SPECSPLICE: DYNAMIC COMPOSITION OF PARAMETRIC AND RETRIEVAL-BASED DRAFTS FOR AGENTIC SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic workloads interleave novel text with repeated spans, making parametric and retrieval drafters complementary for speculative decoding. Within a single round, parametric draft tokens can expose retrieval matches unavailable from the committed context alone. Existing hybrids limit these opportunities through round-level source selection or prescribed attachment rules. We introduce SpecSplice, which jointly selects candidates from both sources under a shared node budget, treating the root and every node of a supplied parametric draft as potential splice points. A unified contribution score estimates each candidate's survival probability, accounting for the parametric prefix leading to a retrieved node. Score bounds enable a best-first algorithm to interleave retrieval with node selection, skipping queries that cannot affect the selected tree. We prove that the resulting tree maximizes the total contribution score over this candidate space under the node budget, exactly as if retrieval were queried at every splice point. Across four agentic workloads and three target models, SpecSplice achieves up to a $6.91\times$ speedup over autoregressive decoding and 13–37% higher throughput than the strongest baseline on every agentic model–workload pair.

1 INTRODUCTION

Large language models (LLMs) generate text autoregressively, so each new token requires a sequential forward pass of the target model. Speculative decoding (SD) alleviates this bottleneck: a lightweight drafter proposes future tokens, and the target model verifies them in parallel (Leviathan et al., 2023; Chen et al., 2023). Its speedup depends on two quantities: the number of draft tokens the target model accepts per verification round, and the cost of each round, which grows with the number of draft tokens it verifies (Miao et al., 2024; Chen et al., 2024).

The accepted length depends on the underlying drafter, which typically falls into either a parametric or retrieval paradigm. Parametric methods predict future tokens from learned parameters (Leviathan et al., 2023; Cai et al., 2024; Li et al., 2024a; 2025), ensuring broad applicability across any context; however, their agreement with the target model typically degrades with draft depth (Chen et al., 2026; Ringel & Romano, 2026). In contrast, retrieval-based methods reuse continuations observed in the prompt, prior turns, or other indexed sequences (Saxena, 2023; He et al., 2024; Zhao et al., 2024b; Hu et al., 2025; Oliaro et al., 2025). Although their generation terminates immediately upon sequence divergence, they can propose extensive, bursty continuations when the output repeats earlier text. Parametric drafting is therefore broadly applicable but typically shallow, whereas retrieval drafting is sparse but potentially deep (Liu et al., 2026).

These complementary mechanisms are particularly relevant to agentic workloads. An LLM agent solves a task over many turns: it reasons, invokes tools, and receives their results as context (Jimenez et al., 2024; Patil et al., 2025; Lei et al., 2025; Yao et al., 2025). The resulting trajectory interleaves newly generated content with spans that recur from earlier turns, as the agent revises earlier outputs, repeats tool-call formats, and quotes tool results. Crucially, the two regimes also intersect within a single round. When a round begins with novel text, retrieval may not find a match at the committed context; yet, a few parametric draft tokens can bridge the gap to a repeated span. Once retrieval is queried after these tokens, the span becomes reachable as a *draft-exposed match* and extends the

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.

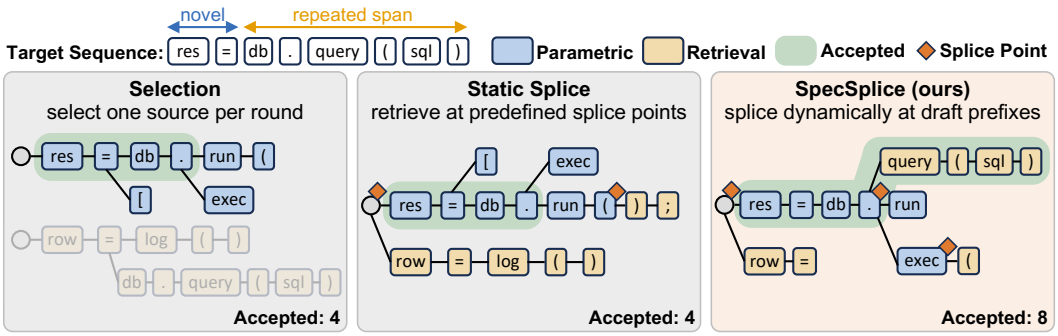


Figure 1: Conceptual comparison of hybrid drafting on the same target sequence. Selection uses one source per round, and static splicing restricts retrieval to predefined splice points. Under the same verification budget, SpecSplice dynamically retrieves draft-exposed matches at parametric prefixes and jointly selects candidates from both sources, yielding a longer accepted path.

draft (Figure 1). Exploiting such matches therefore requires composing both paradigms within the same verification round, rather than choosing one per round.

Existing hybrids, however, restrict how the two sources can be composed. Some select a single source for the entire round (Oliaro et al., 2025; Hu et al., 2025; Su et al., 2026), while others combine parametric and retrieved drafts through prescribed composition rules (Quan et al., 2025; Zhao et al., 2024a; Divilkovskiy et al., 2025; Shen et al., 2026). Yet different parametric prefixes can expose different retrieved continuations, and a retrieved node contributes only if the parametric prefix leading to it is accepted. These rules limit how the draft can adapt to the current round, as they do not jointly compare candidate contributions across sources.

Taking full advantage of these opportunities requires searching a much larger space of possible compositions between the two sources. Composing parametric prefixes with retrieved continuations of varying lengths creates many alternative draft paths, but verifying all of their nodes would increase the cost of each round. Under a shared node budget B, allocating more nodes to one continuation leaves fewer for alternatives from either source. Retaining the most valuable nodes therefore requires two components: a common score for comparing candidate nodes across sources, and a draft construction algorithm that uses these scores to select nodes within the budget.

We introduce SpecSplice (Figure 2), which selects candidates from both sources under a shared node budget B. The root and every node of the supplied parametric draft are potential *splice points* for attaching retrieved continuations. A *contribution score* ranks candidates on a common scale by estimating their survival probabilities, accounting for the survival of the parametric prefix leading to each retrieved node. Because scores do not increase along draft paths, the B highest-scoring candidates can be selected while retaining their required prefixes, and the score at each splice point bounds those of its retrieved candidates. These bounds allow SpecSplice to defer retrieval queries and skip those that cannot affect the selection; the resulting tree is identical to the one obtained by querying every splice point and maximizes the total contribution score under budget B.

Our contributions are:

- **Budgeted hybrid composition.** We formulate hybrid drafting as joint selection over a supplied parametric draft and retrieved continuations at the root and every parametric node, allowing draft-exposed matches to compete for the same node budget.
- **Unified scoring and interleaved construction.** We derive a prefix-aware contribution score for both sources and show that it is nonincreasing along every draft path in the hybrid candidate space. Consequently, a single best-first pass selects the B candidates that maximize the total score, querying retrieval only where it can affect this selection.
- **Evaluation.** On four agentic workloads with three target models under greedy decoding, SpecSplice achieves up to a $6.91\times$ speedup over autoregressive decoding and 13–37% higher throughput than the strongest baseline on every agentic model–workload pair, while remaining on par with or above the strongest baseline on the non-agentic Spec-Bench (+0.1–0.5%), where retrievable context is scarce.

2 Background and Motivation

2.1 Budgeted Draft Construction

In each verification round of speculative decoding (Leviathan et al., 2023; Chen et al., 2023), the target model accepts the drafted tokens up to the first rejected token and appends one *bonus token* sampled from its own distribution. Tree-based variants extend a single draft chain into multiple alternative draft paths, which are verified in one target pass under a topology-aware attention mask (Miao et al., 2024). Because verification cost grows with the number of tree nodes, construction is typically constrained by a node budget B. Each tree is rooted at a node v_0 that represents the committed context x; following Wang et al. (2025) and Ringel & Romano (2026), the root holds no draft token and is counted neither in the verification tree $\mathcal{T}$ nor toward B. A valid verification tree is ancestor-closed: including a node requires including all of its ancestors.

Let $S_{\mathcal{T}}(v)$ denote the survival probability of node v, i.e., the probability that v and all of its ancestors are accepted. If the accepted length $L(\mathcal{T})$ counts accepted draft tokens, excluding the bonus token, then

$$\mathbb{E}[L(\mathcal{T})] = \sum_{v \in \mathcal{T}} S_{\mathcal{T}}(v). \tag{1}$$

Budgeted draft construction therefore seeks an ancestor-closed set of at most B candidates with high total survival probability; because these probabilities are unknown before verification, practical methods rank candidates by drafter-derived surrogates (Chen et al., 2024; Wang et al., 2025; Li et al., 2024b). With a single drafter, all candidates share the same scoring semantics, making such budgeted ranking well defined.

2.2 Parametric and Retrieval Drafting

Parametric drafting. Parametric drafters predict candidate tokens from learned parameters. They include independent draft models (Leviathan et al., 2023; Chen et al., 2023), self-speculative and early-exit methods (Zhang et al., 2024; Elhoushi et al., 2024; Liu et al., 2024), auxiliary multi-token prediction heads (Stern et al., 2018; Cai et al., 2024; DeepSeek-AI et al., 2025), feature-conditioned drafters (Li et al., 2024a;b; 2025; Zhang et al., 2025), and parallel block predictors (An et al., 2026; Chen et al., 2026; Ringel & Romano, 2026). Their token probabilities provide learned predictive confidence for ranking candidates, but agreement with the target decays at deeper draft positions, so parametric drafts apply to any context yet typically remain shallow (Chen et al., 2026; Ringel & Romano, 2026).

Retrieval drafting. A retrieval drafter instead proposes continuations by matching a retrieval key against previously observed sequences. Existing methods retrieve from prompt or context n-grams (Saxena, 2023; Yang et al., 2023; Dumitru et al., 2025), suffix automata or trees (Hu et al., 2025; Oliaro et al., 2025), external datastores (He et al., 2024), or recycled token pools (Zhao et al., 2024b; Luo et al., 2025; Zhao et al., 2024a). Their evidence therefore reflects historical support for a matched continuation rather than learned predictive confidence. Importantly, the retrieval key need not end at the committed context: it can be extended by a parametric prefix, allowing different parametric paths to expose different retrieved continuations. Combining the two sources therefore breaks the single-drafter ranking of Section 2.1: their scores differ in semantics and scale, and the value of a retrieved candidate depends on the survival of its parametric prefix.

2.3 Hybrid Drafting and Admissible Splice Points

Hybrid methods combine parametric prediction and retrieval, but differ fundamentally in where retrieved continuations can be grafted onto a parametric path. We call the root or a parametric node a *splice point* when the committed context followed by the draft tokens on its path serves as the retrieval key and the returned retrieved continuation is attached there. Existing designs restrict these admissible splice points in different ways (Figure 1). Source-selection methods activate only one source for each round (Oliaro et al., 2025; Hu et al., 2025; Su et al., 2026); root-centered methods attach retrieval at the root or another predefined boundary (Quan et al., 2025; Shen et al., 2026); and other hybrids restrict retrieval to a chain tail, prescribed interior nodes, or a fixed structural layout (Zhao et al., 2024a; Divilkovskiy et al., 2025). Despite their operational differences, these

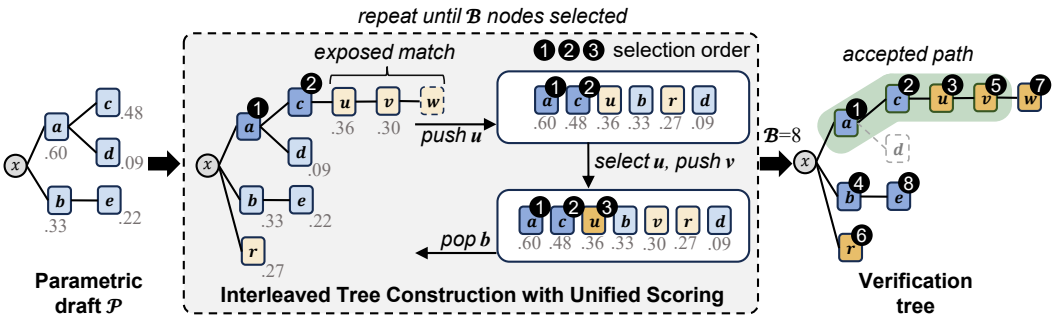


Figure 2: **Illustrative overview of SpecSplice.** Given a parametric draft $\mathcal{P}$, interleaved tree construction ranks candidates from both sources on a single scale and repeatedly selects the top one. Each selected draft prefix becomes a splice point whose retrieved continuation joins the same ranking and competes with the remaining parametric candidates, until the tree holds B nodes. Blue and orange denote parametric and retrieved nodes; badges give the selection order.

approaches constrain cross-source composition through source-level or structurally predefined rules rather than candidate-level scoring.

This restriction directly limits the candidate space because retrieval depends on the exact retrieval key. Different parametric prefixes can expose different continuations, including *draft-exposed matches* that are inaccessible from the committed context but become retrievable after particular parametric prefixes (Figure 1). If the node that exposes such a match is not an admissible splice point, its continuation never enters the candidate space and therefore cannot compete for the node budget, regardless of how accurately the exposed candidates are scored. SpecSplice instead treats every selected parametric node as a potential splice point, allowing the splice structure to emerge from candidate scores and budget allocation. The composition mechanism is not tied to a particular pair of drafters; our implementation instantiates the parametric and retrieval drafters with DFlash (Chen et al., 2026) and SuffixDecoding (Oliaro et al., 2025), respectively.

3 SpecSplice

SpecSplice first scores parametric and retrieved candidates on a common scale (Section 3.1) and then selects them under the node budget while querying retrieval on demand (Section 3.2).

3.1 Unified Contribution Scoring

Selecting candidates across sources requires a score for each candidate's contribution to the expected accepted length in Eq. (1), ideally its survival probability, which for a retrieved candidate includes the survival of the parametric prefix leading to its splice point. Because true survival is revealed only by verification after the tree is built, the contribution score estimates it beforehand from each drafter's evidence and places the estimates on a unified scale for direct cross-source ranking.

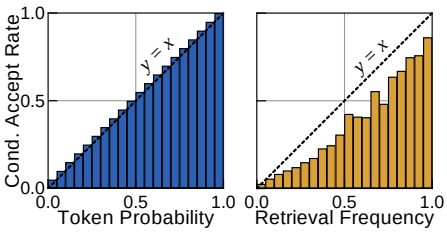
Node contribution. The ideal contribution of a candidate v is its survival probability $S_{\mathcal{T}}(v)$, defined in Section 2.1, since the expected accepted length in Eq. (1) sums these probabilities over the tree. Composing the two drafters yields three types of paths to account for: parametric-only paths, continuations retrieved at the root, and retrieved continuations appended to parametric paths. For a node v, let its splice point g be the last parametric node on the path from the root to v; thus $g = v$ if v is parametric, and $g = v_0$ if v lies in a continuation retrieved at the root (Section 2.3). Splitting the survival probability of each retrieved node at g (Appendix A.1), Eq. (1) decomposes as

$$\mathbb{E}[L(\mathcal{T})] = \sum_{\substack{v \in \mathcal{T} \\ \text{parametric}}} S_{\mathcal{T}}(v) \; + \sum_{\substack{v \in \mathcal{T} \\ \text{retrieved}}} S_{\mathcal{T}}(g)\, S_{\mathcal{T}}(v \mid g), \tag{2}$$

where, for a retrieved node, $S_{\mathcal{T}}(g)$ is the survival probability of its parametric prefix and $S_{\mathcal{T}}(v \mid g)$ is the probability of accepting its retrieved continuation through v given that g survives; $S_{\mathcal{T}}(v_0) = 1$ covers root retrieval. A retrieved node therefore contributes only if its parametric prefix survives.

Both factors, however, are determined by the acceptance decisions of the target model and are observed after verification, whereas the tree is constructed before it. SpecSplice instead estimates them from the evidence that each drafter provides before verification.

Estimating node survival. A node survives only when every node on its root-to-node path $\mathrm{Path}(v)$, which excludes the root, is accepted, so its survival probability is the product of the conditional acceptance probabilities along $\mathrm{Path}(v)$, each given acceptance of its ancestors (Appendix A.1). SpecSplice estimates each of these probabilities from the evidence supplied by the node's drafter. For parametric nodes, the parametric token probability provides an acceptance estimate (Li et al., 2024b); for a block drafter such as DFlash, it is the positionwise marginal of the factorized draft distribution rather than a probability conditioned on the preceding draft tokens (Ringel & Romano, 2026). For retrieved nodes, the estimate is the retrieval frequency, assuming that historically frequent continuations are more likely to be accepted (Oliaro et al., 2025). The acceptance estimates are therefore



(a) Parametric Drafter (b) Retrieval Drafter

Figure 3: Acceptance estimates versus conditional acceptance. The empirical acceptance rate conditioned on accepted ancestors, grouped by (a) parametric token probability and (b) retrieval frequency.

$$\widehat{a}_{\mathrm{p}}(u) = q_{\mathrm{p}}(u), \qquad \widehat{a}_{\mathrm{r},g}(u) = \frac{N_g(u)}{N_g(\mathrm{pa}(u))}, \tag{3}$$

where $q_{\mathrm{p}}(u)$ is the parametric token probability at node u, and $N_g(u)$ counts the matches of the retrieval key of g that continue through u, so the ratio is the fraction of matches reaching $\mathrm{pa}(u)$ that continue with the token at u. Figure 3 supports these choices: higher parametric token probabilities and retrieval frequencies generally correspond to higher conditional acceptance rates.

A unified contribution score. Construction depends on candidate rankings, but the two drafters' estimates have different relationships to empirical conditional acceptance (Figure 3). SpecSplice therefore adjusts their relative scale with a source weight $\lambda \in [0, 1]$. Applying this weight once to the retrieved continuation in Eq. (2) gives

$$\widehat{S}(v) = \underbrace{\prod_{u \in \mathrm{Path}(g)} \widehat{a}_{\mathrm{p}}(u)}_{\text{parametric prefix}} \cdot \underbrace{\lambda^{\mathbf{1}[v \neq g]} \prod_{u \in (g,v]} \widehat{a}_{\mathrm{r},g}(u)}_{\text{retrieved continuation}}, \tag{4}$$

where $(g, v]$ contains the nodes after g through v, and an empty product equals one. For $\lambda > 0$, the weight preserves the ordering within each source and adjusts only how the two sources interleave: as λ increases, retrieved candidates can only rise in rank relative to parametric ones, moving the budget split monotonically from the parametric-only tree ($\lambda \to 0$) to the unweighted combination of both sources ($\lambda = 1$; Appendix A.2). Node-wise or depth-wise reweighting would instead alter the ordering within the retrieval source; a depth-wise discount, in particular, would systematically penalize long continuations, which carry the value of draft-exposed matches. Section 3.2 uses the contribution scores, together with the upper bound $\lambda \leq 1$, to allocate the node budget across sources and splice points.

3.2 Interleaved Retrieval and Node Selection

Verification cost grows with the number of tree nodes, so the node budget limits the verification tree to B nodes (Section 2.1). SpecSplice takes as given the parametric candidate tree $\mathcal{P}$ supplied by the parametric drafter (Appendix B). Composing it with retrieval yields the candidate space $\mathcal{C}$, which contains $\mathcal{P}$ and the retrieved continuations obtainable at the root and at every node of $\mathcal{P}$ and typically exceeds the budget. Replacing the survival probabilities in Eq. (1) with the contribution scores of Eq. (4) (Section 3.1) gives the surrogate objective

$$\max_{\substack{\mathcal{T} \subseteq \mathcal{C},\ |\mathcal{T}| \leq B \\ \mathcal{T} \text{ prefix-distinct and ancestor-closed}}} \widehat{L}(\mathcal{T}), \qquad \widehat{L}(\mathcal{T}) = \sum_{v \in \mathcal{T}} \widehat{S}(v), \tag{5}$$

where $\mathcal{T}$ contains at most one candidate per draft prefix and ancestor-closure is over draft prefixes, as in Section 2.1.

Algorithm 1 Interleaved construction

Input: committed context x, parametric candidate tree $\mathcal{P}$, retrieval drafter, node budget B, source weight $0 \le \lambda \le 1$.
$z(v)$ is the draft prefix of v, with $z(v_0)$ empty and $\widehat{S}(v_0) = 1$; the roots of $\mathcal{P}$ and $\mathcal{R}_g$ correspond to v_0 and g. D maps each selected draft prefix to its node. **PushChildren**(o, u) queues each child candidate of o at its original score, with selected parent u; no duplicate check is performed here. For equal priorities, queries precede candidates and shallower candidates precede deeper ones.

$\mathcal{T} \leftarrow \varnothing$; $D \leftarrow \{z(v_0) \mapsto v_0\}$; $\mathcal{F} \leftarrow \{\textsc{Query}(v_0),\ \text{priority} = \lambda\}$
PushChildren$(\mathrm{root}(\mathcal{P}), v_0)$
while $|\mathcal{T}| < B$, $\mathcal{F} \neq \varnothing$, and $\mathrm{MaxPriority}(\mathcal{F}) > 0$ **do**
  $e \leftarrow \mathrm{PopMax}(\mathcal{F})$
  if $e = \textsc{Query}(g)$ **then**
    $\mathcal{R}_g \leftarrow \mathrm{Retrieve}(x \,\|\, z(g))$; **PushChildren**$(\mathrm{root}(\mathcal{R}_g), D[z(g)])$
  else e is candidate v
    if $z(v) \notin D$ **then** create node u for $z(v)$ under its selected parent; $D[z(v)] \leftarrow u$; $\mathcal{T} \leftarrow \mathcal{T} \cup \{u\}$
    else $u \leftarrow D[z(v)]$
    PushChildren(v, u)
    if v is a parametric candidate **then** queue $\textsc{Query}(v)$ with priority $\lambda\widehat{S}(v)$
return $\mathcal{T}$

First, the scores are nonincreasing across splice points. Within either source, extending a draft path multiplies its score by a factor in $[0, 1]$, so a descendant cannot outscore its ancestors (Wang et al., 2025; Ringel & Romano, 2026). Because a retrieved score includes the survival of its parametric prefix and $\lambda \le 1$, the next proposition shows that the same ordering holds across splice points, so top-B selection extends from a single source to both sources.

Proposition 1. *For $0 \le \lambda \le 1$, every node v of the retrieved continuation $\mathcal{R}_g$ returned by a query at splice point g satisfies*

$$\widehat{S}(v) \le \lambda\widehat{S}(g) \le \widehat{S}(g), \tag{6}$$

so contribution scores are nonincreasing along every path of $\mathcal{C}$. Consequently, the B highest-scoring candidates with distinct draft prefixes, with ancestors first on ties, form an ancestor-closed tree that maximizes Eq. (5)*.*

Proposition 1 reduces construction to selecting the B highest-scoring candidates, without enforcing ancestor-closure separately; the proof is given in Appendix A.2. The source weight thus serves two roles that point in the same direction: it corrects the relative scale of the two sources, and its bound $\lambda \le 1$ keeps the ordering intact at every splice point.

Proposition 2. *For $0 \le \lambda \le 1$, let t be the B-th highest score among the candidates of $\mathcal{C}$ with distinct draft prefixes. Leaving unqueried every splice point g with $\lambda\widehat{S}(g) < t$ does not change the tree of Proposition 1.*

Second, score bounds allow SpecSplice to defer or skip retrieval queries. Equation (6) bounds every candidate returned at splice point g by $\lambda\widehat{S}(g)$, so querying g cannot affect selections above this bound. Proposition 2 shows that retrieval therefore need not be queried at every splice point.

A query can thus be postponed while higher-scoring candidates are selected, and left unqueried if the budget fills first; the proof is given in Appendix A.2.

Interleaved tree construction. Building on these two propositions, Algorithm 1 implements construction with one priority queue that holds candidates at their contribution scores and pending queries at their bounds $\lambda\widehat{S}(g)$. Each selected parametric node registers a query, and each pop either selects the highest-scoring remaining candidate or issues the query with the highest bound, whose retrieved continuation then enters the queue. A candidate whose draft prefix is already selected is merged into the existing node, which inherits its children. Construction stops once B nodes are selected. In Figure 2, the query at c is issued before b is selected, so u enters the tree ahead of b, whereas the query at e never reaches the top. The resulting tree is the tree of Proposition 1, obtained without querying the splice points excluded by Proposition 2 (Appendix A.2).

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Table 1: **End-to-end decoding performance under greedy decoding.** Bold marks the best result for each target–workload pair; blue Δ rows report SpecSplice's speedup gain over the strongest baseline. Sampling results are in Appendix D. Suffix. denotes SuffixDecoding.

Method	SWE-bench Lite		Spider2-DBT		BFCL-v4		GAIA2		Spec-Bench	
	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT
Qwen3.5-9B										
DFlash	2.61×	4.72	3.62×	6.15	3.25×	5.28	3.09×	5.61	2.62×	4.98
Suffix.	2.90×	3.99	3.21×	4.31	2.74×	3.35	2.34×	3.15	1.34×	1.53
Graft	3.30×	6.09	4.28×	7.14	3.69×	5.83	3.00×	5.90	2.49×	4.74
Selection	3.54×	6.67	5.13×	7.74	3.89×	6.31	3.37×	6.51	2.63×	4.98
SpecSplice	**4.46×**	**8.90**	**6.42×**	**9.76**	**4.56×**	**7.52**	**3.80×**	**7.77**	**2.64×**	**4.99**
Δ vs. best	+26.0%		+25.3%		+17.2%		+12.8%		+0.2%	
Qwen3.5-27B										
DFlash	3.77×	5.58	4.67×	6.02	3.85×	5.49	4.12×	5.39	3.98×	5.70
Suffix.	3.25×	3.54	3.20×	4.04	2.94×	3.33	2.86×	3.37	1.27×	1.52
Graft	4.23×	6.61	5.32×	7.13	4.48×	5.99	4.38×	5.87	3.55×	5.10
Selection	4.35×	7.11	6.08×	7.97	4.41×	6.72	4.60×	7.07	3.95×	5.70
SpecSplice	**5.04×**	**8.26**	**6.91×**	**9.14**	**5.23×**	**7.92**	**5.61×**	**8.41**	**3.99×**	**5.71**
Δ vs. best	+15.8%		+13.6%		+16.6%		+21.9%		+0.1%	
Llama-3.1-8B										
DFlash	2.46×	3.96	2.75×	4.32	3.38×	5.14	2.44×	4.44	2.57×	5.30
Suffix.	4.38×	6.45	4.69×	7.20	4.52×	5.83	2.90×	3.65	1.52×	1.75
Graft	3.34×	5.51	3.54×	6.54	3.64×	5.57	2.86×	4.88	2.46×	5.07
Selection	4.20×	8.60	4.67×	10.52	4.48×	7.34	2.85×	5.59	2.59×	5.42
SpecSplice	**5.48×**	**10.89**	**5.57×**	**12.61**	**5.25×**	**8.73**	**3.98×**	**7.31**	**2.61×**	**5.58**
Δ vs. best	+25.2%		+18.8%		+16.3%		+37.1%		+0.5%	

Models and workloads. We evaluate SpecSplice with Qwen3.5-9B and Qwen3.5-27B in thinking mode (Qwen Team, 2026b;a), and Llama-3.1-8B-Instruct (Grattafiori et al., 2024), covering both model scale and family. The main agentic evaluation uses a common Claude Code harness (Anthropic, 2026) across four workloads: SWE-bench Lite (Jimenez et al., 2024) for software engineering, Spider2-DBT (Lei et al., 2025) for SQL workflows, BFCL-v4 (Patil et al., 2025) for tool use with memory and web search, and GAIA2 (Froger et al., 2026) for personal-assistant tasks in dynamic simulated environments. We additionally evaluate Spec-Bench (Xia et al., 2024) without an agent harness as a non-agentic control. The main results use greedy decoding, under which all methods follow the same target trajectory for each task; results under sampling and the benchmarks' official agent harnesses are provided in Appendices D and E.

Baselines. We compare against AR and four speculative baselines spanning the design space discussed in Section 2. *DFlash* (Chen et al., 2026), whose verification tree is constructed with DDTree (Ringel & Romano, 2026), represents parametric-only drafting. *SuffixDecoding* (Oliaro et al., 2025) represents retrieval-only drafting using per-request and global suffix trees. Among hybrid methods, *Selection* (Oliaro et al., 2025) chooses between the two sources at each round according to the retrieval draft's estimated accepted length, with DFlash as the parametric fallback, whereas *Graft* (Shen et al., 2026) combines them through root-centered retrieval after pruning the parametric tree. Together, these baselines isolate the distinction between single-source drafting, round-level source selection, and predefined splicing.

Implementation and protocol. Our default instantiation pairs DFlash with DDTree as the parametric backend and SuffixDecoding as the retrieval backend; Appendix B details their interfaces and the construction of the supplied parametric tree. Each method runs at its throughput-optimal node budget, with method-specific parameters—Selection's threshold τ and SpecSplice's source weight λ—selected over predefined grids; search ranges and selected values are reported in Appendix C. Unless stated otherwise, experiments use one NVIDIA RTX PRO 6000 with a single active request. An agent that enters a degenerate loop of repeated actions is terminated consistently across methods.

Metrics. Our primary metric is *speedup*, defined as decode throughput (output tokens per second) relative to AR on the corresponding trajectories. To expose the mechanism behind throughput, we also measure *MAT* (mean accepted tokens), the mean number of tokens committed per verification step including the bonus token.

4.2 End-to-End Decoding Performance

Consistent agentic gains. SpecSplice achieves the highest MAT and speedup in all 12 agentic model–workload pairs of Table 1, reaching a 6.91× speedup over AR and 13–37% higher throughput than the strongest baseline. The gains hold across software engineering, SQL, tool-use, and personal-assistant workloads, and its MAT improvements carry over to wall-clock speedup, so the opportunities exposed by dynamic composition repay their drafting and retrieval overhead.

The margin also holds across model scale and family, remaining double-digit on every agentic workload with Qwen3.5-27B. On Llama-3.1-8B, whose parametric drafter is weaker and whose retrieval is stronger, SuffixDecoding even outruns Selection despite a lower MAT, since its step carries no drafter pass; SpecSplice nevertheless remains fastest, accepting substantially more tokens per round at a per-step cost comparable to Selection's. On the non-agentic Spec-Bench, SpecSplice remains on par with or above the strongest baseline (+0.1–0.5%), incurring no throughput penalty when cross-source opportunities are absent.

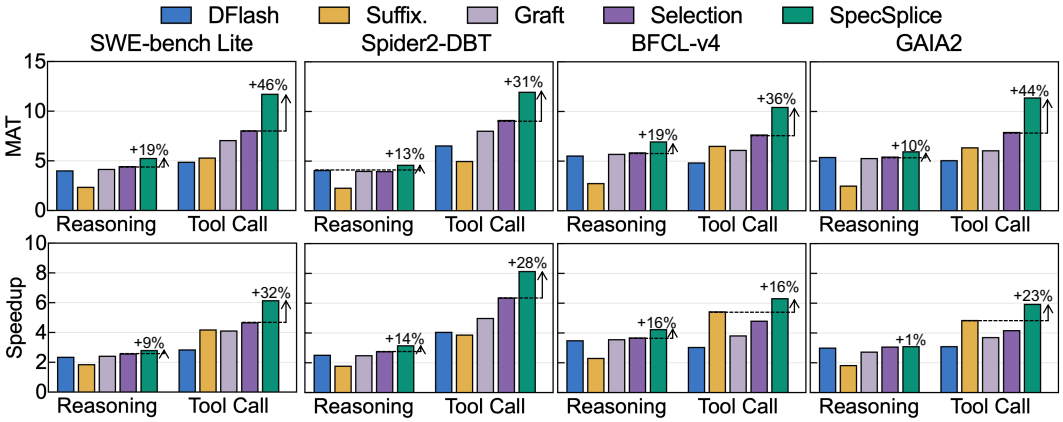


Figure 4: **Performance by trajectory segment.** MAT and speedup for reasoning and tool-call segments with Qwen3.5-9B under greedy decoding. Speedup is measured relative to AR on the corresponding segment; arrows show SpecSplice's gain over the strongest baseline.

Fine-grained complementarity within agentic trajectories. Figure 4 localizes these gains within the generated trajectory. The largest improvements arise in tool calls, where recurring call syntax, argument names, and boilerplate are interleaved with task-specific values that change from one invocation to the next. A parametric prefix can generate these novel values and thereby expose a previously observed continuation, after which retrieval efficiently extends the recurring structure. Reasoning exhibits the same mechanism at a smaller scale: new inferences alternate with restatements of prior observations and tool results, creating retrieval opportunities only after a novel parametric bridge. Accordingly, the two single-source drafters often favor different parts of the trajectory, whereas SpecSplice attains the highest MAT and speedup on both reasoning and tool-call segments across all workloads. This supports the central premise of SpecSplice: the useful boundary between parametric prediction and retrieval occurs within a verification round, rather than at the round level.

4.3 Mechanism Analysis

Unified scoring and splice-point coverage. Figure 5 isolates the two design choices behind dynamic composition. In Figure 5(a), retrieval is restricted to the root for all configurations, holding the retrieval location fixed while varying how candidates from the two sources compete for the budget. Unified contribution scoring reaches a MAT of 6.59, improving over Selection by 17.9% and Graft by 26%. The difference exposes the limitation of coarse source allocation: Selection excludes candidates from one source at the round level, while Graft reserves the released budget according

to its composition rule; unified scoring instead lets individual candidates compete according to their estimated survival contribution.

Figure 5(b) holds unified scoring fixed and varies only the admissible splice points. Allowing retrieval at every eligible parametric node raises MAT by 10.9% over root-only retrieval and by 53% over tail-only retrieval. Root-only retrieval cannot expose continuations whose matching keys emerge after speculative parametric tokens, while tail-only retrieval restricts access to a single deeper prefix whose continuation is useful only if that entire prefix survives. Dynamic splice points recover the intermediate *draft-exposed matches* missed by both restrictions. Together, unified scoring and fine-grained splice-point access improve MAT by 31% over Selection in this controlled setting, showing that both cross-source competition and candidate-space coverage contribute materially to the final tree.

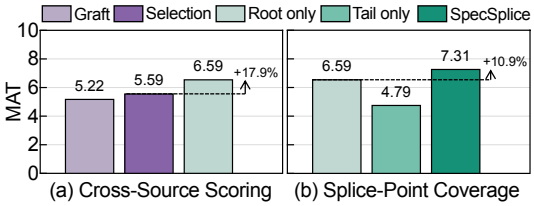


Figure 5: **Component ablation.** (a) Retrieval is fixed at the root, isolating cross-source candidate allocation. (b) Unified scoring is fixed while the admissible splice points are varied.

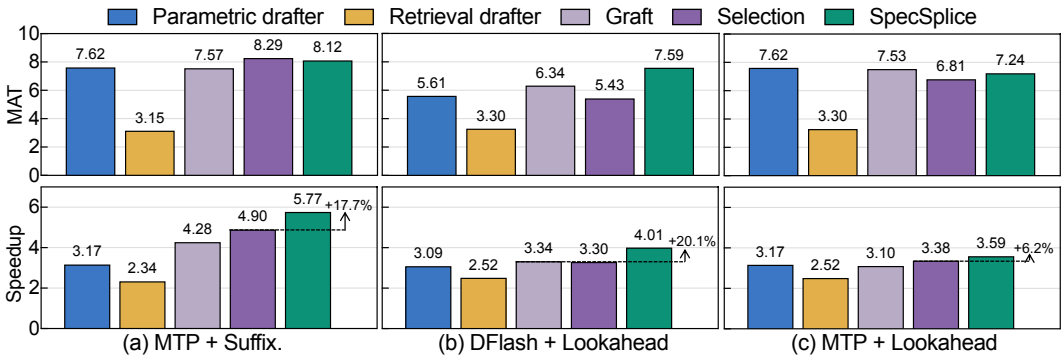


Figure 6: **Generality across drafting backends.** Qwen3.5-9B on GAIA2 under greedy decoding. Each panel replaces the parametric drafter, retrieval drafter, or both; all methods use the same backend pair and their throughput-optimal configurations.

Generality across drafting backends. The benefit of SpecSplice is not specific to its default DFlash–SuffixDecoding pair. Figure 6 replaces DFlash with MTP heads (DeepSeek-AI et al., 2025), SuffixDecoding with Lookahead n-grams (Zhao et al., 2024b), or both simultaneously, rebuilding every baseline with the same pair of drafters. SpecSplice remains the fastest method in all three configurations, indicating that the gain follows from how the two sources are composed rather than from a particular backend combination. The MTP experiments further highlight the distinction between accepted length and decoding efficiency: configurations using more autoregressive MTP steps can attain a higher MAT yet lower throughput because the additional drafting cost outweighs the accepted-token gain. SpecSplice reaches its throughput optimum with fewer MTP steps by extending selected parametric prefixes with retrieval, reinforcing the cost-aware objective established in Section 1.

5 Conclusion

We presented SpecSplice, which jointly selects parametric and retrieved candidates under a shared node budget. Retrieval at parametric draft prefixes exposes matches beyond those available from the committed context. Unified contribution scores guide interleaved tree construction, which maximizes the total score over the candidate space without querying every splice point. Experiments show gains in accepted length and throughput over single-source and hybrid baselines on agentic workloads, while retaining competitive throughput on Spec-Bench.

AI USE STATEMENT

The authors used large language models as general purpose assistants for prose editing, typesetting, and initial drafts of analysis code. The authors directed these uses and independently reviewed every assisted output. All technical ideas, derivations, proofs, and experimental designs are the authors' work. The authors verified all assisted text and code and take full responsibility for the manuscript. No AI system meets the criteria for authorship.

REPRODUCIBILITY STATEMENT

Section 3 specifies the drafting objective, scoring rules, and candidate-tree construction. Section 4 describes the evaluation protocol, baselines, metrics, and experimental configuration. Appendix A states the assumptions and provides the proofs of the theoretical results. Additional implementation and measurement details are provided in Appendix C.

REFERENCES

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. PARD: Accelerating LLM inference with low-cost PARallel draft model adaptation. In *International Conference on Learning Representations*, volume 2026, pp. 144562–144579, 2026.

Anthropic. Claude code overview. Claude Code Documentation, 2026. URL https://code.claude.com/docs/en/overview. Accessed September 22, 2026.

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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 2024.

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2026.

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, pp. 129531–129563. Curran Associates, Inc., 2024.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2025. URL https://arxiv.org/abs/2412.19437.

Maxim Divilkovskiy, Vitaly Malygin, Sergey Zlobin, Stanislav Ilyushin, Sultan Isali, Vasily Kalugin, Nuriza Aitassova, Fei Yi, and Weidi Zeng. READER: Retrieval-assisted drafter for efficient LLM inference. *arXiv preprint arXiv:2508.09072*, 2025. URL https://arxiv.org/abs/2508.09072.

Razvan-Gabriel Dumitru, Minglai Yang, Vikas Yadav, and Mihai Surdeanu. CopySpec: Accelerating LLMs with speculative copy-and-paste. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 26301–26332, Suzhou, China, November 2025. Association for Computational Linguistics.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642. Association for Computational Linguistics, August 2024.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. In *International Conference on Learning Representations*, volume 2026, pp. 119758–119789, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL `https://arxiv.org/abs/2407.21783`.

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

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. SAM decoding: Speculative decoding via suffix automaton. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12187–12204, Vienna, Austria, July 2025. Association for Computational Linguistics.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations*, volume 2025, pp. 28691–28735, 2025.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, Miami, Florida, USA, November 2024b. Association for Computational Linguistics.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, pp. 136737–136756. Curran Associates, Inc., 2025.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating LLMs via double early exiting. In *Advances in Neural Information Processing Systems*, volume 37, pp. 11946–11965. Curran Associates, Inc., 2024.

Xiaoxuan Liu, Jiaxiang Yu, Jongseok Park, Ion Stoica, and Alvin Cheung. Speculative decoding: Performance or illusion? In *Proceedings of Machine Learning and Systems*, volume 8, pp. 1719–1741. MLSys, 2026.

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 *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6816–6831, Vienna, Austria, July 2025. Association for Computational Linguistics.

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, Volume 3*, ASPLOS '24, pp. 932–949, New York, NY, USA, 2024. Association for Computing Machinery.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. SuffixDecoding: Extreme speculative decoding for emerging AI applications. In *Advances in Neural Information Processing Systems*, volume 38, pp. 126326–126354. Curran Associates, Inc., 2025.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025.

Guofeng Quan, Wenfeng Feng, Chuzhan Hao, Guochao Jiang, Yuewei Zhang, and Hao Henry Wang. RASD: Retrieval-augmented speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6167–6177, Vienna, Austria, July 2025. Association for Computational Linguistics.

Qwen Team. Qwen3.5-27B model card. Hugging Face, 2026a. URL `https://huggingface.co/Qwen/Qwen3.5-27B`. Accessed September 8, 2026.

Qwen Team. Qwen3.5-9B model card. Hugging Face, 2026b. URL `https://huggingface.co/Qwen/Qwen3.5-9B`. Accessed September 8, 2026.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. In *Third Conference on Language Modeling*, 2026.

Apoorv Saxena. Prompt lookup decoding. GitHub, 2023. URL `https://github.com/apoorvumang/prompt-lookup-decoding`. Accessed September 22, 2026.

Yuhao Shen, Tianyu Liu, Xinyi Hu, Quan Kong, Baolin Zhang, Jun Dai, Jun Zhang, Shuang Ge, Lei Chen, Yue Li, Mingcheng Wan, and Cong Wang. Draft less, retrieve more: Hybrid tree construction for speculative decoding. *arXiv preprint arXiv:2605.20104*, 2026. URL `https://arxiv.org/abs/2605.20104`.

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

Xin Su, Dawid Majchrowski, Fangyuan Yu, Vanshil Atul Shah, Sebastian Rogawski, Pawel Morkisz, Anahita Bhiwandiwalla, and Phillip Howard. Hybrid verified decoding: Learning to allocate verification in speculative decoding. *arXiv preprint arXiv:2606.01019*, 2026. URL `https://arxiv.org/abs/2606.01019`.

SWE-agent Team. mini-swe-agent: The minimal AI software engineering agent. GitHub repository, 2026. URL `https://github.com/SWE-agent/mini-swe-agent`. Accessed September 8, 2026.

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

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

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, volume 2025, pp. 9965–10017, 2025.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025.

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 (Volume 1: Long Papers)*, pp. 11263–11282. Association for Computational Linguistics, August 2024.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. In *International Conference on Learning Representations*, volume 2025, pp. 35367–35388, 2025.

Weilin Zhao, Yuxiang Huang, Xu Han, Wang Xu, Chaojun Xiao, Xinrong Zhang, Yewei Fang, Kaihuo Zhang, Zhiyuan Liu, and Maosong Sun. Ouroboros: Generating longer drafts phrase by phrase for faster speculative decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 13378–13393, Miami, Florida, USA, November 2024a. Association for Computational Linguistics.

Yao Zhao, Zhitian Xie, Chen Liang, Chenyi Zhuang, and Jinjie Gu. Lookahead: An inference acceleration framework for large language model with lossless generation accuracy. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 6344–6355, New York, NY, USA, 2024b. Association for Computing Machinery. ISBN 9798400704901.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024.

A Acceptance Identities and Selection Guarantees

A.1 Survival factorization

For a fixed committed context, verification tree, and verification rule, a node is accepted only if all of its ancestors are accepted. Successive conditioning along $\mathrm{Path}(v)$ therefore expresses $S_{\mathcal{T}}(v)$ as the product of the conditional acceptance probabilities used in Section 3.1; no independence assumption is needed. Because the accepted length counts the nodes on the accepted path, linearity of expectation gives Eq. (1). Splitting the product at the splice point g of a retrieved node gives $S_{\mathcal{T}}(v) = S_{\mathcal{T}}(g)\, S_{\mathcal{T}}(v \mid g)$, and substituting this factorization into Eq. (1) gives Eq. (2).

Under greedy verification, acceptance is deterministic for a fixed context; the empirical rates in Figure 3 average outcomes across contexts and rounds. Under sampled verification, the conditional probabilities also depend on the verification rule, including residual resampling (Leviathan et al., 2023) and the order in which sibling candidates are tried; as in prior tree construction, the contribution scores ignore this dependence. Parametric token probabilities and retrieval frequencies estimate these conditional quantities before verification. The guarantee below concerns the resulting surrogate objective; losslessness follows from the unchanged verification procedure.

A.2 Optimality of interleaved construction

Fix the parametric candidate tree $\mathcal{P}$ supplied for a round, the scores of its nodes, and the retrieval index with its return policy. Let $\mathcal{C}$ contain $\mathcal{P}$ and every retrieved continuation obtainable by querying the root or a node of $\mathcal{P}$; retrieved nodes are not splice points. Each candidate keeps its score from Eq. (4). Within either source, extending a draft path multiplies the score by a parametric token probability or a retrieval frequency, both in $[0, 1]$. A retrieved candidate $v \in \mathcal{R}_g$ scores $\lambda\widehat{S}(g)$ times a product of retrieval frequencies, so $\widehat{S}(v) \le \lambda\widehat{S}(g) \le \widehat{S}(g)$ for $0 \le \lambda \le 1$, which is Eq. (6). Scores are therefore nonincreasing along every path of $\mathcal{C}$.

The two sources may propose the same draft prefix, and Algorithm 1 does not score such a prefix separately: a popped candidate whose draft prefix is already selected reuses the existing node and pushes its children under it (lines 8–10). For analysis, consider a scan of all candidates in nonincreasing score order, with ancestors first on ties, that keeps a candidate only if its draft prefix has not been kept before; each kept prefix carries the score of the candidate that first reaches it. Because the parent prefix of every candidate is proposed by a candidate with at least its score, this handling preserves the ordering along paths. Every ancestor of a prefix scores at least as high and precedes it on ties, so the B highest-scoring prefixes contain the ancestors of their members; and because the objective sums nonnegative scores, no ancestor-closed set of at most B prefixes has a larger sum. This proves Proposition 1.

Proof of Proposition 2. By Eq. (6), every candidate retrieved at a splice point g with $\lambda\widehat{S}(g) < t$ scores below t, and so does every descendant it proposes. None of these candidates is among the B highest-scoring distinct draft prefixes, and a draft prefix they share with another candidate is kept at that candidate's score. Removing them therefore leaves the tree of Proposition 1 unchanged.

Interleaved tree construction returns the tree of Proposition 1. The argument relies on one property of Algorithm 1: every popped candidate pushes all its children at their original scores. First, every unprocessed candidate in $\mathcal{C}$ is bounded by a queued entry. An unqueued parametric descendant is bounded by its first unprocessed parametric ancestor, which also bounds retrieval below any later splice point on that path. Once a splice point is processed, its pending query is represented by $\lambda\widehat{S}(g)$. After retrieval, an unqueued retrieved descendant is bounded by the first unprocessed candidate on its retrieved continuation. Popping an entry pushes only entries of no greater priority. Consequently, candidates are processed in nonincreasing score order, even when retrieval is interleaved with selection.

For a draft prefix already selected, the algorithm reuses the existing node but still pushes the popped candidate's children at their original scores, and a parametric candidate still registers its query, including when its prefix was first selected through retrieval. Thus neither merging nor reuse removes an eligible continuation or changes its score. Distinct draft prefixes are selected in the order of the

reference scan, with possible alternatives on ties, and their parents have already been selected because children are pushed only after their originating parents and retrieved children only after their splice points. The selected nodes therefore form an ancestor-closed tree whose score sum equals that of the B largest distinct prefix scores, or of all positive-score prefixes if fewer exist, which is the maximum of Eq. (5). Finally, suppose the algorithm selects B nodes, and let t be the score of the B-th. Entries are popped in nonincreasing priority, and construction stops once the B-th node is selected, so a pending query whose bound $\lambda\widehat{S}(g)$ is below t is still queued when construction stops and is never issued. Every candidate such a query could return scores at most its bound, hence below t, and could not have entered the tree, consistent with Proposition 2.

The result concerns the fixed surrogate over the candidate space induced by the supplied $\mathcal{P}$.

Monotonicity in the source weight. Fix $\mathcal{C}$ and, for simplicity, assume distinct draft prefixes and no ties. Increasing λ multiplies every retrieved score by the same factor and leaves every parametric score unchanged, so the ordering within each source is preserved and a retrieved candidate can only move ahead of parametric candidates. A retrieved candidate among the B highest-scoring candidates therefore remains among them, and the number of retrieved nodes in the tree of Proposition 1 is nondecreasing in λ. Since every retrieved score is at most λ, taking λ below the smallest positive parametric score places every retrieved candidate after every such parametric candidate, so the tree approaches the B highest-scoring nodes of $\mathcal{P}$ as $\lambda \to 0$; at $\lambda = 1$, both sources are ranked by their unweighted path products.

B Drafter Interfaces

SpecSplice requires two interfaces: a parametric drafter that supplies a candidate tree with parametric token probabilities, and a retrieval drafter that returns, for a retrieval key, a continuation tree with match counts. Composition operates only on these outputs.

B.1 Parametric drafters

The parametric drafter supplies a candidate tree $\mathcal{P}$ under its own generation policy; its depth, width, and pruning determine candidate coverage and drafting cost, whereas the node budget B limits the verification tree. How $\mathcal{P}$ is formed depends on whether node scores are available without further drafter calls. A block drafter such as DFlash predicts all draft positions in one forward pass, so the score of every candidate is a product of positionwise probabilities and the B highest-scoring candidates can be enumerated directly (Ringel & Romano, 2026). For such a $\mathcal{P}$, keeping B nodes loses nothing. Let t be the B-th score in $\mathcal{P}$. Every parametric candidate outside $\mathcal{P}$ scores at most t, and so does every retrieved candidate spliced below such a candidate by Eq. (6); since $\mathcal{P}$ already contains B distinct draft prefixes with scores at least t, adding these candidates cannot increase the maximum of Eq. (5). Path-conditioned drafters, such as EAGLE-style feature drafters (Li et al., 2024a; 2025) and sequential multi-token prediction heads (DeepSeek-AI et al., 2025), condition each prediction on the preceding draft token, so the children of a node are scored only after the node is expanded. Their trees are built by dynamic expansion, e.g., expanding the highest-scoring nodes layer by layer and keeping the highest-scoring nodes overall (Li et al., 2024b; Wang et al., 2025), with $\widehat{a}_{\mathrm{p}}(u)$ given by the parametric token probability at each expanded node. Propositions 1 and 2 therefore hold relative to the supplied tree: with a block drafter, $\mathcal{P}$ contains the highest-scoring candidates under the drafter's distribution, whereas with a path-conditioned drafter it reflects the drafter's expansion policy.

B.2 Retrieval drafters

The retrieval drafter takes the retrieval key of a splice point g, i.e., the committed context followed by the draft tokens on $\mathrm{Path}(g)$, and returns a retrieved continuation $\mathcal{R}_g$ with match counts $N_g(u)$. The retrieval frequency in Eq. (3) is the fraction of matches reaching $\mathrm{pa}(u)$ that continue with the token at u; its product from g through v equals $N_g(v)/N_g(g)$, so path scores follow directly from the counts. Extending a draft prefix cannot increase its match count, which gives nonincreasing scores. Empty matches return no candidates, and frequencies are not renormalized over the children

that the index returns. SpecSplice multiplies each retrieval path score by $\widehat{S}(g)$ and applies λ once, as in Eq. (4). Any retrieval drafter that exposes such counts can be used.

SuffixDecoding. The reported instantiation uses the SuffixDecoding index (Oliaro et al., 2025), which maintains a per-request suffix tree over the current request's prompt and output and a global suffix tree over generated responses. For a retrieval key, each tree builds a continuation for every suffix of the key that it matches, up to a maximum depth, and keeps the one with the highest score, i.e., the sum of its node probabilities; the higher-scoring continuation of the two trees is returned. The counts N_g of a returned continuation therefore come from a single tree and count the occurrences of a single matched suffix. The index's own limits on continuation length and branching define the return policy fixed in Appendix A.2.

Lookahead. For the retrieval-drafter ablation, we use the n-gram drafter of SGLang, which follows the n-gram pool of Lookahead decoding (Zhao et al., 2024b) and indexes completed outputs in a single global trie; prompts are not inserted. It matches the longest trailing token sequence of the retrieval key present in the trie and builds the draft from the tokens that follow. Its default breadth-first construction discards frequencies once the tree is built; we instead use its probability-ordered mode, which expands the most frequent continuations first, and modify its output to return the frequency-based probability of each node alongside its token and parent, which serves as the retrieval frequency.

C Experimental Details

Environment. Unless stated otherwise, all runs use SGLang (Zheng et al., 2024) on a single NVIDIA RTX PRO 6000, with bfloat16 weights and KV cache and the FlashInfer attention backend (Ye et al., 2025). All drafters are registered as SGLang plugins. The parametric drafters are the public DFlash (Chen et al., 2026) checkpoints (`z-lab/Qwen3.5-9B-DFlash`, `z-lab/Qwen3.5-27B-DFlash`, `z-lab/LLaMA3.1-8B-Instruct-DFlash-UltraChat`), used without retraining, with block length 16 for Qwen and 10 for Llama. Our adapter routes requests from Claude Code (Anthropic, 2026) to the SGLang endpoint. The evaluation functions of BFCL-v4 and GAIA2 are exposed to the agent as Model Context Protocol (MCP) tools. For BFCL-v4, the prerequisite tasks that populate the agent's memory are executed but excluded from all metrics.

Selected hyperparameters. Selection's threshold τ and SpecSplice's source weight λ are selected per target and workload, jointly with each method's node budget, for the highest decode throughput over $\tau \in \{0.5, 1, 1.5, 2, 3, 4, 5, 6, 8, 10, 12, 16, 24, 32\}$ and $\lambda \in \{0.001, 0.005, 0.01, 0.02, 0.05, 0.08, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.7, 1.0\}$. Table 2 lists the selected values for the main evaluation (Table 1).

Table 2: Selected hyperparameters: Selection threshold τ^* and SpecSplice source weight λ^*.

Target	Workload	τ^*	λ^*
Qwen3.5-9B	SWE-bench Lite	12	0.15
	Spider2-DBT	16	0.1
	BFCL-v4	16	0.1
	GAIA2	12	0.08
	Spec-Bench	32	0.005
Qwen3.5-27B	SWE-bench Lite	10	0.1
	Spider2-DBT	10	0.1
	BFCL-v4	12	0.08
	GAIA2	8	0.15
	Spec-Bench	32	0.005
Llama-3.1-8B	SWE-bench Lite	8	0.3
	Spider2-DBT	5	0.4
	BFCL-v4	8	0.3
	GAIA2	10	0.3
	Spec-Bench	24	0.02

D RESULTS UNDER SAMPLING

Under greedy decoding all methods follow the same trajectory per task. Table 3 removes this control: each method generates its own trajectories at T=0.6 with the sampling parameters Qwen3.5 recommends for coding in thinking mode (top-p 0.95, top-k 20, min-p 0, no presence or repetition penalty), with the same loop termination as in Section 4.1 applied to every method, so the trajectories and the retrieval indices built from them differ across methods. SpecSplice has the highest MAT and speedup in every agentic cell.

Table 3: **Results under sampling** (T=0.6). Speedup is relative to AR; Suffix.: SuffixDecoding.

Method	SWE-bench Lite		Spider2-DBT		BFCL-v4		GAIA2		Spec-Bench	
	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT
Qwen3.5-9B										
DFlash	3.93×	6.15	4.73×	7.33	3.45×	4.99	3.27×	5.18	3.75×	4.63
Suffix.	2.83×	3.98	4.47×	5.72	2.88×	3.47	2.69×	3.36	1.71×	1.51
Graft	3.69×	6.04	4.48×	6.67	3.66×	5.62	3.70×	5.77	3.54×	4.29
Selection	4.07×	6.88	6.17×	9.16	3.60×	5.21	3.50×	5.82	**3.75×**	**4.64**
SpecSplice	**5.88×**	**9.39**	**7.18×**	**11.10**	**4.81×**	**6.90**	**5.15×**	**8.63**	**3.75×**	4.63
Δ vs. best	+44.4%		+16.3%		+31.6%		+39.2%		0.0%	
Qwen3.5-27B										
DFlash	3.90×	5.56	3.95×	5.52	3.95×	5.34	4.05×	5.30	3.56×	**5.68**
Suffix.	2.71×	3.57	2.75×	3.40	2.61×	3.12	2.96×	3.48	1.21×	1.52
Graft	3.86×	6.25	4.35×	6.08	3.62×	5.41	4.48×	5.89	3.20×	4.63
Selection	5.18×	7.21	4.95×	6.93	3.94×	5.28	5.49×	7.12	3.55×	5.66
SpecSplice	**5.35×**	**7.68**	**5.53×**	**7.68**	**5.05×**	**6.78**	**6.47×**	**8.40**	**3.61×**	**5.68**
Δ vs. best	+3.4%		+11.6%		+27.9%		+17.8%		+1.6%	
Llama-3.1-8B										
DFlash	2.64×	4.21	3.19×	5.30	4.29×	6.11	2.63×	4.71	2.84×	4.71
Suffix.	4.57×	6.27	3.32×	3.97	5.24×	6.53	3.07×	3.45	1.42×	1.69
Graft	2.90×	4.92	3.74×	5.04	4.62×	6.27	2.88×	4.70	2.80×	4.59
Selection	3.06×	5.66	6.15×	8.56	7.70×	10.56	2.97×	5.37	2.84×	4.70
SpecSplice	**4.70×**	**8.89**	**7.57×**	**11.03**	**9.18×**	**13.19**	**3.86×**	**6.96**	**2.90×**	**4.84**
Δ vs. best	+2.8%		+23.2%		+19.2%		+25.5%		+2.2%	

E RESULTS UNDER THE OFFICIAL HARNESSES

Table 4 repeats the agentic evaluation on Qwen3.5-9B with each benchmark's official agent in place of Claude Code. All other settings are unchanged. The official agents are mini-swe-agent (SWE-agent Team, 2026) for SWE-bench Lite, Spider-Agent (Lei et al., 2025) for Spider2-DBT, `bfcl-eval` (Patil et al., 2025) for BFCL-v4, and ARE (Froger et al., 2026) for GAIA2. Under greedy decoding (T=0), SpecSplice achieves the highest MAT and speedup on every workload. Its speedup gain over the strongest baseline is smaller than under Claude Code on SWE-bench Lite (+21.9% vs. +26.0%), Spider2-DBT (+16.8% vs. +25.3%), and BFCL-v4 (+9.5% vs. +17.2%), but larger on GAIA2 (+14.4% vs. +12.8%).

F SCALING WITH NODE BUDGET AND BATCH SIZE

To examine how the gain of SpecSplice changes as the node budget and the batch size grow, we vary each of them while keeping the other settings fixed. We use Qwen3.5-9B on Spider2-DBT under greedy decoding. In addition to the RTX PRO 6000, we use one NVIDIA B200 with the Triton attention backend, because SGLang's FlashInfer backend does not support the Gated DeltaNet (GDN) kernels of Qwen3.5 on the B200. Figure 7(a) varies the node budget B from 16 to 256 with one active request, and Figure 7(b) varies the batch size with each method at its throughput-optimal B for that batch size. Batch sizes 8 and 16 exceed the memory of the RTX PRO 6000.

Table 4: **Greedy decoding** (T=0) **under the official harnesses**, Qwen3.5-9B. The blue row gives SpecSplice's speedup gain over the strongest baseline.

Method	SWE-bench Lite		Spider2-DBT		BFCL-v4		GAIA2	
	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT
DFlash	4.66×	6.99	4.20×	6.26	3.18×	4.95	5.26×	9.11
SuffixDecoding	3.58×	4.35	2.45×	3.01	2.40×	2.59	3.82×	5.12
Graft	4.49×	7.35	4.35×	6.45	3.02×	4.80	4.97×	8.28
Selection	5.08×	7.72	4.34×	6.40	3.47×	5.41	6.74×	11.67
SpecSplice	**6.19**×	**9.62**	**5.08**×	**7.63**	**3.80**×	**5.92**	**7.71**×	**13.58**
Δ vs. best	+21.9%		+16.8%		+9.5%		+14.4%	

With one active request, MAT increases monotonically with B for both methods, and the MAT lead of SpecSplice over Selection widens from 1.9 to 3.4 tokens per step between B=16 and 256, while the relative lead stays between 26% and 40%. Throughput peaks at an intermediate budget (B=48 for both methods on the RTX PRO 6000; 128 for SpecSplice and 64 for Selection on the B200), and the throughput lead of SpecSplice is largest at the largest budgets. As the batch grows, the throughput-optimal B shrinks, yet SpecSplice remains 15–31% faster than Selection at every batch size on both GPUs, and the absolute gap widens from 75 to 158 tok/s on the RTX PRO 6000 and from 29 to 92 tok/s on the B200.

SpecSplice assigns each additional node to the candidate with the higher contribution score from either source, so it keeps converting budget into accepted tokens after Selection, which spends the whole budget on one source per step, has saturated. Because the gain comes from how the budget is allocated rather than from its size, it persists at the small budgets that large batches require. The method-specific throughput peak is also why each method in Table 1 runs at its own throughput-optimal budget.

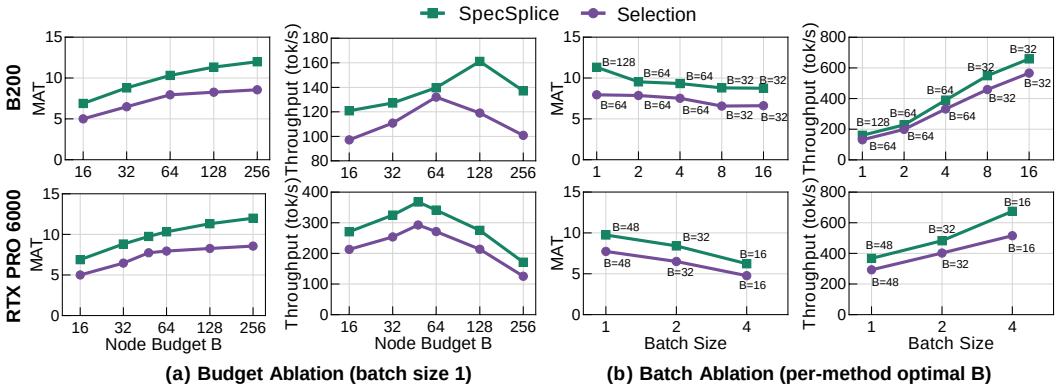


Figure 7: **Scaling with node budget and batch size**, Qwen3.5-9B, Spider2-DBT, greedy decoding, on an NVIDIA B200 and an RTX PRO 6000 (rows). (a) Node budget sweep with one active request. (b) Batch-size sweep with each method at its throughput-optimal B, printed next to each point.

G Comparison with Adaptive Source Scaling

SpecSplice places the two sources on one scale with a single fixed source weight λ (Section 3.1). The scale could instead be set by calibrating each source's acceptance estimates to the observed conditional acceptance, or by tracking acceptance online. This appendix explains why SpecSplice uses the fixed weight. We compare it with offline calibration of the drafter probabilities, the retrieval frequencies, or both, using a linear, logistic, beta, or isotonic map, and with an online exponentially weighted moving average (EWMA) of retrieved-node acceptance with decay β and prior strength κ. All runs use Llama-3.1-8B-Instruct on GAIA2 under greedy decoding with B=64. Every calibrator is fitted and evaluated on the same trace, which favors the alternatives.

Table 5: **Best configuration per family**, Llama-3.1-8B-Instruct, GAIA2, greedy decoding, B=64.

Family	Candidates	Best configuration	MAT	vs. fixed λ
Fixed λ (SpecSplice)	14	λ=0.3	7.313	—
Parametric calibration	4	beta	7.221	-1.26%
Retrieval calibration	4	beta	7.319	$+0.08\%$
Both calibrated	16	linear, linear	7.316	$+0.03\%$
EWMA	16	β=0.97, κ=100	7.309	-0.05%

Table 6: **MAT for each calibration map.** The first two columns calibrate one source with the map in the row; the last four calibrate both, with the parametric map in the row and the retrieval map in the column. The fixed weight reaches 7.313.

	One source		Both sources (retrieval map)			
Map	Parametric	Retrieval	linear	logistic	beta	isotonic
linear	7.217	7.319	7.316	7.293	7.315	7.302
logistic	7.221	7.298	7.306	7.297	7.308	7.298
beta	7.221	7.319	7.313	7.296	7.314	7.301
isotonic	7.218	7.304	7.312	7.295	7.314	7.301

The contribution scores are used only to rank candidates, so the two sources need a common scale only relative to each other, and a single scalar sets this relative scale. Tables 5 and 6 show that fitting richer maps does not improve on this scalar. Calibrating the retrieval frequencies, whose raw scale departs most from conditional acceptance (Figure 3), raises MAT by at most 0.08%, and calibrating both sources by at most 0.03%, even though every calibrator is fitted on the evaluation trace itself. Calibrating only the drafter probabilities lowers MAT by 1.26%, to the level of scoring the two sources identically (λ=1, MAT 7.224), because it leaves the relative scale of the two sources unchanged. An online EWMA estimate needs no offline fitting but introduces a decay and a prior strength, and it is sensitive to the prior: without it (κ=0), MAT falls as low as 5.62, and with the best prior it still does not exceed the fixed weight. The fixed weight thus matches the best calibrated alternative within 0.1% while requiring only one hyperparameter, selected together with the others (Appendix C), and no calibration data or online state.