bit FNV-style hash of the

last m token IDs. Because x_t is an explicit prefix, only suffixes that end in the same token can collide. Collisions are resolved by last-write-wins overwrite: a collision may make a later lookup return candidates from an unrelated suffix context, which can lower draft quality but cannot affect correctness, since the verifier checks every candidate. To quantify the effect, we reconstruct the full token sequences of the Protocol I benchmark and measure collision statistics offline (Table 8). The state-level collision rate is below 2% on both models, and the top-1 oracle accuracy (the fraction of suffix lookups whose stored top-1 candidate equals the true next token) drops by only about 0.55 points from a collision-free table to the hashed one, so the default bucket count suffices.