Frequencies Rank, Probabilities Allocate: Calibrated Draft Trees for Drafter-Free Speculative Decoding

Anonymous authors
Paper under double-blind review

Abstract

Drafter-free speculative decoding verifies trees of continuations retrieved from the prompt, the output so far, or a datastore, and allocates each tree by retrieval scores, such as n-gram frequencies, that rank candidates. We show that ranking is not enough: over all candidate trees, best-first allocation is invariant to a continuous, strictly increasing transformation of edge scores if and only if it is a power map; scores that order every set of siblings correctly can still lose a factor of K in expected accepted draft tokens; and errors of conditional acceptance probabilities along root paths bound the loss. Motivated by source- and depth-dependent miscalibration of retrieval frequencies on real traces, we introduce CalTree, a 29-parameter acceptance model fitted on annotation-free replays of the target's own generations, over retrieval and request-local evidence. On five targets from 1.5B to 32B parameters, it generates 2.6–6.5% more tokens per verification pass than SSSD on Spec-Bench and 4.5–24.5% more on code editing, at every tested budget and target. Source and depth discounts tuned for this metric at each budget on the same replays recover 24–53% of the Spec-Bench gain, and normalizing CalTree's probabilities over the retrieved siblings keeps their order but forfeits 1.3–5.0 points at the tested budgets. Scores fitted on chat transfer to code, and a Qwen2.5-7B scorer to four other targets, without refitting. In fixed-load SGLang benchmarks, CalTree-LR raises throughput over SSSD by 1.0–3.0% on Spec-Bench and 2.6–4.0% on code editing; full-workload outputs are not always token-identical to autoregressive decoding.

1 Introduction

Speculative decoding accelerates autoregressive generation by drafting several future tokens cheaply and verifying them in a single forward pass of the target model (Leviathan et al., 2023; Chen et al., 2023; Stern et al., 2018). Drafts can come from auxiliary heads or a small drafter trained to imitate the target (Cai et al., 2024; Li et al., 2024a; 2025), or, without training any auxiliary model, from text that already exists: the prompt and the output generated so far (Saxena, 2023; Yang et al., 2023; Hu et al., 2025), a datastore of past generations (He et al., 2024; Oliaro et al., 2025), or both (Marzollo et al., 2026). Such *drafter-free* proposers need no draft model, work with any target, and are deployed in production serving engines; on repetitive workloads such as retrieval-augmented generation, summarization, code editing, and agents, they rival drafter-based methods (Oliaro et al., 2025).

Every tree-based proposer faces the same decision each round: which of its candidate tokens to place in a verification tree of K nodes (Miao et al., 2024). Drafter-based systems decide with the drafter's probabilities (Li et al., 2024b; Svirschevski et al., 2024; Wang et al., 2025), which recent work calibrates into acceptance estimates (Wang et al., 2026; Zhou et al., 2026). Drafter-free systems have no drafter probabilities. They decide with retrieval scores: frequency ratios multiplied along paths (Oliaro et al., 2025), source-specific discounts (Marzollo et al., 2026), or source-level acceptance rates (Jin et al., 2026). These scores *rank* candidates but need not estimate acceptance on a common scale, leaving open a question: *what must a score get right for the draft tree to be good?*

Ranking is not allocation. We prove that best-first allocation—which is optimal when scores are the true conditional acceptance probabilities (Chen et al., 2024; Oliaro et al., 2025)—is invariant, over

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.

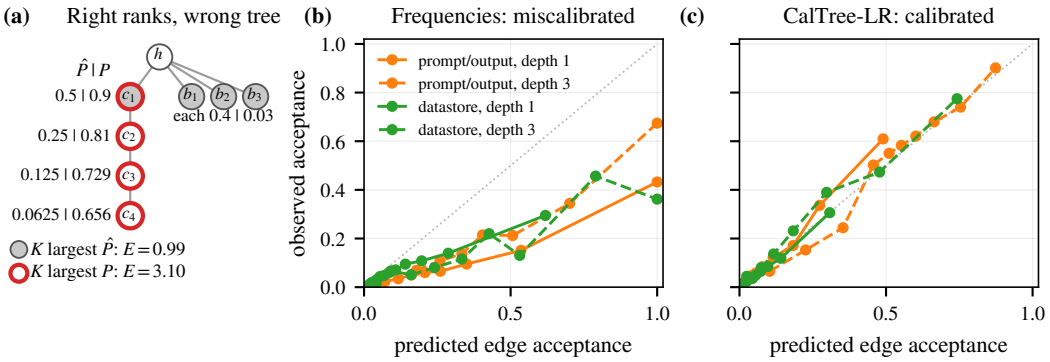


Figure 1: **Ranking siblings correctly is not enough to allocate a draft tree.** (a) A $K = 4$ instance of Proposition 1; each node shows the product of edge scores $\widehat{P}$ and the acceptance probability P. The scores order every set of siblings as the true probabilities do, but their products shrink along the copy run $c_1 \to c_4$, so best-first allocation on $\widehat{P}$ (shaded) expects about a third of the tokens the copy run (outlined) yields. (b, c) Predicted versus observed acceptance of single-source edges at depths 1 and 3 that the verifier compared, each in the score's own trees: frequency ratios (b) and CalTree-LR (c); Qwen2.5-7B, Spec-Bench, $D = 16$.

all candidate trees, to a transformation of edge scores if and only if the transformation is a power map $p \mapsto p^\alpha$ (Theorem 1). Every strictly increasing transformation preserves the order of siblings, but only this one-parameter family preserves every tree. We also construct scores that order every set of siblings correctly yet lose a factor of K in expected accepted draft tokens (Proposition 1, Figure 1a), and we bound the loss of any score by its edge-probability errors along root paths, up to a power map (Proposition 2). The bound points to where drafter-free scores can fail: deep edges, where errors compound, and comparisons across retrieval sources on unrelated scales.

Measured miscalibration. On Spec-Bench traces of five targets (Xia et al., 2024), frequency scores fail in exactly these places (Figure 1b): at depth ≥ 3, a candidate that is the only continuation its source retrieves (frequency ratio 1) is accepted 67–88% of the time when it comes from the request's own text but 36–41% when it comes from a datastore. An oracle that promotes the target's next token among the *same* retrieved candidates would cut verification passes by 12–24%, far more than current scores do.

Learning acceptance from replays. Because the chain rule makes the product of conditional acceptance probabilities the probability that a node is accepted, best-first allocation with the true conditional probabilities is optimal for a fixed candidate tree and budget (Lemma 1). We estimate them. Replaying held-out generations of the target yields, at every true prefix, a label for every retrieved candidate—whether it equals the next token—at no annotation cost. We fit a small calibrated model over evidence that any drafter-free proposer can compute: per-source retrieval statistics, cross-source agreement, request-local repetition, and the length of the longest earlier context that a candidate would continue, computed online with a suffix automaton (Blumer et al., 1985).

Contributions.

- **Theory.** We characterize the score transformations that preserve best-first allocation on every candidate tree, separate sibling ranking from allocation quality, and bound the loss from inaccurate edge probabilities, with consequences for depth discounts and multi-source fusion (Section 2).
- **Diagnosis.** On real traces, retrieval frequencies relate to acceptance through source and depth in ways that no power map can absorb (Section 4.2).
- **Method.** CalTree, a retrieval-based acceptance model fitted on annotation-free replays; it transfers across targets and tokenizers and, refitted on enlarged pools, gains from added retrieval sources.
- **Evidence.** Five targets (1.5B–32B) on chat, code generation, and code editing, and an implementation in SGLang (Zheng et al., 2024): in exact replay, CalTree generates **2.6–6.5%** more tokens

per verification pass than SSSD on Spec-Bench and **4.5–24.5%** more on code editing; end to end, it raises offline SGLang throughput over SSSD by 1.0–3.0% and 2.6–4.0%.

2 What a draft tree needs from its scores

2.1 Setup

Let h be the verified prefix and $y_1, y_2, \ldots$ the continuation the target would produce greedily. A proposer returns a *candidate tree* $\mathcal{C}$ rooted at h whose nodes are tokens; siblings are distinct tokens. A draft tree $\mathcal{T} \subseteq \mathcal{C}$ is *ancestor-closed* and has at most K non-root nodes, over which all sums and orderings of $\mathcal{T}$ below range. Greedy verification accepts the longest root path of $\mathcal{T}$ that agrees with y and adds one token of the target's own. A node u at depth d is therefore accepted iff its path spells $y_{1:d}$. For an edge e entering u, let

$$p_e = \Pr\big[u \text{ accepted} \mid \text{parent of } u \text{ accepted}\big], \qquad P(u) = \Pr[u \text{ accepted}] = \prod_{e \in \text{path}(u)} p_e, \quad (1)$$

where the product is the chain rule, since a node can only be accepted if its parent is. As at most one node per depth is accepted, the number of accepted draft tokens is $A(\mathcal{T}) = \sum_{u \in \mathcal{T}} \mathbf{1}[u \text{ accepted}]$ and

$$E(\mathcal{T}) = \mathbb{E}[A(\mathcal{T})] = \sum_{u \in \mathcal{T}} P(u). \quad (2)$$

Probabilities describe uncertainty about the greedy continuation y given the evidence the proposer observes, not sampling by the target. A *score* is any assignment $\widehat{p}_e \in (0, 1]$; it induces $\widehat{P}(u) = \prod_{e \in \text{path}(u)} \widehat{p}_e$ and $\widehat{E}(\mathcal{T}) = \sum_{u \in \mathcal{T}} \widehat{P}(u)$. Frequency ratios $q_e = f(\text{child})/f(\text{parent})$ in a retrieval trie are one such score; SuffixDecoding multiplies them along paths (Oliaro et al., 2025).

Lemma 1 (Best-first allocation)**.** *Since $\widehat{P}$ is non-increasing along every root path, the K non-root nodes with the largest $\widehat{P}$ (all, if fewer; ties broken by a fixed order that prefers shallower nodes) form an ancestor-closed tree that maximizes $\widehat{E}$, and best-first expansion by $\widehat{P}$ returns it.*

This is standard (Chen et al., 2024; Oliaro et al., 2025); it makes the allocation a function of the *order* of $\widehat{P}$ over all nodes—across depths and branches, not only among siblings.

2.2 Allocation is invariant only to power maps

Consider a transformation g applied to every edge score. Sibling order is preserved by every strictly increasing g. Allocation is much more demanding.

Theorem 1 (Power-map characterization)**.** *Let $g : (0, 1] \to (0, 1]$ be continuous and strictly increasing. Best-first allocation with scores $g(\widehat{p}_e)$ selects the same tree as with $\widehat{p}_e$ for every candidate tree, every score assignment, and every budget K if and only if $g(p) = p^\alpha$ for some $\alpha > 0$. Candidate trees with three non-root nodes and $K = 2$ already separate every other g.*

The proof (Appendix A) reduces invariance on a two-edge path versus a single edge to Cauchy's multiplicative equation $g(xy) = g(x)g(y)$. Two consequences frame the rest of the paper.

Corollary 1 (Depth discounts)**.** *A calibration of frequencies $\widehat{p}_e = c(q_e)$ changes the allocation of raw frequency products only through its departure from a power map. For the logistic family $c(q) = \sigma(b + a \log q)$ with $a > 0$, $\log \widehat{P}(u) = d_u\, b + \sum_{e \in \text{path}(u)} \big[a \log q_e - \log(1 + e^b q_e^a)\big]$: a power map, plus a depth-proportional offset, plus a per-edge saturation term.*

Corollary 2 (Fusing sources)**.** *Suppose candidates come from sources s whose raw scores relate to the true conditional probabilities by continuous, strictly increasing maps onto $(0, 1)$, $p_e = f_s(r_e)$. For continuous, strictly increasing ψ_s with values in $(0, 1]$, ranking nodes by products of $\psi_s(r_e)$ yields the optimal allocation for every candidate tree iff $\psi_s = f_s^\alpha$ with a common $\alpha > 0$. In particular, a fixed multiplicative discount per source (Marzollo et al., 2026) is optimal on every candidate tree only if the discounted scores equal the conditional probabilities up to a common power.*

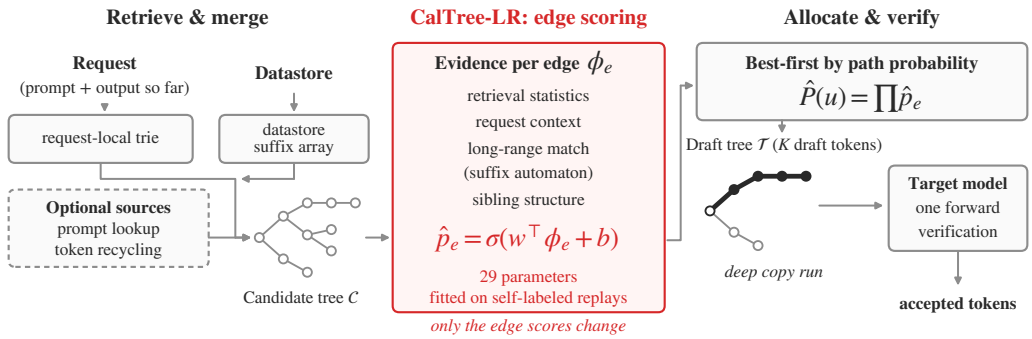


Figure 2: **Method overview (Section 3): CalTree changes edge scoring, not retrieval, the allocator, or verification.** Algorithm 1 (Appendix B) gives one decoding step.

2.3 Ranking accuracy does not bound allocation loss

Proposition 1 (Ranking-perfect, allocation-poor)**.** *For every $K \geq 2$ and $\varepsilon > 0$ there is a candidate tree and a score that orders every set of siblings exactly as the true probabilities do, such that best-first allocation with the score achieves $E(\widehat{\mathcal{T}}) \leq (1+\varepsilon)\, E(\mathcal{T}^\star)/K$, where $\mathcal{T}^\star$ is optimal.*

The instance is a copy run (Figure 1a): a chain of edges that the target will almost surely accept, scored only moderately above unrelated root candidates. Best-first spends the budget on the root candidates because the chain's scores shrink with depth while its probabilities do not. It is a worst case, not a predicted K-fold loss; Section 4.2 finds both ingredients in real traces.

2.4 Probability errors control allocation loss

Proposition 2 (Regret bound)**.** *Let $\widehat{\mathcal{T}}$ be the best-first tree under $\widehat{P}$ and $\mathcal{T}^\star$ an optimal tree under P with the same budget. Then, for every $\alpha > 0$,*

$$E(\mathcal{T}^\star) - E(\widehat{\mathcal{T}}) \;\leq \sum_{u \in \mathcal{T}^\star \triangle \widehat{\mathcal{T}}} \left|\widehat{P}(u)^\alpha - P(u)\right|, \qquad \left|\widehat{P}(u)^\alpha - P(u)\right| \;\leq \sum_{e \in \mathrm{path}(u)} \left|\widehat{p}_e^{\,\alpha} - p_e\right|. \quad (3)$$

Only nodes in the symmetric difference of the two trees contribute, and conditional-probability errors accumulate along their paths. The bound concerns single-round allocation at a fixed budget, is sufficient rather than necessary for good allocation, and gives no statistical guarantee for a learned scorer; within source-wise monotone edge maps, optimality on every candidate tree requires probabilities accurate up to a common power *conditionally on the parent being accepted*, at every depth and across sources (Corollary 2)—exactly where frequencies fail. Binned calibration error does not certify the pointwise errors in the bound, so we also evaluate allocation directly (Section 4.3).

3 Learning calibrated acceptance

CalTree replaces a proposer's edge scores by an estimate $\widehat{p}_e$ of the conditional acceptance probability in equation 1, keeping retrieval, candidate merging, the allocator, and verification (Figure 2). A token proposed at a parent by several sources or matched orders is one edge with pooled evidence.

Evidence. The feature vector ϕ_e uses only quantities that a drafter-free proposer already has or maintains incrementally, in four groups (all 28 in Appendix B; cost in Section 4.6):

- *Retrieval*: which sources propose the token; for the request-local prompt/output index and the datastore, the frequency ratio q_e, parent support Z_e (log scale), and matched order; when enabled, prompt-lookup match length and chain position and token-recycling rank and log-probability; the number of sources; the depth.
- *Request context*: how often the token occurred in the verified prefix and drafted path (a cache-model signal; Kuhn & De Mori, 1990), whether it never did, and the distance since it last did.

- *Long-range match*: with an online suffix automaton over the verified prefix (Blumer et al., 1985), walked along the drafted path, the length ℓ_e of the longest suffix of prefix-plus-path that occurred earlier, the length $L_e \leq \ell_e$ of the longest such suffix that was followed by the candidate, and the number of distinct tokens that followed it. Unlike trie matches, which proposers cap at a few tokens, L_e is uncapped and grows along a copy run, so a deep edge inside the run carries the evidence that its shallow counterpart would carry once the parent is verified.
- *Sibling structure*: the entropy of the child counts of the trie and of the datastore at the parent.

Self-labeled supervision. Labels come from replaying the target's greedy generations through the proposer. The generations are split in two folds; generations of one fold are replayed against a datastore built from the other, so no replayed text is retrieved from itself. *Stage 1*: at every true prefix, every first-level candidate is labeled by whether it equals the next token (negatives subsampled at rate $1/4$, reweighted). *Stage 2*: trees are built with the stage-1 model, and every edge in them whose parent is accepted during verification—the event that equation 1 conditions on—is labeled at every depth. Maximum likelihood thus targets conditional acceptance on the edges these trees visit, not on unvisited edges; calibration in the final trees is checked separately (Section 4.2).

Model. CalTree-LR is $\widehat{p}_e = \sigma(w^\top \phi_e + b)$ with 29 parameters, fitted by ℓ_2-regularized Newton steps on the weighted log-likelihood (settings in Appendix D); we check its calibration per depth and source on held-out tasks (Section 4.2). Computing its evidence roughly doubles the CPU time of SSSD's tree builder (Section 4.6). CalTree-GB replaces the linear logit by 300 gradient-boosted trees over the same features (Friedman, 2001; Pedregosa et al., 2011); it measures how much the evidence supports beyond a linear model. Neither model sees the target's weights or hidden states, nor its logits unless the optional token-recycling source supplies them.

4 Experiments

4.1 Setup

Targets and data. Our five targets are Qwen2.5-1.5B/7B-Instruct (Qwen Team, 2024), Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and Qwen3-8B/32B in non-thinking mode (Yang et al., 2025), decoding greedily up to 1024 new tokens. Evaluation uses all 480 Spec-Bench questions (Xia et al., 2024)—80 each of translation, summarization, question answering, mathematical reasoning, retrieval-augmented generation, and multi-turn MT-Bench (Zheng et al., 2023)—for 560 turns per target. Each target's datastore holds its greedy completions of 2048 ShareGPT prompts filtered against the evaluation turns (1.1–1.5M tokens including prompts; Appendix C). The same completions supply the training labels (Section 3); evaluation text is never used for fitting. We also replay HumanEval's 164 function-completion problems (Chen et al., 2021) and CanItEdit's 105 editing problems under both instruction styles (Cassano et al., 2023), with the same models and datastores.

Proposer. We reimplement the retrieval rules of SSSD (Marzollo et al., 2026) as it runs in SGLang (Zheng et al., 2024): a request-local trie of 8-token windows over prompt and output and a suffix array over the datastore, both queried with short suffixes of the verified prefix (Appendix C). The tree budget $D = K + 1$ (nodes per request, including the root) also sets the datastore continuation length and the per-node fan-out by SSSD's rule; $D \in \{4, 8, 16, 32\}$ covers the sizes SSSD chooses on an A100 for batch sizes 32, 16, 8, and 1. On the checked prefixes, SSSD's C++ extension and our replay retrieve identical candidates, and our C++ implementation of CalTree-LR reproduces every feature and edge probability to within 5×10^{-15} (Appendix C).

Scores. We compare scoring rules on SSSD's candidates: at a given prefix and budget, all scores see the same candidate tree, although the prefixes they reach can differ. *Native*: SSSD's source-discounted priorities. *Frequency*: products along paths of each edge's largest frequency ratio over sources, a SuffixDecoding-style control, not the complete system (Oliaro et al., 2025); unlike native priorities, it maximizes over sources before multiplying, without discounts. *Iso*: a frequency-only calibration—isotonic regression (Zadrozny & Elkan, 2002) of each source's frequency ratio, separately per source and depth $(1, 2, 3, \geq 4)$ and maximized over sources at each edge, fitted on the same labels as CalTree. *Tuned discounts*: native priorities multiplied along the path by one factor per source and depth group, tuned for τ itself at each budget on the replays that supply CalTree's labels (Appendix D). *Oracle*: native priorities with the target's next token promoted whenever retrieved,



Figure 3: **CalTree generates more tokens per verification pass at every budget and target.** Gain in τ over native SSSD with retrieval fixed at each prefix and budget (%; Spec-Bench, 560 turns per target; D counts the root), with 95% bootstrap intervals over questions; values in Table 9.

so every round accepts as much as its candidates allow (a round-wise reference, not a pass-minimal policy).

Metrics. Replaying the stored greedy traces measures acceptance exactly. We report τ, total generated tokens over total verification passes (target forwards, each also emitting one token of the target's own), and its relative gain over native SSSD; as all methods emit identical tokens, the τ ratio is the inverse ratio of their passes. Paired 95% bootstrap intervals resample questions (2000 replicates).

4.2 Frequencies are miscalibrated where it matters

We examine the edges that each score placed in its trees and that the verifier compared, i.e., whose parent was accepted—the population that equation 1 conditions on. These populations differ between scores, so calibration statistics diagnose each score on its own trees rather than compare predictors on common edges. Three observations hold for all five targets (Figure 1b,c; Appendix I).

Equal frequencies, unequal acceptance. A candidate that is the only continuation its source retrieves ($q = 1$) is accepted 36–61% of the time at depth 1 and 67–88% at depth ≥ 3 when it comes from the request's own text, but only 24–31% and 36–41% when it comes from the datastore. Such edges are 45–65% of compared request-local edges, so frequency alone cannot distinguish them. No power map of frequencies reconciles the sources (Corollary 2), and depth calls for Corollary 1's offset.

Acceptance rises along the path. On CalTree-LR's compared edges, conditional acceptance grows from 10–14% at depth 1 to 57–79% at depth ≥ 4, consistent with copy runs among accepted paths. Frequency ratios overestimate acceptance at every depth (expected calibration error 0.17–0.18).

Calibration is learnable from replays. CalTree-LR, fitted on ShareGPT and evaluated on Spec-Bench, has an expected calibration error of 0.005–0.007 overall and at most 0.033 in each depth group ($1, 2, 3, \geq 4$), for every target. Iso reaches 0.024–0.029 but still overestimates acceptance at depths 2 and 3 for every target; a maximum of separately fitted maps need not be calibrated.

4.3 Calibrated trees accept more drafts

Spec-Bench. CalTree-LR raises τ over native SSSD by 2.6–6.5% at every budget and for every target (Figure 3), with 95% intervals of half-width 0.2–0.8 points. Frequency products are mostly worse than native priorities (-3.5 to $+0.7$%), and isotonic calibration of frequencies per source and depth recovers 0.3–2.6%, which is 10–46% of CalTree's gain in the same setting. Budget-tuned discounts gain 0.7–2.5%, 24–53% of CalTree-LR's gain; the tested search does not exhaust this family, and whether CalTree-LR beats it in serving throughput is untested. CalTree-GB, on the same evidence, adds another 0.4–1.3 percentage points at every budget and target. Relative to the round-wise oracle, CalTree-LR closes 9–24% of the gap in τ from native SSSD.

Table 1: **Chat-fitted scores transfer to code without refitting.** At $D = 16$, τ is native SSSD's tokens per verification pass and the other columns are gains in τ (%) of Iso and CalTree-LR/-GB; models and datastores are from ShareGPT; bold: best per target and workload.

	Spec-Bench				HumanEval				CanItEdit			
Target	τ	Iso	LR	GB	τ	Iso	LR	GB	τ	Iso	LR	GB
Qwen2.5-7B	1.92	+1.1	+3.3	**+4.4**	2.12	+0.9	+3.7	**+4.6**	3.25	+3.7	+8.8	**+9.4**
Llama-3.1-8B	2.02	+1.2	+3.4	**+4.7**	2.26	+1.2	+3.7	**+5.2**	3.18	+3.6	+7.7	**+8.7**
Qwen3-8B	1.87	+0.7	+3.5	**+4.2**	2.15	+1.1	+4.2	**+4.8**	3.26	+3.6	+8.8	**+8.9**
Qwen3-32B	1.76	+0.7	+3.2	**+4.0**	2.19	+0.9	+3.7	**+4.0**	3.09	+2.7	**+8.4**	+8.3
Qwen2.5-1.5B	2.51	+2.3	+5.0	**+5.9**	2.17	+1.0	+3.9	**+4.6**	4.66	+9.4	+15.4	**+15.6**

Table 2: **Power maps leave the trees unchanged; normalizing over siblings keeps their order but removes most of the gain.** Qwen2.5-7B on Spec-Bench: gain in τ over native SSSD (%) of CalTree-LR under sibling-order-preserving transforms of its probabilities (left; normalized rows divide by the sum over the retrieved siblings) and with feature groups removed or kept alone, refitted (right).

Transform of $\hat{p}$	D=4	D=8	D=16	Features	D=4	D=8	D=16
calibrated $\hat{p}$	+2.9	+3.8	+3.3	all features	+2.9	+3.8	+3.3
power $\hat{p}^{0.5}$ or $\hat{p}^2$	+2.9	+3.8	+3.3	retrieval only	+0.9	+1.7	+1.1
normalized over siblings	+1.4	−0.7	+0.3	retrieval only, normalized	0.0	−1.8	−1.2
temperature T=0.5	+2.6	+2.5	+2.6	− request context	+2.6	+3.6	+3.1
temperature T=1.4	+2.7	+3.9	+3.3	− long-range match	+2.3	+3.4	+3.0
depth exponent 0.8^{d-1}	+2.8	+3.1	+2.7	− sibling structure	+2.8	+3.7	+3.2
depth exponent 1.1^{d-1}	+2.9	+3.9	+3.4	− retrieval	−1.4	−0.2	−0.5

Across tasks and domains. At $D = 16$, every Spec-Bench category improves for every target, from +0.8% on translation to +7.1% on summarization (Appendix G), and the ShareGPT-fitted models and datastores carry over unchanged to code (Table 1, Appendix E). On HumanEval, CalTree-LR gains 1.9–5.7% across budgets while frequency products lose up to 2.9%. CanItEdit's outputs largely overlap the prompt (56–85% of their tokens lie in 8-grams of it, against 5–24% on Spec-Bench): native SSSD already generates 3.09–4.66 tokens per pass at $D = 16$, and across budgets CalTree-LR raises τ by a further 4.5–24.5%, most at $D = 8$ for every target. Frequency products gain there too ($+0.3$ to $+14.6$%), consistent with less discounting of copy runs, but CalTree-LR beats them and Iso (-2.7 to $+11.0$%) at every budget and target.

4.4 Allocation, not ranking

Raising $\widehat{p}$ to the power 0.5 or 2 reproduces every verification pass exactly, as Theorem 1 guarantees, whereas the tested non-power transformations change the trees (Table 2; all settings in Table 12): rescaling the logits by a temperature, $\sigma(z/T)$, which keeps the order of all edges, moves the gain by -1.3 to $+0.1$ points, and depth-dependent exponents, $\widehat{p}^{\gamma^{d-1}}$, which keep the order of every set of siblings, by -0.7 to $+0.1$ points; the calibrated scores are at or near the best of either family.

Learned rankings are not enough either. Normalizing CalTree's probabilities over each set of retrieved siblings keeps every sibling order but forces the retrieved siblings to carry unit mass, whereas their true acceptance probabilities sum to at most one, the rest belonging to unretrieved tokens (a bound our independent estimates do not enforce). It removes 1.5–4.6 points of the gain on Qwen2.5-7B, leaving -0.7 to $+1.4$% (Table 2), and 1.3–5.0 points across the five targets at $D \in \{4, 8, 16\}$. In the crossed ablation (Table 2), keeping the learned scale rather than normalizing adds 0.9–4.6 points with either feature set, and evidence beyond retrieval statistics adds 1.0–2.2 points at either scale: sibling order alone is insufficient, though this does not show that calibration alone explains the gain.

Where the extra tokens come from. Relative to native SSSD at $D = 16$, CalTree-LR raises the share of rounds that accept at least one draft token by 1.5–2.5 points and makes full-depth runs of 7 accepted tokens 2.8–3.5 times as frequent, while the shares for the middle thresholds $k = 3$–5 move by at most 0.4 points (Appendix H). Full-depth runs copy request-local text, and their increase is consistent with the failure mode of Proposition 1; depth-aware isotonic calibration already recovers

Table 3: **CalTree-LR is faster than SSSD at every tested load; whether tree speculation beats AR depends on the batch size.** Median of three Spec-Bench runs in SGLang on one A100-80GB: autoregressive decoding (AR), speedup over AR (×AR), change over SSSD (Δ%, largest in bold), and engine-counted τ. Batch: maximum concurrency; D: tree budget; timings include prefill.

	AR	SSSD		Iso	CalTree-LR			CalTree-GB	
Batch (D)	tok/s	×AR	τ	Δ%	×AR	Δ%	τ	Δ%	τ
Qwen2.5-7B									
1 (32)	74	1.47	1.98	+0.1	1.50	**+1.5**	2.02	−0.1	2.05
8 (16)	505	0.99	1.93	−0.1	1.02	**+2.4**	1.99	+1.3	2.01
16 (8)	903	0.95	1.80	−0.5	0.97	**+2.0**	1.86	+1.1	1.88
32 (4)	1512	0.84	1.62	−1.3	0.85	**+1.0**	1.66	−0.4	1.68
Llama-3.1-8B									
1 (32)	71	1.45	2.05	+1.2	1.50	**+3.0**	2.12	−0.8	2.10
8 (16)	468	1.06	2.02	+0.3	1.08	**+1.9**	2.07	+1.0	2.09
16 (8)	824	0.98	1.88	−0.4	1.00	**+1.9**	1.95	+1.0	1.96
32 (4)	1278	0.89	1.67	−0.5	0.90	+1.1	1.72	**+1.2**	1.74

most of it (58–87% of CalTree's gain). At the first position, sources compete on unrelated scales (Corollary 2), and Iso obtains at most 47% of the gain.

What the evidence adds. On Qwen2.5-7B ($D \leq 16$), logistic regression on retrieval statistics alone gains 0.9–1.7%, comparable to tuned discounts (0.7–1.6%), against 2.9–3.8% with all features. Every other group helps, the long-range match most, but they overlap (removing any one of these three costs at most 0.62 points), and without retrieval statistics the rest falls below SSSD: CalTree refines rather than replaces retrieval scores.

Transfer and supervision. Applied unchanged to the other four targets at $D \in \{8, 16\}$, including Llama-3.1-8B's different tokenizer, the Qwen2.5-7B scorer changes the gain by -0.2 to $+0.2$ points relative to their own fits; each target keeps its own datastore. On Qwen2.5-7B at these budgets, fitting on the replays of 16 ShareGPT generations (0.11M labels) instead of 512 changes the gain by at most 0.1 points with the datastores fixed (nested subsets), and first-level labels alone come within 0.1 points of the two-stage fit (Appendices D and F); this measures scorer supervision, not datastore cost.

4.5 Composing retrieval sources

We test score fusion on enlarged candidate pools, not complete retrieval systems. Adding prompt lookup (Saxena, 2023) and token recycling (Luo et al., 2025) to SSSD's candidates raises the oracle's gain on Qwen2.5-7B from 19.3–27.5% to as much as 40.8–48.2% (Table 14, Appendix K). Refitted on each pool, CalTree-LR gains more than on SSSD's own candidates in every combination we test, reaching 4.7%; on Llama-3.1-8B it rises from 3.4–4.1% to 4.4–5.7% with both sources. Native and frequency scores, extended by fixed rules that serve as controls for incompatible scales, yield up to 15.1% *fewer* tokens per pass than native SSSD on its own candidates; this does not rank CalTree against tuned fusion rules.

4.6 Serving-engine throughput

We implement CalTree inside SSSD's C++ tree builder, which SGLang calls at every decoding step (Figure 2); the fan-out and budget rules are untouched, and the suffix automaton and request statistics are maintained per request. Building a tree takes 119–213 μs with CalTree-LR and 362–670 μs with CalTree-GB, against 64–112 μs natively (one core, $D = 4$–32); the extension builds the trees of a batch in parallel. We serve Spec-Bench with Qwen2.5-7B and Llama-3.1-8B, each on one A100-80GB under the fixed configuration of Appendix J, at batch sizes 1, 8, 16, and 32 with SSSD's draft sizes 32, 16, 8, and 4: latency at batch size 1 on every fourth turn (140 turns), and throughput at larger batch sizes with all 560 turns at once. Each configuration runs three times, alternating methods.

The gains in τ carry over to the engine, whose τ is within 0.9% (Qwen2.5-7B) and 2.3% (Llama-3.1-8B) of the replay's on the same turns, and CalTree-LR raises SSSD's throughput by 1.0–2.4% and

1.1–3.0%, respectively, at every batch size (Table 3); every run of CalTree-LR is faster than every run of SSSD, by at least 0.7% and 0.5%. On CanItEdit (Qwen2.5-7B, Appendix J), CalTree-LR adds 2.6–4.0%. These are gains over SSSD, not over autoregressive decoding (AR) at every load: SSSD's own speedup falls from 1.47× at batch size 1 to 0.84× at 32 (1.45× to 0.89×), as larger batches leave less idle compute. Gains are smaller than in τ because scoring is paid in full: SGLang overlaps scheduling with the forward pass only for chain drafts, so every tree-speculation step adds its CPU time. At batch size 1, CalTree-LR lengthens a step of 18.2 ms by 0.8% and CalTree-GB by 3.7% (Qwen2.5-7B; 0.6% and 3.4% for Llama-3.1-8B), which offsets GB's higher acceptance in replay. Full-workload throughput is not an equal-output comparison: 83–90% and 89–94% of speculative outputs match AR token for token, and near-tied logits at the checked divergences do not establish equal quality (Appendix J). On the 107 and 116 of 140 latency turns where all methods match in all runs, CalTree-LR is 1.8% and 1.4% faster than SSSD, on this output-selected subset only.

5 Related work

Tree construction from acceptance estimates. SpecInfer and SpecTr verify token trees and sets of drafts (Miao et al., 2024; Sun et al., 2023); Sequoia builds hardware-aware trees from positional acceptance rates (Chen et al., 2024); EAGLE-2, SpecExec, and OPT-Tree expand trees by drafter probabilities (Li et al., 2024b; Svirschevski et al., 2024; Wang et al., 2025). C2T, SpecDec++, TAPS, TreeSpark, and GRAFT learn or calibrate acceptance estimates on top of a drafter's outputs (Huo et al., 2025; Huang et al., 2024; Wang et al., 2026; Zhou et al., 2026; Ye et al., 2026). These estimators start from a drafter's outputs, which drafter-free proposers lack. Our contribution is not acceptance prediction itself but a retrieval-evidence formulation and controlled tests of the score properties that allocation requires; the fixed-pool experiments isolate scoring, not complete retrieval systems.

Drafter-free proposers. Drafts come from prompt lookup and reference copying (Saxena, 2023; Yang et al., 2023), adaptive n-gram tables (Ou et al., 2024), multi-branch tries over prompts and past responses (Zhao et al., 2024), token recycling (Luo et al., 2025), next-next-token logits as retrieval keys (Liu et al., 2025), suffix automata (Hu et al., 2025), suffix trees (Oliaro et al., 2025), n-grams collected by Jacobi iteration (Fu et al., 2024), datastores (He et al., 2024), and their combination (Marzollo et al., 2026; Jin et al., 2026; Liu et al., 2026). Goose shapes the tree by source, a deep chain of context matches with wide branches of other candidates (Jin et al., 2026); CalTree instead scores individual retrieved candidates and is complementary to better retrieval.

Calibration. Platt scaling and isotonic regression map scores to probabilities (Platt, 1999; Zadrozny & Elkan, 2002; Niculescu-Mizil & Caruana, 2005), and modern networks are often miscalibrated (Guo et al., 2017). Our use is decision-theoretic, like calibrating predictions to the decisions they serve (Zhao et al., 2021): allocation compares path products and is invariant exactly to power maps of edge scores (Theorem 1).

6 Conclusion

Sibling rankings alone do not determine a good draft tree: over all candidate trees, best-first allocation is invariant only to power maps of the edge scores, and errors in conditional acceptance probabilities bound its loss. Retrieval frequencies are miscalibrated in source- and depth-dependent ways; a 29-parameter acceptance model learned from the target's own replays improves allocation on SSSD's candidates for five targets and three workloads, with consistent but modest serving gains over SSSD and the largest replay gains on copy-heavy code editing. The analysis applies to any best-first tree built from products of edge scores, including a drafter's.

Limitations. CalTree, like the oracle, only chooses among retrieved candidates. Features estimate the conditional probabilities only approximately, and the theory gives no statistical guarantee for the learned estimator. We study fixed-budget greedy allocation, not sampling or latency-optimal budgets, and do not compare complete alternative retrieval systems. Engine tests cover one GPU type, one proposer family, and two targets under fixed offline loads, without tuned discounts; full-workload outputs are not always identical, and on Spec-Bench, tree speculation is at or below AR at batch sizes 16–32.

AI USE STATEMENT

We used an LLM coding assistant to implement and modify the code; it also contributed to the theoretical analysis and proofs, the experimental design, the analysis of results, and the writing. The authors reviewed all AI-assisted work and take responsibility for its content.

ETHICS STATEMENT

This work accelerates inference without changing the target's decoding rule: in exact arithmetic, greedy verification emits the target's own tokens, and finite-precision engine outputs can differ as quantified in Section 4.6. It trains no new language model, involves no human subjects, and uses only public models and datasets (Section 4.1). Cheaper inference lowers the cost of deploying language models, with the benefits and risks of the models themselves.

REPRODUCIBILITY STATEMENT

All experiments use public models (Qwen2.5-1.5B/7B-Instruct, Llama-3.1-8B-Instruct, Qwen3-8B/32B) and public data (Spec-Bench, HumanEval, CanItEdit, and the cleaned ShareGPT split), with greedy decoding and fixed seeds for pool sampling, negative subsampling, and gradient boosting. Section 4.1 and Appendices B–I specify the retrieval parameters, features, label collection, and evaluation protocol. The supplementary material contains the replay engine (suffix-array datastore, prompt trie, suffix automaton, all scores), the fitting and analysis scripts that produce every number, table, and figure from stored replay outputs and check every per-target claim, the modified C++ tree builder with a patch against SSSD, and the SGLang benchmark harness; we will release it publicly.

REFERENCES

János Aczél. *Lectures on Functional Equations and Their Applications*. Academic Press, 1966.

Anselm Blumer, J. Blumer, David Haussler, Andrzej Ehrenfeucht, M. T. Chen, and Joel Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40: 31–55, 1985.

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 *International Conference on Machine Learning*, 2024.

Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Jacob Ginesin, Edward Berman, George Chakhnashvili, Anton Lozhkov, Carolyn Jane Anderson, and Arjun Guha. Can it edit? Evaluating the ability of large language models to follow code editing instructions. *arXiv preprint arXiv:2312.12450*, 2023.

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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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*, 2024.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *International Conference on Machine Learning*, 2024.

Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pp. 1321–1330, 2017.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. REST: Retrieval-based speculative decoding. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2024.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. SAM decoding: Speculative decoding via suffix automaton. In *Annual Meeting of the Association for Computational Linguistics*, 2025.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. SpecDec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv preprint arXiv:2405.19715*, 2024.

Feiye Huo, Jianchao Tan, Kefeng Zhang, Xunliang Cai, and Shengli Sun. C2T: A classifier-based tree construction method in speculative decoding. *arXiv preprint arXiv:2502.13652*, 2025.

Tao Jin, Phuong Minh Nguyen, and Naoya Inoue. Goose: Anisotropic speculation trees for training-free speculative decoding. *arXiv preprint arXiv:2604.02047*, 2026.

Roland Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Conference on Empirical Methods in Natural Language Processing*, 2024b.

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

Runheng Liu, Jincheng Xie, Wen Hu, Xingchen Xiao, and Heyan Huang. AdaPLD: Adaptive retrieval and reuse for efficient model-free speculative decoding. *arXiv preprint arXiv:2606.05742*, 2026.

Tianyu Liu, Qitan Lv, Hao Li, Xing Gao, Xiao Sun, and Xiaoyan Sun. LogitSpec: Accelerating retrieval-based speculative decoding via next next token speculation. *arXiv preprint arXiv:2507.01449*, 2025.

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 *Annual Meeting of the Association for Computational Linguistics*, 2025.

Michele Marzollo, Jiawei Zhuang, Niklas Roemer, Niklas Zwingenberger, Lorenz K. Müller, and Lukas Cavigelli. SSSD: Simply-scalable speculative decoding. In *Annual Meeting of the Association for Computational Linguistics*, pp. 33123–33139, 2026.

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 *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024.

Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *International Conference on Machine Learning*, pp. 625–632, 2005.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. SuffixDecoding: Extreme speculative decoding for emerging AI applications. In *Advances in Neural Information Processing Systems*, 2025.

Jie Ou, Yueming Chen, and Wenhong Tian. Lossless acceleration of large language model via adaptive n-gram parallel decoding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Industry Track*, pp. 10–22, 2024.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pp. 61–74. MIT Press, 1999.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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

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

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

Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. SpecExec: Massively parallel speculative decoding for interactive LLM inference on consumer devices. In *Advances in Neural Information Processing Systems*, 2024.

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

Zhuoyu Wang, Junnan Huang, and Xinyu Chen. TAPS: Target-aware prefix tree selection for diffusion-drafted speculative decoding. *arXiv preprint arXiv:2606.00487*, 2026.

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*, 2024.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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.

Xuming Ye, Zeming Ma, Runjie Yu, Yuan Liu, Tianle Li, Shuhan Bai, Jian Zhou, and Fei Wu. GRAFT: Adaptive DLM-based draft tree construction with target-distilled edge scoring. *arXiv preprint arXiv:2608.20375*, 2026.

Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 694–699, 2002.

Shengjia Zhao, Michael P. Kim, Roshni Sahoo, Tengyu Ma, and Stefano Ermon. Calibrating predictions to decisions: A novel approach to multi-class calibration. In *Advances in Neural Information Processing Systems*, 2021.

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*, pp. 6344–6355, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023.

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*, 2024.

Huapeng Zhou, Huayu Wang, and Xinyu Wang. TreeSpark: Calibrated, load-adaptive draft trees for semi-autoregressive speculative decoding. *arXiv preprint arXiv:2609.22098*, 2026.

A Proofs

Conventions. If a parent has probability zero, p_e is undefined and can be set to any value in $(0, 1]$ without changing $P(u) = 0$. Ties are broken by a fixed order that prefers shallower nodes.

Lemma 1. Every factor lies in $(0, 1]$, so $\widehat{P}$ does not increase from a parent to a child. With ties broken toward shallower nodes, a parent precedes each of its children in the order, hence any prefix of the order is ancestor-closed. The first K non-root nodes (all, if fewer) maximize $\sum_{u\in\mathcal{T}} \widehat{P}(u)$ over all sets of at most K non-root nodes, in particular over feasible trees. Best-first expansion inserts, at every step, the largest-$\widehat{P}$ node whose parent is already in the tree; by the same monotonicity this is the largest remaining node overall. □

Theorem 1. *If.* For $g(p) = p^\alpha$, the transformed priority is $\prod_e \widehat{p}_e^{\,\alpha} = \widehat{P}(u)^\alpha$, a strictly increasing function of $\widehat{P}(u)$, so the order of all nodes, and by Lemma 1 the allocation, is unchanged. *Only if.* Consider the candidate tree with root children a and c, where a has one child b, and scores $\widehat{p}_a = x$, $\widehat{p}_b = y$, $\widehat{p}_c = z$ with $x, y \in (0, 1]$ and $z < x$; let $K = 2$. Best-first first selects a; it then selects b if $xy > z$ and c if $xy < z$. Invariance requires $g(x)g(y) > g(z)$ whenever $z < xy$ and $g(x)g(y) \le g(z)$ whenever $xy < z < x$ (a tie goes to the shallower c). Letting $z \uparrow xy$ and $z \downarrow xy$ (possible when $y < 1$) and using continuity gives $g(xy) = g(x)g(y)$ for all $x \in (0, 1]$ and $y \in (0, 1)$; continuity extends this to $y = 1$, which also gives $g(1) = 1$. Strict monotonicity makes g positive on $(0, 1]$. Define $G(t) = -\log g(e^{-t})$ for $t \ge 0$; then G is continuous and additive, $G(s+t) = G(s)+G(t)$, hence $G(t) = \alpha t$ (Aczél, 1966), and $\alpha > 0$ because g is strictly increasing. Therefore $g(p) = p^\alpha$. □

Corollary 1. Expand $\log \sigma(b + a \log q) = b + a \log q - \log(1 + e^b q^a)$ and sum along the path. □

Corollary 2. Ranking by products of $\psi_s(r_e) = \psi_s(f_s^{-1}(p_e))$ is ranking by products of $g_s(p_e)$ with $g_s = \psi_s \circ f_s^{-1}$, which is continuous and strictly increasing on $(0, 1)$. *If.* For $g_s(p) = p^\alpha$, every node is ranked by $P(u)^\alpha$, hence as by P, and best-first allocation by P is optimal (Lemma 1). *Only if.* Take the candidate tree of the proof of Theorem 1 with all three edges from source s and true probabilities $p_a = x \le \frac{1}{2}$, $p_b = y < 1$, and $p_c = z < x$; the root's children are feasible because $x + z < 1$. With $K = 2$, the optimal tree contains b if $xy > z$ and c if $xy < z$, so the argument of that proof gives $g_s(xy) = g_s(x)g_s(y)$ for all $x \in (0, \frac{1}{2}]$ and $y \in (0, 1)$. With $G(t) = -\log g_s(e^{-t})$, this reads $G(a + t) = G(a) + G(t)$ for $a \ge \log 2$ and $t > 0$. Expanding $G(a + t + u)$ once as $G(a) + G(t + u)$ and once as $G(a + t) + G(u) = G(a) + G(t) + G(u)$ shows that G is additive on $(0, \infty)$; being continuous, $G(t) = \alpha_s t$ with $\alpha_s > 0$ (Aczél, 1966), i.e., $g_s(p) = p^{\alpha_s}$ and $\psi_s = f_s^{\alpha_s}$. Finally, let $K = 1$ and let the root have two children, from sources s and t, with true probabilities $x, z \le \frac{1}{2}$. Whatever the tie rule, optimality requires $x^{\alpha_s} \ge z^{\alpha_t}$ whenever $x > z$ and $x^{\alpha_s} \le z^{\alpha_t}$ whenever $x < z$; letting $z \to x$ gives $x^{\alpha_s} = x^{\alpha_t}$ for all $x \in (0, \frac{1}{2})$, hence $\alpha_s = \alpha_t$. □

Proposition 1. Let the root have children $c_1, b_1, \ldots, b_{K-1}$ and let $c_1 \to c_2 \to \cdots \to c_K$ be a chain. Set the true probabilities $p_{c_d} = 1 - \eta$ and $p_{b_j} = \delta$ with $(K - 1)\delta \le \eta$, so that the root's children, of which at most one is accepted, have total probability at most one, and the score $\widehat{p}_{c_d} = 0.5$, $\widehat{p}_{b_j} = 0.4$. Every parent other than the root has a single child, and at the root the score orders c_1 above every b_j, as the truth does, so every set of siblings is ordered correctly. Best-first selects c_1 (0.5) and then the b_j ($0.4 > 0.25 = \widehat{P}(c_2)$), giving $E(\widehat{\mathcal{T}}) = (1 - \eta) + (K - 1)\delta$. The chain $c_{1:K}$

gives $E(\mathcal{T}^\star) \geq \sum_{d=1}^{K}(1-\eta)^d \geq K(1-\eta)^K$. Choosing η small enough, and $\delta \leq \eta/(K-1)$, makes the ratio at most $(1+\varepsilon)/K$. □

Proposition 2. By Theorem 1, $\widehat{\mathcal{T}}$ is also the best-first tree under the scores $\widehat{p}_e^{\,\alpha}$, whose path products are $\widehat{P}(u)^\alpha$, so it suffices to prove the case $\alpha = 1$. Write $E(\mathcal{T}^\star) - E(\widehat{\mathcal{T}}) = [E(\mathcal{T}^\star) - \widehat{E}(\mathcal{T}^\star)] + [\widehat{E}(\mathcal{T}^\star) - \widehat{E}(\widehat{\mathcal{T}})] + [\widehat{E}(\widehat{\mathcal{T}}) - E(\widehat{\mathcal{T}})]$. The middle term is non-positive by optimality of $\widehat{\mathcal{T}}$ for $\widehat{E}$ (Lemma 1). In the outer terms, nodes in $\mathcal{T}^\star \cap \widehat{\mathcal{T}}$ cancel, leaving at most $\sum_{u \in \mathcal{T}^\star \triangle \widehat{\mathcal{T}}} |\widehat{P}(u) - P(u)|$. For the second inequality, telescope $\prod_{i=1}^{d} a_i - \prod_{i=1}^{d} b_i = \sum_{i=1}^{d} \big(\prod_{j<i} a_j\big)(a_i - b_i)\big(\prod_{j>i} b_j\big)$ and bound every product by one. □

B Evidence features

Every feature of an edge e from parent v to token x at depth d is computed from the proposer's retrieval results and the request's verified prefix h; the drafted path π is the root path of v. Absent sources contribute zeros and a zero indicator.

Group	Features
Retrieval (prompt/output trie)	indicator; $\log q$ and $\log(1+Z)$ of the matched order with the largest q (ties to the longer order); the longest matched order; the number of matched orders proposing x
Retrieval (datastore)	indicator; $\log q$; $\log(1+Z)$; matched suffix length after back-off
Retrieval (prompt lookup)	indicator; matched n-gram length; position in the copied chain
Retrieval (token recycling)	indicator; rank of x in the recycled top-k list; recycled log-probability
Retrieval (shared)	depth d; number of sources proposing x
Request context	$\log(1+c)$ with c the occurrences of x in $h\pi$; indicator that x never occurs; $\log(1+r)$ with r the distance from the last occurrence
Long-range match	indicator $L_e < 0$ (x never follows any earlier suffix); $\log(1+L_e)$; $\log(1+\ell_e)$; indicator $L_e = \ell_e \geq 1$; $\log(1+\ell_e - L_e)$; log of the number of distinct successors of the matched state
Sibling structure	entropy of the child counts at v in the trie and in the datastore

The suffix automaton is built online over h (amortized $O(1)$ per token) and walked along π; ℓ_e is the length of the longest suffix of $h\pi$ that occurs in h ending before its last position, and L_e the length of the longest suffix of $h\pi$ that occurs in h followed by x (-1 if x does not occur in h). When $L_e = -1$, the features $\log(1+L_e)$, $\log(1+\ell_e - L_e)$, and the indicator $L_e = \ell_e \geq 1$ are set to zero; when x never occurs in $h\pi$, so is $\log(1+r)$. Algorithm 1 shows where the features enter one decoding step.

Algorithm 1 CalTree inside a drafter-free proposer, one decoding step. Lines 3 and 8 replace the proposer's edge scores; everything else is the proposer's own.

Require: verified prefix h; budget K; retrieval sources; fitted edge model $\widehat{p}$ (CalTree-LR: $\widehat{p}_e = \sigma(w^\top \phi_e + b)$)

1: $\mathcal{C} \leftarrow$ candidates retrieved for h from every source, merged into one tree ▷ unchanged
2: $\mathcal{T} \leftarrow \{\text{root}\}$; $\mathcal{F} \leftarrow$ children of the root in $\mathcal{C}$
3: $\widehat{P}(v) \leftarrow \widehat{p}(\phi_e)$ for every $v \in \mathcal{F}$, with e the edge into v ▷ evidence, Section 3
4: **while** $|\mathcal{T} \setminus \{\text{root}\}| < K$ and $\mathcal{F} \neq \emptyset$ **do**
5: $\quad u \leftarrow \arg\max_{v \in \mathcal{F}} \widehat{P}(v)$ ▷ best-first (Lemma 1)
6: $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{u\}$; $\mathcal{F} \leftarrow \mathcal{F} \setminus \{u\}$
7: $\quad$ **for** every child c of u in $\mathcal{C}$, with e the edge into c **do**
8: $\quad\quad \widehat{P}(c) \leftarrow \widehat{P}(u)\,\widehat{p}(\phi_e)$; $\mathcal{F} \leftarrow \mathcal{F} \cup \{c\}$ ▷ ϕ_e extends u's automaton walk
9: $\quad$ **end for**
10: **end while**
11: **return** $\mathcal{T}$ ▷ verified in one forward pass of the target

C Replay protocol

The 2048 ShareGPT prompts of each datastore are sampled from 52,934 deduplicated English first turns after removing every prompt that shares more than half of its word 5-grams with an evaluation

turn (the largest remaining overlap is 0.25). The request-local trie is queried with every suffix of length ≤ 4 of the verified prefix, so a k-token match continues for up to $8 - k$ tokens, and the datastore with the length-4 suffix. SSSD's rule sets the datastore continuation length and the per-node fan-out from the budget: $(3, 3)$, $(5, 5)$, $(6, 5)$, and $(6, 5)$ for $D = 4, 8, 16, 32$. Native SSSD scores a node by the product of frequency ratios along its path, multiplying request-local edges by $0.6 + 0.1k$ for a k-token match, and takes the maximum over the sources and matched orders that propose it; Frequency and Iso instead take, at each edge, the largest ratio or source-wise map over sources, and Iso falls back to its depth-1 map where a depth group has fewer than 200 labels. Each Spec-Bench turn is replayed against its stored greedy trace. A round builds a draft tree from the verified prefix with the method under test, verifies it greedily against the trace (the longest root path that agrees with the trace is accepted and the next trace token is emitted), and appends the emitted tokens to the prefix, the request-local trie, the suffix automaton, and the recycling table. Because every method emits the trace, the per-turn outputs of all methods are identical and are checked to be; methods differ only in the number of rounds. Prompt lookup (Saxena, 2023) proposes the continuation after the most recent earlier match of the longest suffix of length ≤ 8, chained up to 8 tokens; token recycling (Luo et al., 2025) proposes the top-8 successors of the parent token in the target's own next-token distributions earlier in the request, and its *warm* variant initializes this table from the target's distributions over its datastore generations; after each round, the table is updated only with the distributions at the newly accepted positions, all of which that round's verification pass has computed. The datastore is a suffix array over the 2048 ShareGPT completions of the same target, sampled as in SSSD's C++ reader (evenly spaced over the matching range, at most 100 continuations of at most the continuation length, back-off while fewer than 50 continuations match); we verified the sampling against a brute-force implementation of the same rule on 150 random datastores and queries, and the request-local index against a transcription of SSSD's C++ insertion rules (prompt windows start at every position but the last three, which join the output buffer; output windows are inserted once 8 tokens are available) on random sequences. Finally, we run SSSD's C++ extension (with our C++ implementation of CalTree) and our replay side by side on the first 20 Spec-Bench turns, feeding both the same verified prefixes. With CalTree-LR, over 13159 rounds at $D \in \{8, 16, 32\}$, 217400/217400 groups of retrieved candidates coincide and every feature and edge probability agrees to within 5×10^{-15}; the same holds for Iso, while CalTree-GB's probabilities differ on 0.08% of edges, whose features lie within rounding error of a split threshold. With LR and Iso, trees differ only where candidates have exactly equal scores and are expanded in a different order. Because native SSSD breaks such ties by the layout of its hash tables, which changes even between runs of the extension, we also replay all 560 turns of Qwen2.5-7B through the extension itself: τ differs from our replay by at most 0.07% for CalTree-LR and 0.4% for native SSSD, where our insertion-order tie-breaking favors the baseline slightly.

D Training labels

For each target, 256 held-out ShareGPT generations per fold are replayed against a datastore built from the other fold's 1024 completions, with $D = 16$ for stage 2. Table 4 lists the resulting labels (stage-1 negatives are subsampled at rate $1/4$ and carry weight 4) and the calibration of a model fitted on one fold and evaluated on the other: weighted log-loss for both folds and, at each depth d, the larger of the two folds' absolute mean errors $|b_d|$ of the predicted probability.

Fitting. CalTree-LR takes weighted Newton steps with a ridge penalty of 10^{-3} on the weighted sum of log-likelihood terms, applied to every weight but the bias, on unstandardized features, for at most 60 iterations, stopping once no coordinate moves by more than 10^{-8}. Each fold's stage-2 trees are built with a stage-1 model fitted on that fold's own stage-1 labels; the final model uses the labels of both folds. CalTree-GB uses scikit-learn's histogram gradient boosting (Pedregosa et al., 2011) (version 1.8) with 300 iterations, learning rate 0.08, 31 leaves per tree, and ℓ_2 regularization 1 on the same labels and weights; the library's default early-stopping check on a 10% validation split never triggered. Iso fits weighted isotonic regressions of each source's native statistic per source and depth group on the same labels.

On Qwen2.5-7B, stage 2 matters for Iso but hardly for CalTree-LR. Fitted on the stage-1 labels alone, which contain first-level candidates only, CalTree-LR gains 2.6–3.8% on Qwen2.5-7B at $D \in \{4, 8, 16, 32\}$, within 0.1 points of the two-stage fit (2.6–3.8%) at every budget, and stays calibrated at depths 2 to ≥ 4 (expected calibration error at most 0.036 on Spec-Bench, $D = 16$): its

evidence extends to depths absent from its labels. Iso, which sees only frequency ratios, must then apply its depth-1 maps at every depth and loses up to 0.5 points.

Table 4: Self-supervised labels per target (SSSD candidates) and cross-fold calibration of CalTree-LR (log-loss for both folds; $|b_d|$: the larger of the two).

Target	Labels	Positive	Stage 2	Log-loss (folds)	$\lvert b_1\rvert$	$\lvert b_2\rvert$	$\lvert b_3\rvert$	$\lvert b_{\geq 4}\rvert$
Qwen2.5-7B	3.68M	310k	813k	0.0639/0.0656	0.000	0.008	0.016	0.014
Llama-3.1-8B	3.46M	301k	743k	0.0612/0.0627	0.000	0.012	0.021	0.026
Qwen3-8B	4.70M	381k	1042k	0.0626/0.0632	0.000	0.004	0.017	0.012
Qwen3-32B	4.64M	351k	1057k	0.0620/0.0628	0.000	0.006	0.013	0.022
Qwen2.5-1.5B	2.78M	302k	587k	0.0622/0.0610	0.000	0.012	0.017	0.013

Tuned discounts. This control multiplies each native path priority (Appendix C) of a candidate at depth d from source s (request-local or datastore) by $\prod_{j \leq d} c_{s,\min(j,4)}$ and keeps the maximum over sources and matched orders, so that $c \equiv 1$ reproduces native SSSD exactly. Scaling both depth-1 factors by a common constant changes no order, so the request-local one is fixed to 1. For each target, the other seven factors are chosen by coordinate ascent over $\{0.1, 0.2, 0.3, 0.5, 0.7, 1, 1.4, 2, 3\}$ (two sweeps from $c \equiv 1$; a factor changes only if that saves passes) to maximize τ at $D = 16$ on the replays that supply the labels above. Starting from these factors, a pattern search then refines them separately at each budget on a continuous scale: each round tries multiplying and dividing every factor by 1.4, later by 1.15, and keeps the best move if it saves passes (at most two moves per step size, searched on each fold's first 128 generations). Each budget uses its own refined factors (Table 5), an advantage that CalTree's single model does not have. On the tuning replays (each fold's first 256 generations) they raise τ by 0.8–2.5%, within 0.3 points of their Spec-Bench gains; the refinement moves the Spec-Bench gains of the $D = 16$ coordinate-ascent factors used at every budget from 0.4–2.5% to 0.7–2.5%. The search still does not establish the best factors of this family. The factors set search priorities rather than probabilities; with factors above one, priorities can grow along a path, so Lemma 1 does not apply to this control, which we evaluate in replay only.

Table 5: **Discounts tuned for τ at each budget.** Factors per source and depth group (request-local depth-1 factor fixed to 1) refined at each budget D on the label replays, and gains in τ (%) over native SSSD at the same D on those replays (Tuning) and on Spec-Bench, with CalTree-LR's Spec-Bench gain for reference.

		Request-local			Datastore				Gain in τ (%)		
Target	D	d=2	d=3	$d\geq 4$	d=1	d=2	d=3	$d\geq 4$	Tuning	Spec-Bench	CalTree-LR
Qwen2.5-7B	4	1.00	0.50	1.00	0.80	0.45	0.70	0.70	+0.9	+0.7	+2.9
	8	1.00	0.50	1.00	0.98	0.24	0.70	0.61	+1.4	+1.3	+3.8
	16	1.00	0.57	1.00	0.80	0.20	0.70	0.70	+1.6	+1.6	+3.3
	32	1.00	0.50	1.00	0.70	0.20	0.61	0.80	+1.1	+1.2	+2.6
Llama-3.1-8B	4	1.40	0.48	1.00	0.50	0.32	0.70	0.70	+0.8	+1.1	+2.9
	8	1.40	0.30	1.00	0.50	0.45	0.70	0.70	+1.5	+1.4	+4.1
	16	1.40	0.30	1.00	0.80	0.14	0.61	0.70	+1.8	+1.7	+3.4
	32	1.40	0.34	1.15	0.36	0.20	0.70	0.70	+1.2	+1.2	+2.6
Qwen3-8B	4	0.98	0.70	1.00	0.80	0.48	0.70	0.70	+0.8	+0.8	+3.0
	8	0.70	0.80	0.87	0.70	0.42	0.98	0.70	+1.4	+1.2	+3.8
	16	0.70	0.50	1.00	0.70	0.30	0.61	0.70	+1.4	+1.3	+3.5
	32	0.98	0.80	1.00	0.70	0.21	0.70	0.70	+1.1	+1.1	+2.9
Qwen3-32B	4	1.15	0.70	1.00	1.00	0.42	1.13	0.70	+0.8	+0.7	+2.8
	8	1.00	0.80	0.71	1.00	0.48	0.70	0.70	+1.4	+1.4	+3.8
	16	1.00	0.70	1.00	1.15	0.42	0.70	0.50	+1.2	+1.1	+3.2
	32	1.00	0.50	1.22	1.00	0.30	0.70	0.61	+0.9	+1.0	+2.7
Qwen2.5-1.5B	4	2.00	0.48	1.00	0.50	0.32	0.50	0.70	+1.6	+1.9	+4.3
	8	2.00	0.42	1.00	0.80	0.20	0.43	0.70	+2.0	+2.1	+6.5
	16	2.00	0.30	1.00	0.57	0.17	0.50	0.70	+2.5	+2.5	+5.0
	32	2.00	0.30	1.40	0.50	0.20	0.50	0.80	+1.5	+1.8	+3.4

E Code generation and editing

HumanEval (Chen et al., 2021) contains 164 Python function-completion problems; the signature and docstring of each problem form the user message. CanItEdit (Cassano et al., 2023) contains 105 Python editing problems, each with a descriptive and a lazy instruction; the user message asks for the complete edited code, followed by the original code in a code block and the instruction, for 210 requests, and bootstrap intervals resample problems. Every target generates up to 1024 tokens greedily, as for Spec-Bench. Nothing is refitted: all scores and datastores are those fitted on and built from the ShareGPT generations (Section 4.1), whose prompts share at most 18% and 12% of their word 5-grams with a HumanEval and a CanItEdit prompt. Of the output tokens, 14–26% on HumanEval, 56–85% on CanItEdit, and 5–24% on Spec-Bench lie in 8-grams that also occur in the request's prompt, and native SSSD reaches $\tau = 2.12$–2.26 on HumanEval and 3.09–4.66 on CanItEdit at $D = 16$. Across $D \in \{4, 8, 16, 32\}$, CalTree-LR gains 1.9–5.7% and 4.5–24.5%, and frequency products -2.9 to -0.4% and $+0.3$ to $+14.6$%; Tables 6 and 7 detail $D \in \{8, 16\}$.

Table 6: HumanEval: gain in τ over native SSSD (%) with the scores and datastores fitted on and built from ShareGPT.

	Qwen2.5-7B		Llama-3.1-8B		Qwen3-8B		Qwen3-32B		Qwen2.5-1.5B	
Score	D=8	D=16	D=8	D=16	D=8	D=16	D=8	D=16	D=8	D=16
Frequency product	-2.9	-2.9	-2.6	-2.1	-2.3	-1.7	-1.4	-2.0	-2.3	-2.4
Iso (source×depth)	$+0.3$	$+0.9$	$+0.4$	$+1.2$	$+0.5$	$+1.1$	$+0.6$	$+0.9$	$+1.7$	$+1.0$
CalTree-LR	$+3.6$	$+3.7$	$+3.7$	$+3.7$	$+5.1$	$+4.2$	$+5.7$	$+3.7$	$+4.5$	$+3.9$
CalTree-GB	$+4.8$	$+4.6$	$+4.9$	$+5.2$	$+5.4$	$+4.8$	$+5.8$	$+4.0$	$+5.5$	$+4.6$
Oracle	$+33.2$	$+22.9$	$+33.6$	$+23.1$	$+33.5$	$+23.6$	$+33.2$	$+22.3$	$+34.9$	$+23.6$

Table 7: CanItEdit: gain in τ over native SSSD (%) with the scores and datastores fitted on and built from ShareGPT.

	Qwen2.5-7B		Llama-3.1-8B		Qwen3-8B		Qwen3-32B		Qwen2.5-1.5B	
Score	D=8	D=16	D=8	D=16	D=8	D=16	D=8	D=16	D=8	D=16
Frequency product	$+3.7$	$+1.9$	$+2.4$	$+1.4$	$+4.3$	$+2.6$	$+3.1$	$+1.7$	$+14.6$	$+9.5$
Iso (source×depth)	-0.3	$+3.7$	$+1.3$	$+3.6$	-0.1	$+3.6$	-2.4	$+2.7$	$+11.0$	$+9.4$
CalTree-LR	$+12.1$	$+8.8$	$+10.8$	$+7.7$	$+12.7$	$+8.8$	$+10.9$	$+8.4$	$+24.5$	$+15.4$
CalTree-GB	$+12.8$	$+9.4$	$+11.8$	$+8.7$	$+12.7$	$+8.9$	$+11.1$	$+8.3$	$+24.9$	$+15.6$
Oracle	$+34.2$	$+22.4$	$+34.4$	$+22.4$	$+32.6$	$+21.1$	$+31.5$	$+21.1$	$+36.9$	$+22.2$

F Amount of supervision

Table 8 refits CalTree-LR and Iso for Qwen2.5-7B on the replays of the first n ShareGPT generations of each fold, with both labeling stages and all datastores unchanged, and replays Spec-Bench with the refitted scores. The subsets are nested, so the table does not measure variability across independently resampled training sets.

G Spec-Bench gains by budget and category

Table 9 lists the gains of Figure 3 with native SSSD's τ, and Table 10 breaks those of CalTree-LR down by Spec-Bench category; MT-Bench covers both turns of its 80 questions, the second conditioned on the target's own first answer.

H Accepted lengths

Figure 4 decomposes the gains of Section 4.3 by the number of draft tokens accepted per round. At $D = 16$, drafts are at most 7 tokens deep: request-local windows hold 8 tokens, so a match of one token continues for at most 7, and SSSD's rule continues datastore matches for at most 6; full-depth runs thus copy the request's own text. The feature ablations of Table 2 separate the two ends

Table 8: Gain in τ over native SSSD (Spec-Bench, Qwen2.5-7B, %) of scores fitted on the labels of n generations per fold; $n = 256$ is the main setting.

Supervision			Iso		CalTree-LR	
Per fold	Total	Labels	D=8	D=16	D=8	D=16
2	4	0.03M	+0.6	+1.0	+3.5	+3.2
4	8	0.06M	+0.4	+0.9	+3.6	+3.2
8	16	0.11M	+0.7	+1.0	+3.8	+3.4
16	32	0.20M	+0.7	+1.0	+3.8	+3.3
32	64	0.45M	+0.7	+1.1	+3.8	+3.3
64	128	0.94M	+0.8	+1.1	+3.8	+3.3
128	256	1.83M	+0.9	+1.1	+3.8	+3.3
256	512	3.68M	+0.8	+1.1	+3.8	+3.3

Table 9: Spec-Bench: native SSSD's τ and gains in τ (%) over it with retrieval fixed at each prefix and budget, per target and budget (Figure 3); bold: best non-oracle score per row. Tuned: discounts tuned for τ at each budget (Appendix D).

Target	D	Native τ	Freq.	Tuned	Iso	LR	GB	Oracle
Qwen2.5-7B	4	1.61	−0.5	+0.7	+0.3	+2.9	**+3.8**	+32.1
	8	1.80	−2.9	+1.3	+0.8	+3.8	**+5.0**	+27.5
	16	1.92	−2.1	+1.6	+1.1	+3.3	**+4.4**	+19.3
	32	1.99	−0.7	+1.2	+0.8	+2.6	**+3.4**	+15.0
Llama-3.1-8B	4	1.67	−0.2	+1.1	+0.5	+2.9	**+3.9**	+29.6
	8	1.89	−2.3	+1.4	+0.9	+4.1	**+5.3**	+26.3
	16	2.02	−1.6	+1.7	+1.2	+3.4	**+4.7**	+18.2
	32	2.10	−0.4	+1.2	+0.8	+2.6	**+3.4**	+13.9
Qwen3-8B	4	1.57	−0.7	+0.8	+0.5	+3.0	**+3.5**	+32.4
	8	1.75	−3.3	+1.2	+0.5	+3.8	**+4.6**	+27.1
	16	1.87	−2.4	+1.3	+0.7	+3.5	**+4.2**	+19.5
	32	1.93	−0.9	+1.1	+0.7	+2.9	**+3.3**	+15.5
Qwen3-32B	4	1.51	−0.9	+0.7	+0.4	+2.8	**+3.5**	+32.2
	8	1.66	−3.5	+1.4	+0.8	+3.8	**+4.7**	+26.3
	16	1.76	−2.7	+1.1	+0.7	+3.2	**+4.0**	+19.0
	32	1.82	−1.1	+1.0	+0.6	+2.7	**+3.2**	+15.5
Qwen2.5-1.5B	4	1.95	+0.7	+1.9	+1.7	+4.3	**+5.0**	+30.3
	8	2.31	−0.2	+2.1	+2.6	+6.5	**+7.2**	+30.9
	16	2.51	−0.3	+2.5	+2.3	+5.0	**+5.9**	+20.6
	32	2.64	+0.1	+1.8	+1.5	+3.4	**+4.1**	+14.7

on Qwen2.5-7B. With retrieval statistics alone, CalTree-LR keeps most of its full-depth gain (1.4 instead of 1.6 points), as does the model without the long-range match (1.4 points), but its gain at the first position falls from 1.9 to 0.3 points.

I Calibration by target and depth

Table 11 reports, for every target, the expected calibration error (20 equal-width bins) of the edges each score placed in its trees and the verifier compared at $D = 16$, and the mean predicted versus observed acceptance of these edges by depth. SSSD's native priorities are path scores rather than probabilities and are omitted. Each score is evaluated on its own compared edges, so the populations differ between scores; these are calibration diagnostics, not estimates of the pointwise errors in Proposition 2. Table 12 lists every tested transform of the Qwen2.5-7B scores behind Table 2.

Table 10: Gain in τ of CalTree-LR over native SSSD by Spec-Bench category ($D = 16$, %).

Target	MT	Trans.	Summ.	QA	Math	RAG	All
Qwen2.5-7B	+3.5	+2.3	+3.1	+2.7	+3.0	+4.0	+3.3
Llama-3.1-8B	+3.8	+0.8	+2.9	+2.6	+3.4	+5.1	+3.4
Qwen3-8B	+3.9	+1.9	+2.6	+2.9	+2.9	+4.4	+3.5
Qwen3-32B	+3.6	+0.8	+2.5	+2.8	+2.1	+4.1	+3.2
Qwen2.5-1.5B	+4.8	+2.7	+7.1	+4.3	+3.9	+6.9	+5.0

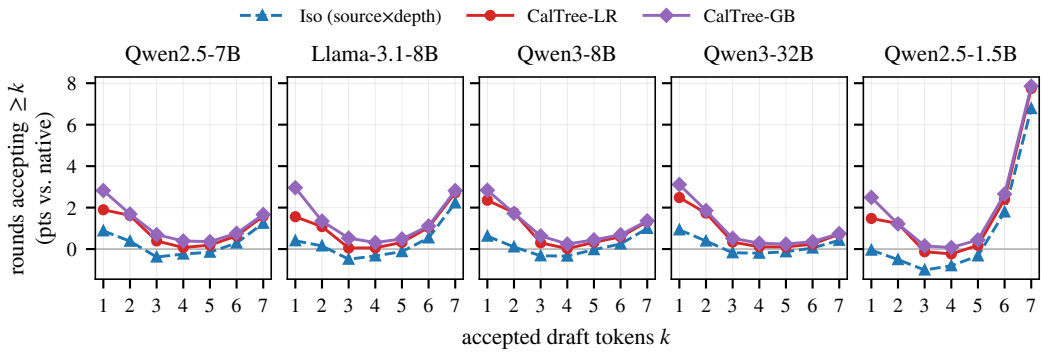


Figure 4: Change in the share of verification rounds that accept at least k draft tokens, relative to native SSSD with retrieval fixed at each prefix and budget (Spec-Bench, $D = 16$, percentage points).

J Serving code editing and output agreement

Configuration. All runs use SGLang's offline engine (commit 194f2d7, with SSSD's C++ speculator and our patches from the supplementary material) on one A100-80GB, with FP16 weights, an FP32 output head, FlashInfer attention, CUDA graphs, the radix cache disabled, and at most as many running requests as the batch size; SSSD's adaptive draft size is disabled, so each batch size uses the draft size in Table 3. Before timing, each server generates two 8-token requests and, for batch sizes above 1, one batch of 32-token requests from ShareGPT prompts disjoint from the evaluation. At batch size 1 we sum the wall-clock times of the individual requests; at larger batch sizes we time one call that submits all requests, from submission to the last completion. Both include prefill but exclude datastore construction and fitting; throughput is generated tokens per second, and the engine's τ is generated tokens per verification step.

Output agreement. On Spec-Bench, 83–90% (Qwen2.5-7B) and 89–94% (Llama-3.1-8B) of the outputs of the speculative methods match autoregressive decoding token for token, and where a divergence can be checked against the stored logits, the top two tokens at the first divergence are within 0.04 nats of each other, consistent with numerical sensitivity; this check does not establish that divergent outputs are equally good. The latency comparison restricted to requests where all methods match in all runs covers 107 and 116 of the 140 latency turns; equal-output speedups are established only on these subsets, not on the full workloads or at larger batch sizes.

Code editing. We serve CanItEdit with Qwen2.5-7B in the configuration of Section 4.6, at batch size 1 (D=32; latency on every third request, 70 requests) and 8 (D=16; throughput with all 210 requests submitted at once), three runs each in alternating method order (Table 13). SSSD itself runs 2.38× as fast as autoregressive decoding at batch size 1 and 1.58× at 8, and CalTree-LR raises its throughput by 2.6–4.0%; every run of CalTree-LR is faster than every run of SSSD, by at least 1.6%. For every configuration with a matching replay (all but CalTree-GB at $D = 32$), the engine's τ is within 3.5% of the replay's on the same requests, and 91–94% of outputs match autoregressive decoding; on the 61 of 70 latency requests where all methods match in all runs, CalTree-LR is 2.9% faster than SSSD. At batch size 1, scoring lengthens a step of 19.2 ms by 1.0% for CalTree-LR and 3.9% for CalTree-GB, which again offsets GB's higher τ.

Table 11: Calibration on Spec-Bench of compared edges ($D = 16$): expected calibration error, and mean predicted / observed acceptance in each depth group d.

Target	Score	ECE	d=1	d=2	d=3	$d\geq 4$
Qwen2.5-7B	Frequency product	0.175	0.28/0.14	0.55/0.30	0.71/0.42	0.89/0.64
	Iso (source×depth)	0.024	0.13/0.11	0.26/0.22	0.47/0.40	0.69/0.65
	CalTree-LR	0.005	0.11/0.11	0.24/0.23	0.42/0.42	0.66/0.64
	CalTree-GB	0.003	0.12/0.11	0.25/0.24	0.42/0.42	0.66/0.66
Llama-3.1-8B	Frequency product	0.178	0.29/0.14	0.58/0.32	0.78/0.51	0.92/0.72
	Iso (source×depth)	0.026	0.13/0.11	0.28/0.25	0.54/0.50	0.74/0.75
	CalTree-LR	0.006	0.11/0.11	0.24/0.25	0.47/0.49	0.72/0.72
	CalTree-GB	0.004	0.12/0.12	0.27/0.27	0.50/0.50	0.72/0.73
Qwen3-8B	Frequency product	0.171	0.27/0.13	0.55/0.30	0.69/0.40	0.88/0.63
	Iso (source×depth)	0.025	0.12/0.10	0.26/0.22	0.44/0.36	0.71/0.67
	CalTree-LR	0.006	0.11/0.11	0.23/0.23	0.39/0.40	0.65/0.64
	CalTree-GB	0.003	0.11/0.11	0.24/0.24	0.40/0.40	0.65/0.65
Qwen3-32B	Frequency product	0.171	0.26/0.12	0.55/0.29	0.68/0.38	0.86/0.57
	Iso (source×depth)	0.024	0.11/0.10	0.25/0.21	0.41/0.33	0.67/0.63
	CalTree-LR	0.005	0.10/0.10	0.22/0.22	0.37/0.38	0.59/0.57
	CalTree-GB	0.003	0.10/0.10	0.23/0.23	0.38/0.37	0.60/0.59
Qwen2.5-1.5B	Frequency product	0.168	0.33/0.18	0.62/0.38	0.81/0.57	0.92/0.80
	Iso (source×depth)	0.029	0.16/0.14	0.34/0.30	0.60/0.55	0.82/0.81
	CalTree-LR	0.007	0.14/0.14	0.30/0.31	0.54/0.55	0.79/0.79
	CalTree-GB	0.003	0.14/0.14	0.33/0.33	0.56/0.56	0.80/0.80

Table 12: All tested transforms of CalTree-LR's probabilities (Qwen2.5-7B, Spec-Bench): gain in τ over native SSSD (%). Table 2 shows a subset.

Transform of $\hat{p}$	D=4	D=8	D=16
calibrated $\hat{p}$	+2.9	+3.8	+3.3
power $\hat{p}^{0.5}$ or $\hat{p}^2$	+2.9	+3.8	+3.3
normalized over siblings	+1.4	−0.7	+0.3
temperature T=0.5	+2.6	+2.5	+2.6
temperature T=0.7	+2.7	+3.3	+3.0
temperature T=1.4	+2.7	+3.9	+3.3
temperature T=2	+2.1	+2.8	+2.7
depth exponent 0.8^{d-1}	+2.8	+3.1	+2.7
depth exponent 0.9^{d-1}	+2.8	+3.6	+3.1
depth exponent 1.1^{d-1}	+2.9	+3.9	+3.4
depth exponent 1.25^{d-1}	+2.8	+3.8	+3.2

K Composing retrieval sources

Corollary 2 characterizes the source-wise scores that are optimal on every candidate tree. Table 14 adds prompt lookup (Saxena, 2023) and token recycling (Luo et al., 2025) to SSSD's candidates (Appendix C specifies both). The enlarged pools raise the oracle's gain from 19.3–27.5% to as much as 40.8–48.2%, yet with native or frequency scores the same budget yields up to 15.1% *fewer* tokens per pass than native SSSD on its own candidates. We extend these scores to the new sources by their natural analogues, a frequency ratio of 1 for a prompt-lookup continuation and the target's probability for a recycled candidate, a scale unrelated to frequency ratios; these extensions are controls, not configurations of SSSD. Per-source isotonic calibration limits the damage but still falls up to 3.7% below native SSSD. In contrast, CalTree-LR, refitted on each pool, gains from every added source, reaching 4.7%, and CalTree-GB rises from 5.0% to 6.2% at $D = 8$ with prompt lookup and recycling. The same holds on Llama-3.1-8B, whose tokenizer differs: with prompt lookup and recycling added, native and frequency scores lose 4.8–6.0% at $D \in \{8, 16\}$, whereas CalTree-LR rises from 3.4–4.1% to 4.4–5.7% and CalTree-GB reaches 6.6–8.3%. These results cover the tested pools, not arbitrary sources, and are replay measurements: end-to-end gains and the cost of exposing recycled distributions are not evaluated here.

Table 13: CanItEdit served by SGLang on one A100-80GB (Qwen2.5-7B): throughput of autoregressive decoding (AR), speedup over AR (×AR) and change over SSSD (Δ%, largest in bold) of the median throughput over three runs, and tokens per verification pass τ counted by the engine.

	AR	SSSD		Iso	CalTree-LR			CalTree-GB	
Batch (D)	tok/s	×AR	τ	Δ%	×AR	Δ%	τ	Δ%	τ
1 (32)	74	2.38	3.38	+0.2	2.44	**+2.6**	3.50	−0.1	3.51
8 (16)	516	1.58	3.24	−0.3	1.65	**+4.0**	3.43	+1.9	3.44

Table 14: **Refitted on each enlarged pool, CalTree-LR gains from every added source; fixed extensions of native and frequency scores mostly lose.** Qwen2.5-7B on Spec-Bench: gain in τ (%, $D = 8$ / $D = 16$) over native SSSD on its own candidates when prompt lookup (PLD) and token recycling (TR; warm: initialized from the datastore generations) add candidates. Iso, LR, and GB are refitted on each pool; native and frequency scores use fixed extensions to the new sources, which are our controls rather than SSSD configurations (bold: best non-oracle per pool and budget).

Score	SSSD	+PLD	+PLD+TR	+TR (warm)
Native (SSSD)	0.0 / 0.0	−4.3 / +0.3	−6.0 / −7.2	−15.1 / −14.5
Frequency product	−2.9 / −2.1	−4.6 / −2.8	−6.1 / −7.4	−11.6 / −13.7
Iso (source×depth)	+0.8 / +1.1	+0.5 / +1.2	−0.8 / −0.3	−3.7 / −3.5
CalTree-LR	+3.8 / +3.3	+3.9 / +3.7	+4.1 / +4.6	+4.1 / +4.7
CalTree-GB	**+5.0 / +4.4**	**+4.9 / +4.6**	**+6.2 / +6.9**	**+6.3 / +7.2**
Oracle	+27.5 / +19.3	+27.6 / +19.7	+42.8 / +35.7	+48.2 / +40.8