# SEMANTICTREE: GRAMMAR-AWARE TREE ALLOCATION FOR SPECULATIVE DECODING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Tree-based speculative decoding allocates candidate tokens before target verification. In structured generation, grammar checks come too late to reallocate this budget: 73–77% of non-root slots in EAGLE3 trees contain grammar-invalid candidates. We call this *semantic waste*. **SemanticTree** excludes unreachable candidates during tree allocation while preserving the proposer's scores and budget. A parser-state mask cache shares work across branches and rounds, and verification reuses the masks obtained during drafting. We integrate SemanticTree with an autoregressive drafter, EAGLE3, and a block drafter, DDTree. Across 35 drafter–target–workload settings at batch size one, verification rounds fall by 16.4–29.8% with EAGLE3 and 17.2–25.8% with DDTree; decode throughput rises by 19.5–40.3% and 15.5–25.6%, respectively. A cache ablation preserves all round savings but reduces the throughput gain from 33% to 7% on Qwen3-4B/Nexus with EAGLE3, showing that recovering semantic waste requires both effective allocation and low grammar-processing cost.

## 1 INTRODUCTION

Speculative decoding accelerates generation by letting a target model verify several cheaply drafted tokens in one pass (Leviathan et al., 2023; Chen et al., 2023). Tree-based variants spend a fixed budget on alternative prefixes, so the target can accept whichever branch agrees with its continuation (Miao et al., 2024; Li et al., 2024b; 2025; Ringel and Romano, 2026). In many deployments that continuation must also obey a JSON schema, a function signature or another grammar. Serving engines enforce this requirement by masking illegal next tokens (Dong et al., 2025; Zheng et al., 2024).

These two mechanisms operate at different times. A grammar state advances one token at a time along a prefix; a target pass verifies many speculative prefixes at once. The tree-based serving implementations we study therefore build the tree first, then traverse it to prepare grammar masks while the target forward runs. The output remains legal, but the speculative budget has already been spent. An unreachable candidate, together with its descendants, occupies slots the target can never accept. Across EAGLE3 trees on four targets and five tool-calling workloads, this *semantic waste* consumes 73–77% of the 59 non-root slots on average per target.

Figure 1 shows the opportunity. The illustrative trees in (a) and (b) have the same eight-slot budget: excluding illegal candidates before allocation lets other candidates take their place and can extend the accepted path. Panel (c) measures reachable slots with EAGLE3 alone and with SemanticTree, averaging the five workloads per target. Baseline trees are replayed from logs; SemanticTree trees come from a separate profile pass, so these are aggregate occupancy measurements, not matched individual trees (Appendix A).

Recovering this capacity requires two things. First, the proposer must supply reachable candidates that the target actually accepts; a legal but wrong replacement saves no rounds. Second, obtaining grammar masks inside drafting must cost little enough to preserve those savings. Autoregressive decoding can hide one mask behind each forward pass. A speculative tree may need tens of masks per round, and each expansion depends on its parents' masks. Moving the grammar earlier can therefore put host work directly in the GPU's path.

We present **SemanticTree**, which applies exact reachability during tree allocation without changing the proposer's scores or budget. Its system design uses a property of structured generation: different



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.

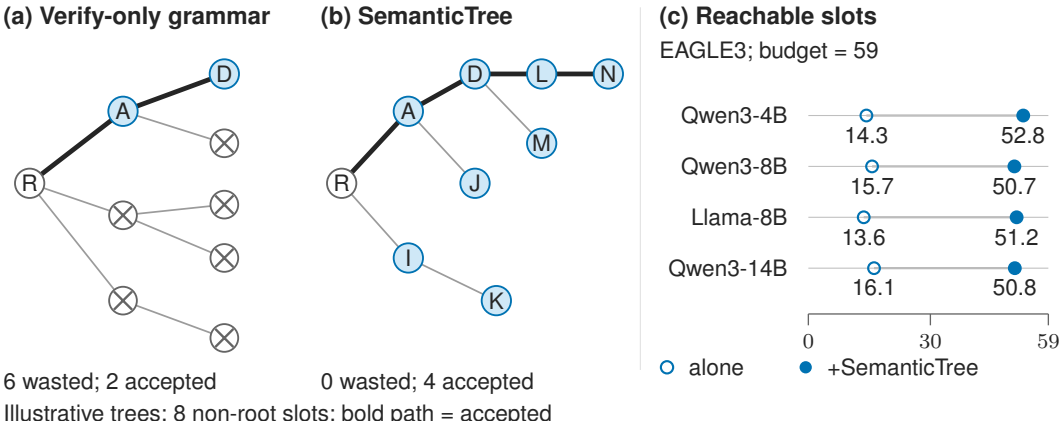


Figure 1: **Semantic waste and allocation at a fixed budget.**

branches and successive rounds repeatedly visit the same parser states. We add a parser-state hash to XGrammar, cache the corresponding masks, and reuse them at verification. An autoregressive drafter obtains these masks inside one captured draft graph; a block drafter uses them in a staged host planner. The same eligibility rule thus serves both EAGLE3 and DFlash with DDTree.

Our contributions are: (i) quantifying semantic waste in verification trees and incorporating exact eligibility into fixed-budget tree allocation while preserving the proposer's ranking of legal candidates; (ii) making this allocation practical through parser-state mask reuse across branches and rounds, with frontier exchange inside a single captured EAGLE3 draft graph and a staged host planner for DDTree; and (iii) showing, for both drafter classes, that recovered slots become faster decoding because mask reuse keeps the round cost nearly unchanged.

## 2 Recovering Semantic Waste

### 2.1 Eligibility and the two conditions for recovery

Let $g_p$ be the grammar state after prefix $p$, $A(g_p)$ its legal next-token set, and $m : V_d \to V$ the mapping from draft to target tokens. The draft-side legal set is

$$A_d(g_p) = \{u \in V_d : m(u) \in A(g_p)\}. \tag{1}$$

A candidate $u$ under $p$ is *reachable* if $u \in A_d(g_p)$. An unreachable node and all its descendants can never lie on the path accepted by the grammar-constrained target. The committed non-root slots they occupy are semantic waste. A reachable node, however, can still disagree with the target: eligibility rules out impossible tokens but does not rank the remaining ones. The first recovery condition is therefore *proposer supply*: the freed slots must extend the continuation the target will accept.

The second condition concerns *round cost*. For a fixed output length, let $R$ and $c$ denote the number of verification rounds and average time per round, and use primes for the modified decoder. Its decode speedup is

$$\underbrace{\frac{Rc}{R'c'}}_{\text{decode speedup}} = \underbrace{\frac{R}{R'}}_{\text{round savings}} \underbrace{\frac{c}{c'}}_{\text{round-cost factor}}. \tag{2}$$

For example, $20\%$ fewer rounds would yield $25\%$ higher throughput at equal round cost. Extra grammar work reduces that gain. This decomposition separates whether legal candidates improve speculation from whether obtaining them is cheap enough.

### 2.2 SEMANTICTREE: ELIGIBILITY DURING ALLOCATION

For a kept parent $p$ with cumulative path score $s(p)$ and draft log-probability $\ell_p(u)$, SemanticTree ranks its children by

$$\widetilde{s}(p,u) = \begin{cases} s(p) + \ell_p(u), & u \in A_d(g_p), \\ -\infty, & u \notin A_d(g_p). \end{cases} \tag{3}$$

The next highest-scoring reachable candidates take the freed slots. Each kept edge forks its parent's grammar matcher, accepts $m(u)$, and carries the new state to the next expansion. Scores are not renormalized over the legal subset: renormalization preserves sibling order but changes cumulative scores and hence competition between parents.

The rule acts at each allocator's existing selection point. EAGLE3 expands parents depth by depth, one draft forward per depth; eligibility enters each parent's Top-$K$ (Li et al., 2025). DFlash proposes a block in one forward (Chen et al., 2026), and DDTree allocates a tree best-first over its position-wise marginals (Ringel and Romano, 2026); eligibility filters the candidates considered by that heap. We use *DDTree* for this pair. Topology, ancestor closure and the physical verification budget follow the underlying allocator.

Algorithm 1 summarizes the shared allocation interface. The allocator $\mathcal{A}$ retains its expansion, selection and finalization rules: EAGLE3 processes frontiers, whereas DDTree uses its heap. $\mathcal{M}$ stores masks already obtained for tree nodes; $\mathrm{Pad}_N$ fills unused verification slots with zero mask rows. Parents with fewer legal candidates leave unused capacity rather than introducing illegal edges.

**Algorithm 1** Grammar-aware allocation with the original proposer scores

**Input:** root $r$, matcher $g_r$, allocator $\mathcal{A}$, verification budget $N$.

$T \leftarrow \{r\}$; $s(r) \leftarrow 0$; $\mathcal{M} \leftarrow \varnothing$
**while** $\mathcal{A}.\mathrm{CanExpand}(T, N)$ **do**
  $(p, \ell_p) \leftarrow \mathcal{A}.\mathrm{Next}(T)$
  $\mathcal{M}[p] \leftarrow \mathrm{CachedMask}(g_p)$
  $C_p \leftarrow \{u \in V_d : \mathcal{M}[p]_{m(u)} = 1\}$
  $U_p \leftarrow \mathcal{A}.\mathrm{Select}\big(\{(u, s(p) + \ell_p(u)) : u \in C_p\}\big)$
  **for** $u \in U_p$ **do**
    $g_{pu} \leftarrow \mathrm{Accept}(\mathrm{Fork}(g_p), m(u))$
    $s(pu) \leftarrow s(p) + \ell_p(u)$; $T \leftarrow \mathrm{Attach}(T, p, u)$
$T \leftarrow \mathcal{A}.\mathrm{Finalize}(T, N)$
**for** $v \in V(T) \setminus \mathrm{dom}(\mathcal{M})$: $\mathcal{M}[v] \leftarrow \mathrm{CachedMask}(g_v)$
**return** $\mathrm{Pad}_N(T, \mathcal{M})$

The target still verifies under the grammar mask. Both the proposer-alone baseline and SemanticTree therefore enforce legality; their difference is whether reachability enters allocation. XGrammar supplies the feasibility primitive (Dong et al., 2025). The remaining challenge is making its masks available when the allocator needs them.

**Verification invariant.** At a committed prefix, the verifier accepts a draft token only when it agrees with the grammar-masked target choice; a disagreement returns control to the target. With identical target logits, exact masks and the same tie-breaking, changing the candidate tree changes the number of tokens accepted in a round, but not the constrained-greedy continuation. This argument assumes that tree verification evaluates each prefix with its correct causal history. It does not guarantee bitwise agreement between executions with different tree shapes; we distinguish that property from mask and planner validation below.

## 3 KEEPING THE GRAMMAR WORK SMALL

An EAGLE3 round can require up to 60 masks per request. Grammar work becomes *exposed* when the GPU waits for those masks, either inside drafting or after the target forward. SemanticTree reduces the repeated work before scheduling it in the two draft loops.

### 3.1 SHARED MASKS ACROSS BRANCHES AND ROUNDS

Tool-call grammars revisit the same parser states, for example within strings or at the same position of successive objects. Different prefixes can therefore require identical next-token masks. We add a 128-bit state hash to XGrammar: it combines the matcher's current Earley states with the contents of the parent states into which they can complete. Token offsets are excluded so equivalent states at different positions share a key. Cyclic parent structures receive no hash and bypass caching. A least-recently-used cache is keyed by the grammar, stop tokens and state hash; a miss computes and stores the exact mask.

The repetition is large (Table 1). With EAGLE3, a SemanticTree round requests 59–78 masks, for the frontiers of seven depths and the verification leaves, but computes only 0.7–3.7 of them; every other request is a cache hit. An offline replay over the five workloads finds 16,884 visited parser states but only 2,510 distinct ones, and Nexus, whose grammar has the costliest masks, revisits 643 distinct states 7,379 times. The number of masks a tree needs thus grows with its size, while the number that must be computed grows only with the new states a round reaches.

Table 1: **Masks requested and computed per SemanticTree round**, EAGLE3, $B{=}1$, range over the four targets. Requested: drafting frontiers and verification leaves; computed: cache misses. Parser states: visited and distinct states in an offline replay of the five workloads.

| | masks per round | | | parser states |
|---|---|---|---|---|
| Workload | requested | computed | cache hit (%) | visited → distinct |
| BFCL-simple | 59–65 | 2.5–3.0 | 95.2–96.1 | 1,764 → 391 |
| BFCL-multiple | 67–70 | 1.7–3.2 | 95.2–97.5 | 2,278 → 538 |
| BFCL-multiturn | 66–71 | 0.7–1.3 | 98.1–99.0 | 2,472 → 338 |
| Nexus | 69–78 | 0.7–1.1 | 98.4–99.1 | 7,379 → 643 |
| ToolHop | 71–76 | 3.1–3.7 | 94.7–95.9 | 2,991 → 600 |

Verification reuses this work. Expanded nodes already have masks, so the verification bitmask is assembled from those rows; only unexpanded leaves need masks, usually cache hits. Padding and slots a parent could not fill receive zero rows. This removes the baseline's second traversal of the finished tree. The state hash and online cache-hit checks are detailed in Appendix E.

**Validation and scope.** We check mask reuse by recomputing every cache hit and comparing its bitmask with the cached value. Across 640 EAGLE3 requests on four targets at batch size one and four concurrent requests, all 547,967 checked hits agree; all 17,890 hits agree on 240 DDTree requests across three targets at batch size one. We also compare DDTree's staged planner with a per-parent reference implementation: verification-round counts and output tokens agree on all 120 online cases. These checks validate the cache and planner optimizations on the tested cases; they do not establish output identity between the proposer alone and SemanticTree. The target still applies its grammar mask at verification, and our integrations are evaluated only for greedy decoding on a single tensor-parallel rank.

Figure 2 shows how this shared mask service supports the two allocators. It is a dependency schematic, not a measured timeline. EAGLE3 alternates frontier publication and masked selection within one captured GPU graph. DDTree places best-first allocation on the host between the draft and target forwards, using staged candidates and returning to the device only when that pool is insufficient.

### 3.2 AUTOREGRESSIVE DRAFTER: ONE CAPTURED DRAFT GRAPH

In EAGLE3, the parents at depth $i{+}1$ are known only after masked Top-$K$ at depth $i$. A direct host implementation would break drafting into one GPU launch per depth with synchronization in between. We instead capture the whole loop, including the root Top-$K$, as one CUDA graph. At each depth a kernel publishes the frontier's tokens, validity bits and parent indices to pinned memory. The host forks the parent matchers, accepts the frontier tokens and obtains their masks from the shared cache. A device-side handshake and captured copy deliver the masks to the next selection step.

The root-mask wait is a median of 85–123 $\mu$s per round at batch size one. Round-numbered flags prevent stale handshakes, and a timeout with a sticky error flag prevents a failing host from hanging

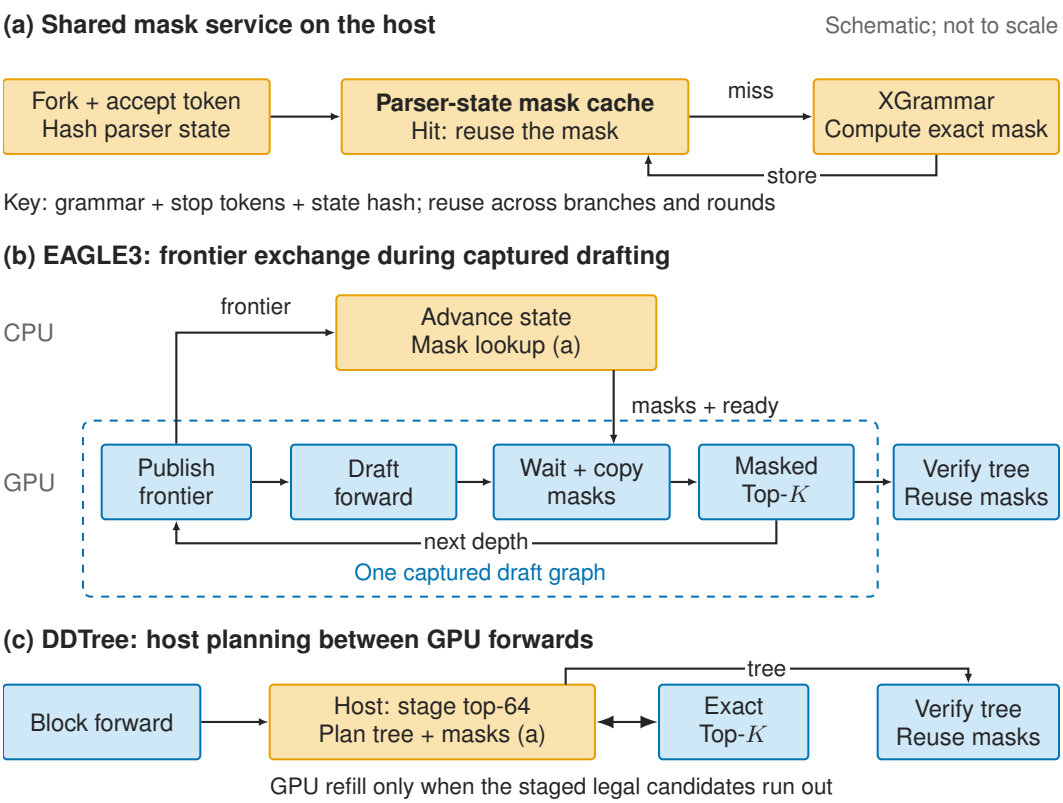


Figure 2: **Shared parser-state masks in two tree allocators.**

the GPU. No node in the graph calls back into the host. Appendix F gives the frontier and handshake details.

### 3.3 Block drafter: a staged host planner

DDTree's planner runs after the draft forward and before target verification, so its time directly lengthens the round. We copy the drafter's top-64 candidates per position, their normalizers, the root token and the prefix length to the host once. Each planned node forks its parent's grammar state; legality checks are bit lookups in its cached mask. The planner consumes the staged candidates lazily and requests exact masked Top-$K$ over the full vocabulary only when a parent's legal candidates run out, 1.1–3.0 times per round in the evaluated workloads.

The DDTree baseline uses the same staging, preserving the comparison's allocator and data movement. The comparison with the per-parent reference planner is reported in Section 3.1.

## 4 Evaluation

We evaluate the two factors in Eq. (2): how many rounds are recovered and what each recovered round costs. The main comparison always pairs a drafter with and without SemanticTree at the same budget.

### 4.1 Setup and measurement

**Configuration.** We use five tool-calling workloads: BFCL-simple, BFCL-multiple, BFCL-multiturn, Nexus and ToolHop (Patil et al., 2025; Nexusflow.ai Team, 2023; Ye et al., 2025). Every output is a schema-constrained call. EAGLE3 uses width $K{=}10$, depth 7 and 60 draft tokens on Qwen3-4B, Qwen3-8B, Llama-3.1-8B-Instruct and Qwen3-14B. DDTree uses a DFlash block of 16 and 15 non-root slots on Qwen3-4B and Qwen3-8B; Llama-3.1-8B uses its available block-10 drafter

and 9 slots. There is no Qwen3-14B DFlash drafter. All main runs use batch size one, SGLang with XGrammar and RTX 5880 Ada GPUs. Serving settings are in Appendix A.

**Replay construction.** Each request pairs a fixed prompt with a JSON schema for one object containing `name` and `arguments`. The prompt includes the available function descriptions and user request. The schema couples each allowed function name to its argument schema, using `oneOf` when several functions are available; BFCL parameter types are normalized to JSON Schema types. BFCL-simple and BFCL-multiple use the supplied conversation. BFCL-multiturn replays individual reference actions: the prompt contains the conversation through the current turn and the preceding reference tool calls, and the candidate functions are restricted to the API namespace identified from that reference action. Nexus uses single-turn benchmark queries. ToolHop replays a subtask with the original question as context and all tools available to that question. Tool execution and namespace routing are outside the timed workload. For each target, constrained target-only greedy decoding produces a reference output; we retain nonempty outputs that terminate before the token cap. Both arms then generate afresh from the same prompt and schema; the reference output is not supplied as a forced continuation. These workloads measure constrained generation from fixed contexts, rather than live tool execution or the benchmarks' official task-success scores.

**Protocol.** Each target uses the same seed subset of 192 cases per BFCL workload and 64 each from Nexus and ToolHop, subject to the reference-completion filter above. We measure rounds over three timed repetitions and throughput with a streaming probe whose window excludes the time to the first token. Paired comparisons retain cases whose probes decode equal numbers of tokens in both arms, excluding at most 11 cases per setting. Output text need not be identical: numerical differences between tree shapes can change a greedy choice. Appendix B reports counts, absolute rates and EAGLE3's sensitivity to retaining all cases.

**Metrics.** We pool rounds over cases and report their change $\Delta R$ relative to the same drafter alone. Decode throughput pools decoded tokens over decode time; the *decode gain* is its percentage change. All confidence intervals are paired case-bootstrap 95% intervals with 10,000 resamples. They measure case-to-case variation, not run-to-run variation.

### 4.2 End-to-end decode gains

Figure 3 gives every paired throughput comparison, with targets separated into panels and a common percentage scale. SemanticTree raises decode throughput by 19.5–40.3% with EAGLE3 and 15.5–25.6% with DDTree. All 35 intervals lie above zero; the lowest lower bound is +13.1%. The effect holds across both drafting strategies, while its size varies with the target and workload. Qwen3-8B gains 34.6–40.3% with EAGLE3, compared with 20.0–28.0% on Qwen3-14B.

For context, Appendix B also compares with target-only grammar-constrained decoding (AR). Averaged geometrically over workloads per target, SemanticTree lifts EAGLE3 from 1.21–2.24× AR to 1.55–2.76×, and DDTree from 2.29–2.69× to 2.75–3.28×. These use a separate earlier AR measurement and a cohort matched across all methods, so the paired drafter comparisons in Figure 3 remain the primary result.

**Batched serving.** At steady-state batch sizes four and eight (Appendix D), SemanticTree keeps EAGLE3's round savings at 15.3–29.7% and its decode gain at 17.1–41.7%; all 40 paired intervals lie above zero, the lowest bound at +11.4%, and a SemanticTree round costs within ±2.9% of an EAGLE3 round. Exposed grammar time grows with the $B \cdot K$ frontier, to 0.37–0.47 ms at $B$=4 and 0.71–1.94 ms at $B$=8, and exceeds EAGLE3's in 37 of the 40 settings.

### 4.3 From reachable slots to fewer verification rounds

The occupancy measurements in Figure 1 show that EAGLE3 can refill most of the excluded slots: the unreachable share falls from 62–82% to 6–19% per setting. More reachable nodes alone do not establish target agreement; the round counts provide that test. Across individual workloads, verification rounds fall by 16.4–29.8% with EAGLE3 and 17.2–25.8% with DDTree (Appendix B). Every target–workload pair saves rounds. Table 2 summarizes how increased accepted length and

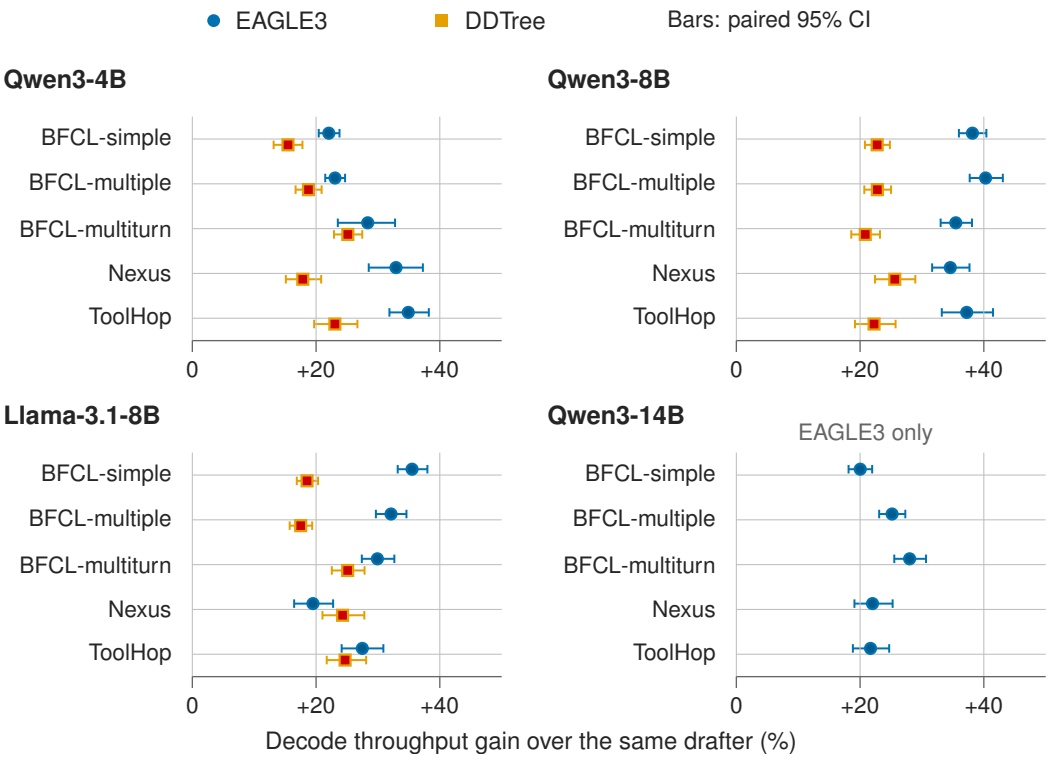


Figure 3: **Paired decode-throughput gains in all 35 settings.**

the change in round cost translate into throughput gains for each target and drafter. DDTree's round savings support the supply condition through accepted progress, although we do not measure its slot occupancy.

**Where recovery stops.** The residual 6–19% is exactly the capacity SemanticTree leaves unfilled: 3.4–11.4 of the 59 slots per round belong to parents that have fewer legal candidates than their share of the budget, and verification writes zero rows for them. This unfilled capacity is largest on BFCL-simple (10.4–11.4 slots) and BFCL-multiturn (7.6–10.6), and smallest on Nexus (3.4–6.4). Round savings do not follow it closely: across the 20 EAGLE3 settings, the correlation between unfilled slots and $\Delta R$ is only 0.31. What bounds recovery is therefore less how many slots are refilled than whether the target accepts the legal candidates that refill them, the supply condition of Section 2; a proposer that ranks legal candidates poorly would recover fewer rounds from the same trees.

Table 2: **From accepted tokens to decode throughput,** $B{=}1$**.** Alone → +ST compares the same drafter and budget on token-matched cases. Geometric means over five workloads; percentage changes are computed from ratios. Time/round is inferred as accepted tokens/round divided by decode throughput. Appendix B gives per-workload results and paired intervals.

| Target | Drafter | Tokens/round alone → +ST | Rounds saved (%) | Time/round change (%) | Decode gain (%) |
|---|---|---|---|---|---|
| Qwen3-4B | EAGLE3 | 2.23 → 2.86 | 22.1 | +0.1 | **+28.2** |
| | DDTree | 3.75 → 4.81 | 21.9 | +6.8 | **+20.0** |
| Qwen3-8B | EAGLE3 | 2.50 → 3.47 | 27.9 | +1.1 | **+37.1** |
| | DDTree | 3.67 → 4.72 | 22.4 | +4.8 | **+22.8** |
| Llama-3.1-8B | EAGLE3 | 2.40 → 3.11 | 23.1 | +1.0 | **+28.8** |
| | DDTree | 3.86 → 4.86 | 20.5 | +3.2 | **+22.0** |
| Qwen3-14B | EAGLE3 | 3.47 → 4.32 | 19.7 | +1.0 | **+23.3** |

These savings largely survive the round-cost factor. An EAGLE3 round costs 0.1–2.1% more in 19 of 20 settings and 6.0% less on Qwen3-4B/Nexus, where the baseline's own grammar traversal is exposed. A DDTree round costs 2.4–7.7% more, reflecting the added host planning. Together with the round ratios, these costs explain the throughput gains in Figure 3. Figure 4 draws the two factors to scale for one profiled Qwen3-4B/BFCL-simple case. Five EAGLE3 rounds and four SemanticTree rounds decode the same 11.8 tokens, while a SemanticTree round is only 0.7 ms longer (a); within the longer draft phase (b), drawn with seven equal depths, the GPU waits 87 $\mu$s for the root masks, and the host spends about 0.1 ms on grammar work per depth.

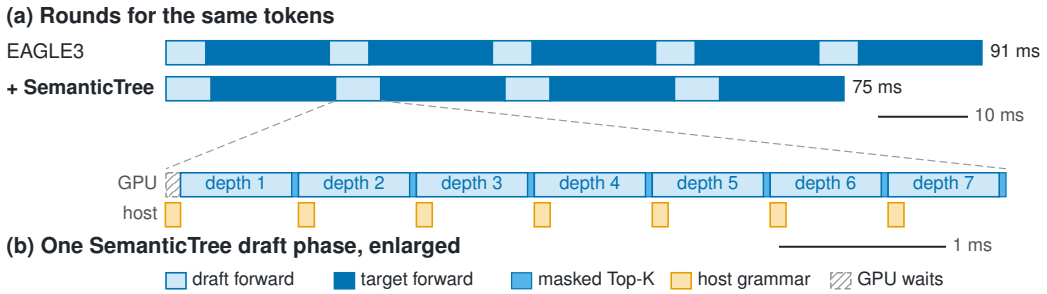


Figure 4: **EAGLE3 with and without SemanticTree, to scale.**

### 4.4 Grammar cost and the cache ablation

**Holding round savings fixed.** We disable the cache on Qwen3-4B for BFCL-multiple and Nexus. This changes how masks are obtained, but not the trees or verification rounds. In Figure 5, (a) shows the unchanged round reduction, (b) the change in average round time, and (c) the resulting decode gain. Every change is relative to the same drafter alone, using cases token-matched across all three arms; only the decode-gain panel has confidence intervals.

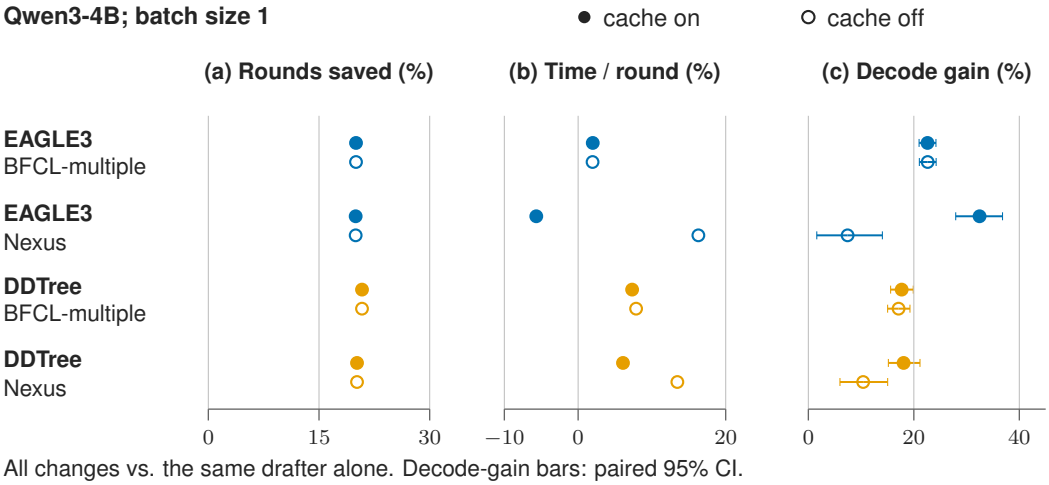


Figure 5: **The cache changes the cost of recovering the same rounds.**

All four settings save 20.0–20.8% of rounds with or without the cache. On BFCL-multiple, masks are already cheap: EAGLE3's host grammar work stays near 1.0 ms per round and its decode gain remains 23%. On Nexus, disabling the cache raises EAGLE3's host grammar work from 1.1 to 13.2 ms and its exposed time from 0.14 to 8.0 ms. A round then costs 16% more than the baseline instead of 6% less, reducing the decode gain from 33% to 7%. DDTree shows the same direction: Nexus planning grows from 0.9 to 3.1 ms and the gain falls from 18% to 10%. The cache is therefore consequential when masks are expensive; its benefit is small when mask computation is already cheap. Table 11 gives the absolute round counts, times and intervals.

**Measured grammar work.** Across the EAGLE3 settings, the cache serves $94.7$–$99.1\%$ of mask requests; host grammar work stays at $0.9$–$1.2$ ms per round and exposed grammar time at $0.14$–$0.27$ ms. The baseline exposes only $0.04$–$0.05$ ms outside Nexus, but up to $2.50$ ms on Nexus. Thus SemanticTree adds a small dependency on most workloads while reducing a larger one on the expensive grammar. With DDTree, cache hits cover $77.8$–$97.7\%$ of requests, and host planning adds $0.6$–$1.4$ ms to a round. Appendix C gives the full profile, including the distinct host and exposed measurements.

## 5 Related Work

**Grammar-constrained generation.** Outlines and SynCode index legal tokens using finite-state or lexer automata (Willard and Louf, 2023; Ugare et al., 2024). XGrammar separates context-independent token checks from parser-stack work and overlaps mask computation with GPU execution (Dong et al., 2025); XGrammar-2 reuses substructures across grammars (Li et al., 2026). Formatron caches Earley states, and other engines reduce preprocessing or avoid model calls for grammar-determined tokens (Sun et al., 2025; Park et al., 2025; Zheng et al., 2024). JSONSchemaBench compares engine efficiency and coverage (Geng et al., 2025). SemanticTree uses exact masks during speculative allocation and shares them across a tree's branches and successive rounds. Work on grammar-aligned sampling addresses the distributional effect of masking (Park et al., 2024); our evaluation is greedy.

**Combining constraints and speculation.** SGLang with XGrammar applies grammar masks to completed verification trees (Zheng et al., 2024; Dong et al., 2025). TensorRT-LLM advances grammar state inside captured host callbacks for draft chains, leaving tree drafting out of scope (NVIDIA TensorRT LLM Team and XGrammar Team, 2026). DOMINO combines subword-aligned constraints with count-based chain speculation (Beurer-Kellner et al., 2024), and constrained speculative lookaheads combine draft proposals with target and reward verification (Nakshatri et al., 2025). Future-validity corrections address grammar-conditional speculative sampling (Nie et al., 2026). ToolSpec and OoO-Spec specialize what is drafted for tool calls (Xia et al., 2026; Zhang et al., 2026). Our question concerns how a fixed tree budget is allocated under an existing proposer, and how per-node grammar work fits into that allocation.

**Tree allocation and execution.** Medusa, EAGLE and its successors improve draft proposals, while SpecInfer and DDTree organize alternatives for tree verification (Cai et al., 2024; Li et al., 2024a;b; 2025; Miao et al., 2024; Ringel and Romano, 2026). Sequoia, SpecExec, OPT-Tree, TALON, SMART and Bastion allocate finite budgets using shape, likelihood, confidence or runtime cost (Chen et al., 2024; Svirschevski et al., 2024; Wang et al., 2025; Liu et al., 2026; Wang and Zhou, 2026; Oh et al., 2026). Pruning and retrieval can also redirect speculative capacity (Zheng and Wang, 2025; Shen et al., 2026). Exact reachability supplies an additional eligibility check: it removes only candidates the constrained target cannot accept while retaining the learned ranking. PEARL and lookahead decoding change dependencies between draft and target work (Liu et al., 2025; Fu et al., 2024); SemanticTree additionally reduces host grammar work by reusing masks before scheduling what remains.

## 6 Limitations and Conclusion

We evaluate greedy structured generation with one tensor-parallel rank, targets up to $14$B, seed subsets and fixed budgets. Prepared tool-call replays do not measure interactive task success, and we do not establish output identity between paired arms. We measure EAGLE3's slot occupancy but not DDTree's, and do not test a proposer unable to refill freed slots. Reachability cannot help when nearly all candidates are legal or legal candidates disagree with the target. Appendix D reports EAGLE3 at batch sizes four and eight; exposed grammar time grows with the $B \cdot K$ frontier even though it remains a small part of a round in those measurements.

SemanticTree recovers semantic waste by applying exact constraints during tree allocation. Across both drafter classes, parser-state reuse turns round savings into throughput gains. The cache ablation shows why both are needed: identical trees and round counts yield different speedups when mask computation reaches the critical path.

## REPRODUCIBILITY STATEMENT

Sections 2 and 3 describe the allocation rule and its two drafter integrations. Appendix A specifies the serving configuration, metric definitions and bootstrap procedure; Appendix B reports per-setting results, sample exclusions and the separately measured AR reference. Appendices C and D document the profiling, cache-ablation and batched-serving protocols. Appendices E and F describe the parser-state hash, its validation and the captured frontier exchange.

## AI USE STATEMENT

Generative AI tools assisted research planning, software implementation and debugging, analysis, and manuscript drafting and editing. All numerical exhibits were regenerated from the experiment artifacts. The authors are responsible for the claims, code, citations and final submission.

## REFERENCES


Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024.

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. arXiv:2401.10774.

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.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. *arXiv preprint arXiv:2602.06036*, 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, 2024.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and Efficient Structured Generation Engine for Large Language Models. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the Sequential Dependency of LLM Inference Using Lookahead Decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 14060–14079, 2024.

Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. JSONSchemaBench: A Rigorous Benchmark of Structured Outputs for Language Models. *arXiv preprint arXiv:2501.10868*, 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*, 2023.

Linzhang Li, Yixin Dong, Guanjie Wang, Ziyi Xu, Alexander Jiang, and Tianqi Chen. XGrammar-2: Dynamic and Efficient Structured Generation Engine for Agentic LLMs. In *Proceedings of the ACM Conference on AI and Agentic Systems (CAIS)*, 2026.

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*, 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*, 2024b.

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, 2025.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. PEARL: Parallel Speculative Decoding with Adaptive Draft Length. In *International Conference on Learning Representations*, 2025.

Tianyu Liu, Qitan Lv, Yuhao Shen, Xiao Sun, and Xiaoyan Sun. TALON: Confidence-Aware Speculative Decoding with Adaptive Token Trees. *arXiv preprint arXiv:2601.07353*, 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 Generative Large Language Model Serving with Tree-based Speculative Inference and Verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024. arXiv:2305.09781.

Nishanth Sridhar Nakshatri, Shamik Roy, Rajarshi Das, Suthee Chaidaroon, Leonid Boytsov, and Rashmi Gangadharaiah. Constrained Decoding with Speculative Lookaheads. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4681–4700, 2025.

Nexusflow.ai Team. NexusRaven-V2: Surpassing GPT-4 for Zero-shot Function Calling. `https://github.com/nexusflowai/NexusRaven-V2`, 2023. Model release with the Nexus function-calling benchmark.

Wenhua Nie, Zijie Meng, Kun Zou, Zheng Lin, Ziwei Li, Haoran Zheng, Jyh-Shing Roger Jang, and Hao Zhang. Future Validity is the Missing Statistic: From Impossibility to Φ-Estimation for Grammar-Faithful Speculative Decoding. *arXiv preprint arXiv:2605.07698*, 2026.

NVIDIA TensorRT LLM Team and XGrammar Team. Combining Guided Decoding and Speculative Decoding: Making CPU and GPU Cooperate Seamlessly. `https://nvidia.github.io/TensorRT-LLM/1.3.0rc7/blogs/tech_blog/blog12_Combining_Guided_Decoding_and_Speculative_Decoding.html`, 2026. TensorRT-LLM technical blog; page last updated March 9, 2026; accessed September 25, 2026.

Soowon Oh, Nam Cao, Yujin Kim, Hojung Jung, Huzama Ahmad, Sangmin Bae, and Se-Young Yun. Bastion: Budget-Aware Speculative Decoding with Tree-structured Block Diffusion Drafting. *arXiv preprint arXiv:2605.29727*, 2026.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. Grammar-Aligned Decoding. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Kanghee Park, Timothy Zhou, and Loris D'Antoni. Flexible and Efficient Grammar-Constrained Decoding. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48262–48275, 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, pages 48371–48392, 2025.

Liran Ringel and Yaniv Romano. Accelerating Speculative Decoding with Block Diffusion Draft Trees. *arXiv preprint arXiv:2604.12989*, 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.

Xintong Sun, Chi Wei, Minghao Tian, and Shiwen Ni. Earley-Driven Dynamic Pruning for Efficient Structured Decoding. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

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*, volume 37, 2024.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. SynCode: LLM Generation with Grammar Augmentation. *arXiv preprint arXiv:2403.01632*, 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*, 2025. arXiv:2406.17276.

Lifu Wang and Pan Zhou. SMART: When is it Actually Worth Expanding a Speculative Tree? *arXiv preprint arXiv:2604.09731*, 2026.

Brandon T. Willard and Rémi Louf. Efficient Guided Generation for Large Language Models. *arXiv preprint arXiv:2307.09702*, 2023.

Heming Xia, Yongqi Li, Cunxiao Du, Mingbo Song, and Wenjie Li. ToolSpec: Accelerating Tool Calling via Schema-Aware and Retrieval-Augmented Speculative Decoding. *arXiv preprint arXiv:2604.13519*, 2026.

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. ToolHop: A Query-Driven Benchmark for Evaluating Large Language Models in Multi-Hop Tool Use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2995–3021, 2025. doi: 10.18653/v1/2025.acl-long.150.

Zhiheng Zhang, Mujie Xu, Feiyu Sun, and Zhixin Zhang. OoO-Spec: Out-of-Order Semantic Speculation for Fast Tool Calling. *arXiv preprint arXiv:2608.00814*, 2026.

Huanran Zheng and Xiaoling Wang. Faster Speculative Decoding via Effective Draft Decoder with Pruned Candidate Tree. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9856–9868, 2025.

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, 2024.

## A Experimental Protocol

**Methods.** Each proposer alone (EAGLE3 or DDTree) uses its own allocator with grammar-enforced verification and no eligibility during allocation; EAGLE3 builds the verification bitmask with XGrammar's native draft-tree traversal after launching the target forward. Adding SemanticTree makes eligibility part of allocation and reuses drafting-time grammar state for the verification bitmask (Section 3). Both DDTree arms use the staged host planner; without eligibility it reduces to DDTree's own best-first allocation over the staged candidates. Target-only constrained decoding supplies each target's reference outputs.

**Metrics.** Decode throughput is $\sum_i N^{(i)}_{\text{decode}} / \sum_i T^{(i)}_{\text{decode}}$ over the streaming probes of a setting's cases, where the decode window starts at the first streamed chunk, so prefill and the time to the first token are excluded. The round reduction pools verification rounds over the cases. MAT (mean accepted tokens per round) is pooled as well: $\sum(\text{completion tokens} - 1) / \sum$ verification rounds, where the token produced by prefill is removed. The resampling unit of every interval is the case, with one index draw per replicate shared by both arms; intervals are percentile intervals from $10{,}000$ resamples at seed `20260921`.

**Unreachable slots.** For EAGLE3, the unreachable share is measured on the verification trees themselves: XGrammar's draft-tree traversal zeroes the bitmask row of every node whose path the grammar rejects, and the remaining rows are the reachable nodes. For the proposer alone we replay up to $150$ logged trees per target and workload; for SemanticTree we count the zero rows its verification bitmask writes for slots a parent could not fill.

Table 3: **Serving settings**, identical for a proposer with and without SemanticTree.

| Setting | Value |
|---|---|
| EAGLE3 tree | width $K{=}10$, depth 7, 60 draft tokens |
| DFlash + DDTree tree | block 16, 15 non-root slots, candidate pool 64 |
| DFlash + DDTree on Llama-3.1-8B | block 10, 9 non-root slots, candidate pool 64 |
| Max new tokens | 256; 768 for BFCL-multiturn |
| Serving | SGLang v0.5.18, XGrammar v0.2.1 with the state hash, radix cache off |
| Static memory fraction | 0.80 |
| Repetitions | 1 warmup, 3 timed repetitions, 1 streaming probe per case |
| Hardware | NVIDIA RTX 5880 Ada (48 GB); server CPU work pinned to the job's cores |
| Arm placement | EAGLE3: both arms on one server and GPU, alternating per workload; DDTree: separate servers per arm, baseline first, same GPU |

## B Per-Drafter Results

Tables 4 and 6 give pooled MAT for both arms and the per-workload token-matched results summarized in Table 2, with superscripts counting the excluded cases; Tables 5 and 7 give absolute decode rates and paired intervals. Over all cases, without the token matching, EAGLE3's round reduction ranges over $15.8$–$34.9\%$ and its decode gain over $19.0$–$40.0\%$; both ends of the first range come from a single runaway case, one on each side: the top from an EAGLE3 case on Qwen3-8B/BFCL-multiturn that repeats a grammar-legal token to the cap in $245$ rounds, and the bottom from a SemanticTree case on Qwen3-4B/BFCL-multiple that runs to the $256$-token cap in $128$ rounds where EAGLE3 needs $12$.

Table 4: **SemanticTree against EAGLE3,** $B{=}1$**.** Pooled MAT for each arm, verification-round change and decode-throughput gain on token-matched cases; superscripts count the excluded cases.

| | | MAT | | | |
|---|---|---|---|---|---|
| Target | Workload | EAGLE3 | +SemanticTree | $\Delta R$ (%) | decode gain (%) |
| Qwen3-4B | BFCL-simple | 2.36 | 2.92 | −19.4 | +22.1[4] |
| | BFCL-multiple | 1.95 | 2.44 | −20.0 | +23.1[4] |
| | BFCL-multiturn | 1.93 | 2.52 | −23.7 | +28.3[7] |
| | Nexus | 2.38 | 2.97 | −20.0 | +32.9[3] |
| | ToolHop | 2.60 | 3.57 | −27.1 | +34.9[8] |
| Qwen3-8B | BFCL-simple | 2.38 | 3.33 | −28.5 | +38.2[1] |
| | BFCL-multiple | 2.38 | 3.38 | −29.8 | +40.3[3] |
| | BFCL-multiturn | 2.39 | 3.29 | −27.1 | +35.5[4] |
| | Nexus | 2.96 | 4.00 | −26.0 | +34.6 |
| | ToolHop | 2.45 | 3.41 | −28.1 | +37.2[4] |
| Llama-3.1-8B | BFCL-simple | 3.02 | 4.15 | −27.1 | +35.5[1] |
| | BFCL-multiple | 3.06 | 4.10 | −25.2 | +32.1[1] |
| | BFCL-multiturn | 2.43 | 3.19 | −23.9 | +29.9[1] |
| | Nexus | 2.05 | 2.46 | −16.4 | +19.5[3] |
| | ToolHop | 1.71 | 2.20 | −22.3 | +27.5[4] |
| Qwen3-14B | BFCL-simple | 3.72 | 4.52 | −17.5 | +20.0[1] |
| | BFCL-multiple | 3.53 | 4.46 | −20.7 | +25.2[4] |
| | BFCL-multiturn | 3.17 | 4.10 | −22.7 | +28.0[11] |
| | Nexus | 3.63 | 4.48 | −19.0 | +22.0[6] |
| | ToolHop | 3.31 | 4.06 | −18.4 | +21.7[9] |

Table 5: **SemanticTree against EAGLE3 with paired** $95\%$ **intervals,** $B{=}1$**.** Decode throughput of EAGLE3 alone and with SemanticTree (+ST), in tok/s, and the round change and decode gain on the token-matched cases counted in Table 4.

| | | decode tok/s | | | |
|---|---|---|---|---|---|
| Target | Workload | alone | +ST | $\Delta R$ (%) [95% CI] | decode gain (%) [95% CI] |
| Qwen3-4B | BFCL-simple | 134.8 | 164.6 | −19.4 [−20.5, −18.2] | +22.1 [+20.4, +23.8] |
| | BFCL-multiple | 109.5 | 134.7 | −20.0 [−21.1, −19.0] | +23.1 [+21.5, +24.7] |
| | BFCL-multiturn | 99.1 | 127.2 | −23.7 [−26.3, −20.5] | +28.3 [+23.5, +32.8] |
| | Nexus | 117.7 | 156.4 | −20.0 [−22.6, −17.7] | +32.9 [+28.5, +37.3] |
| | ToolHop | 142.0 | 191.5 | −27.1 [−29.0, −25.4] | +34.9 [+31.9, +38.2] |
| Qwen3-8B | BFCL-simple | 79.7 | 110.1 | −28.5 [−29.6, −27.3] | +38.2 [+36.0, +40.4] |
| | BFCL-multiple | 78.4 | 110.0 | −29.8 [−31.2, −28.4] | +40.3 [+37.7, +43.1] |
| | BFCL-multiturn | 75.4 | 102.1 | −27.1 [−28.6, −25.8] | +35.5 [+33.0, +38.1] |
| | Nexus | 94.7 | 127.4 | −26.0 [−27.7, −24.3] | +34.6 [+31.6, +37.7] |
| | ToolHop | 80.4 | 110.3 | −28.1 [−30.3, −25.9] | +37.2 [+33.2, +41.5] |
| Llama-3.1-8B | BFCL-simple | 101.9 | 138.0 | −27.1 [−28.5, −25.9] | +35.5 [+33.2, +38.0] |
| | BFCL-multiple | 101.6 | 134.3 | −25.2 [−26.7, −23.8] | +32.1 [+29.7, +34.6] |
| | BFCL-multiturn | 77.5 | 100.7 | −23.9 [−25.5, −22.4] | +29.9 [+27.4, +32.7] |
| | Nexus | 66.7 | 79.7 | −16.4 [−18.8, −14.1] | +19.5 [+16.5, +22.8] |
| | ToolHop | 56.7 | 72.3 | −22.3 [−24.4, −20.2] | +27.5 [+24.1, +30.9] |
| Qwen3-14B | BFCL-simple | 74.4 | 89.3 | −17.5 [−18.9, −16.2] | +20.0 [+18.1, +22.0] |
| | BFCL-multiple | 70.3 | 88.0 | −20.7 [−22.1, −19.4] | +25.2 [+23.1, +27.3] |
| | BFCL-multiturn | 60.6 | 77.6 | −22.7 [−24.4, −21.2] | +28.0 [+25.5, +30.7] |
| | Nexus | 71.2 | 86.9 | −19.0 [−21.2, −16.9] | +22.0 [+19.1, +25.3] |
| | ToolHop | 65.5 | 79.6 | −18.4 [−20.4, −16.4] | +21.7 [+18.8, +24.7] |

Table 6: **SemanticTree against DDTree, $B{=}1$.** Pooled MAT for each arm, verification-round change and decode-throughput gain on token-matched cases; superscripts count the excluded cases.

| | | MAT | | | |
|---|---|---|---|---|---|
| Target | Workload | DDTree | +SemanticTree | $\Delta R$ (%) | decode gain (%) |
| Qwen3-4B | BFCL-simple | 4.75 | 5.82 | −18.3 | +15.5[2] |
| | BFCL-multiple | 3.78 | 4.78 | −20.8 | +18.8[1] |
| | BFCL-multiturn | 2.73 | 3.69 | −25.8 | +25.1[2] |
| | Nexus | 3.78 | 4.74 | −20.2 | +17.8[3] |
| | ToolHop | 4.01 | 5.30 | −24.3 | +23.0[5] |
| Qwen3-8B | BFCL-simple | 4.19 | 5.37 | −21.9 | +22.8[2] |
| | BFCL-multiple | 4.06 | 5.22 | −22.2 | +22.8[1] |
| | BFCL-multiturn | 2.94 | 3.74 | −21.3 | +20.8[4] |
| | Nexus | 3.67 | 4.84 | −24.2 | +25.6[2] |
| | ToolHop | 3.61 | 4.64 | −22.2 | +22.3[1] |
| Llama-3.1-8B | BFCL-simple | 5.03 | 6.10 | −17.6 | +18.5 |
| | BFCL-multiple | 5.07 | 6.12 | −17.2 | +17.5 |
| | BFCL-multiturn | 3.36 | 4.35 | −22.9 | +25.1[2] |
| | Nexus | 3.24 | 4.16 | −22.3 | +24.3[3] |
| | ToolHop | 3.11 | 4.01 | −22.5 | +24.7 |

Table 7: **SemanticTree against DDTree with paired $95\%$ intervals, $B{=}1$.** Decode throughput of DDTree alone and with SemanticTree (+ST), in tok/s, and the round change and decode gain on the token-matched cases counted in Table 6.

| | | decode tok/s | | | |
|---|---|---|---|---|---|
| Target | Workload | alone | +ST | $\Delta R$ (%) [95% CI] | decode gain (%) [95% CI] |
| Qwen3-4B | BFCL-simple | 299.8 | 346.1 | −18.3 [−20.0, −16.6] | +15.5 [+13.1, +17.8] |
| | BFCL-multiple | 229.3 | 272.4 | −20.8 [−22.3, −19.3] | +18.8 [+16.7, +20.9] |
| | BFCL-multiturn | 153.2 | 191.7 | −25.8 [−27.4, −24.3] | +25.1 [+22.9, +27.5] |
| | Nexus | 221.0 | 260.4 | −20.2 [−22.5, −17.9] | +17.8 [+15.1, +20.8] |
| | ToolHop | 235.9 | 290.3 | −24.3 [−26.7, −22.0] | +23.0 [+19.7, +26.7] |
| Qwen3-8B | BFCL-simple | 158.9 | 195.0 | −21.9 [−23.3, −20.6] | +22.8 [+20.8, +24.8] |
| | BFCL-multiple | 151.1 | 185.5 | −22.2 [−23.6, −20.7] | +22.8 [+20.7, +25.0] |
| | BFCL-multiturn | 106.6 | 128.9 | −21.3 [−22.9, −19.6] | +20.8 [+18.6, +23.2] |
| | Nexus | 135.6 | 170.3 | −24.2 [−26.2, −22.2] | +25.6 [+22.4, +28.9] |
| | ToolHop | 134.4 | 164.3 | −22.2 [−24.4, −20.0] | +22.3 [+19.2, +25.7] |
| Llama-3.1-8B | BFCL-simple | 199.3 | 236.2 | −17.6 [−18.9, −16.4] | +18.5 [+16.9, +20.3] |
| | BFCL-multiple | 197.1 | 231.6 | −17.2 [−18.5, −15.9] | +17.5 [+15.7, +19.4] |
| | BFCL-multiturn | 127.4 | 159.4 | −22.9 [−24.7, −21.1] | +25.1 [+22.5, +27.8] |
| | Nexus | 125.8 | 156.4 | −22.3 [−24.6, −20.0] | +24.3 [+21.0, +27.8] |
| | ToolHop | 121.5 | 151.5 | −22.5 [−24.8, −20.5] | +24.7 [+21.7, +28.1] |

**Throughput relative to plain decoding.** Table 8 divides each method's decode throughput by that of target-only constrained decoding (AR); Table 9 gives, per setting, the number of cases each ratio uses and AR's own throughput. The AR runs come from our earlier campaign, with the same serving settings, protocol and seed cases (for Qwen3-8B, the same cases from an earlier replay of the same protocol); the other four methods come from the runs of Tables 4 and 6. For each setting we use the cases that all methods of that target decode to the same number of tokens (79–99.5% of the cases), so every ratio compares the same work. Because the AR runs were measured separately, these ratios carry more run-to-run variation than the paired comparisons above.

Table 8: **Decode throughput relative to plain decoding, per target, method and workload,** $B{=}1$: all methods on a shared token-matched cohort. Throughput of each method divided by that of plain decoding (AR $= 1$) on the same cases; mean: geometric mean over the five workloads. Bold: with SemanticTree. Case counts and AR throughput are in Table 9.

| | | BFCL | | | | | |
|---|---|---|---|---|---|---|---|
| Target | Method | simple | multiple | multiturn | Nexus | ToolHop | mean |
| Qwen3-4B | EAGLE3 | 1.34 | 1.11 | 0.96 | 1.20 | 1.47 | 1.21 |
| | +SemanticTree | **1.64** | **1.37** | **1.26** | **1.57** | **2.00** | **1.55** |
| | DDTree | 2.97 | 2.33 | 1.65 | 2.24 | 2.47 | 2.29 |
| | +SemanticTree | **3.43** | **2.76** | **2.07** | **2.64** | **3.05** | **2.75** |
| Qwen3-8B | EAGLE3 | 1.44 | 1.42 | 1.41 | 1.74 | 1.47 | 1.49 |
| | +SemanticTree | **1.98** | **1.99** | **1.91** | **2.35** | **2.03** | **2.05** |
| | DDTree | 2.87 | 2.75 | 2.01 | 2.51 | 2.43 | 2.50 |
| | +SemanticTree | **3.52** | **3.37** | **2.44** | **3.14** | **2.99** | **3.07** |
| Llama-3.1-8B | EAGLE3 | 1.80 | 1.81 | 1.41 | 1.21 | 1.00 | 1.41 |
| | +SemanticTree | **2.44** | **2.39** | **1.84** | **1.44** | **1.28** | **1.82** |
| | DDTree | 3.52 | 3.51 | 2.35 | 2.25 | 2.16 | 2.69 |
| | +SemanticTree | **4.18** | **4.12** | **2.94** | **2.79** | **2.70** | **3.28** |
| Qwen3-14B | EAGLE3 | 2.42 | 2.29 | 2.01 | 2.35 | 2.15 | 2.24 |
| | +SemanticTree | **2.90** | **2.86** | **2.57** | **2.87** | **2.61** | **2.76** |

Table 9: **Cases and plain-decoding throughput behind Table 8,** $B{=}1$**.** $n$: cases shared by all methods of a target with equal output length; AR tok/s: decode throughput of plain decoding on those cases.

| | Qwen3-4B | | Qwen3-8B | | Llama-3.1-8B | | Qwen3-14B | |
|---|---|---|---|---|---|---|---|---|
| Workload | $n$ | AR tok/s | $n$ | AR tok/s | $n$ | AR tok/s | $n$ | AR tok/s |
| BFCL-simple | 170 | 100.5 | 188 | 55.5 | 190 | 56.6 | 191 | 30.8 |
| BFCL-multiple | 179 | 99.1 | 188 | 55.2 | 191 | 56.2 | 185 | 30.7 |
| BFCL-multiturn | 175 | 93.3 | 183 | 53.6 | 176 | 54.9 | 176 | 30.2 |
| Nexus | 53 | 95.6 | 59 | 54.4 | 56 | 55.4 | 53 | 30.4 |
| ToolHop | 50 | 97.3 | 57 | 54.8 | 57 | 55.8 | 53 | 30.6 |

## C Grammar Cost per Round

Table 10 gives the per-round grammar cost behind Section 4.4 for the autoregressive drafter. The profile pass runs on a separate server with the telemetry enabled and is never mixed with the timed runs. Exposed time is measured with CUDA events: for both arms, the gap between the end of the target forward and the arrival of its verification bitmask; for SemanticTree, in addition, the time the draft graph spends waiting for masks. For the block drafter, the planner's host time is logged per round; the verification bitmask adds 0.013–0.018 ms per round in both arms.

Table 11 gives the cache ablation of Section 4.4 (Qwen3-4B, $B{=}1$). The cache-off server sets the cache capacity to zero, so every mask is filled; both SemanticTree servers commit the same trees and need the same rounds. Rounds, time per round and decode gain come from the timed pass (three

Table 10: **Grammar cost per EAGLE3 decode round,** $B{=}1$ (ms; range over the four targets). CPU: EAGLE3's native draft-tree traversal, replayed offline on EAGLE3 verification trees of the same targets and workloads (mean, 90th percentile in parentheses), and SemanticTree's host grammar time online, drafting plus verification assembly. Exposed: grammar time on the GPU critical path; about $0.05$ ms is the bitmask copy that both arms pay. Cache hit: share of SemanticTree's mask requests served by the state-hash cache.

| | EAGLE3 | | +SemanticTree | | |
|---|---|---|---|---|---|
| Workload | CPU | exposed | CPU | exposed | cache hit (%) |
| BFCL-simple | 0.11–0.20 (0.30–0.73) | 0.04–0.05 | 0.86–0.98 | 0.14–0.17 | 95.2–96.1 |
| BFCL-multiple | 0.14–0.24 (0.37–1.16) | 0.04–0.05 | 0.99–1.02 | 0.14–0.17 | 95.2–97.5 |
| BFCL-multiturn | 0.24–0.35 (0.73–1.45) | 0.04–0.05 | 1.03–1.08 | 0.14–0.17 | 98.1–99.0 |
| Nexus | 1.91–4.61 (4.45–20.44) | 0.05–2.50 | 1.09–1.19 | 0.14–0.27 | 98.4–99.1 |
| ToolHop | 0.32–0.43 (1.20–1.41) | 0.04–0.05 | 1.04–1.09 | 0.14–0.17 | 94.7–95.9 |

repetitions and one streaming probe per case) on the cases whose probe decodes the same number of tokens in all three arms (188 of 192 and 60 of 63 cases with EAGLE3, 191 of 192 and 60 of 63 with DDTree), with the definitions of Appendix A: time per round is the pooled accepted length divided by the decode throughput. Grammar times come from a separate profile pass over 16 cases per workload, in which the cache serves $97.6\%$ and $99.1\%$ of the mask requests with EAGLE3 and $83.1\%$ and $97.6\%$ with DDTree.

Table 11: **Parser-state cache on and off**, Qwen3-4B, $B{=}1$; +ST on/off is SemanticTree with the cache on or off. Rounds, ms per round and decode gain over the proposer alone [paired 95% CI] on token-matched cases; host grammar and exposed time per round from the profile pass. EAGLE3's own traversal is measured offline (Table 10). For DDTree, host grammar time is the planner's time, which lies on the critical path by construction; exposed time counts only the wait for the verification bitmask.

| | | | | | | grammar ms/round | |
|---|---|---|---|---|---|---|---|
| Drafter | Workload | Arm | rounds | ms/round | decode gain (%) | host | exposed |
| EAGLE3 | BFCL-multiple | alone | 2718 | 17.82 | – | – | 0.05 |
| | | +ST, on | 2174 | 18.17 | +22.6 [+21.0, +24.2] | 1.00 | 0.14 |
| | | +ST, off | 2174 | 18.16 | +22.6 [+21.0, +24.2] | 1.02 | 0.13 |
| | Nexus | alone | 1232 | 20.24 | – | – | 2.46 |
| | | +ST, on | 986 | 19.09 | +32.5 [+27.9, +36.8] | 1.11 | 0.14 |
| | | +ST, off | 986 | 23.54 | +7.4 [+1.6, +14.0] | 13.24 | 7.99 |
| DDTree | BFCL-multiple | alone | 1426 | 16.25 | – | 0.17 | 0.02 |
| | | +ST, on | 1129 | 17.44 | +17.7 [+15.6, +19.9] | 1.46 | 0.02 |
| | | +ST, off | 1129 | 17.52 | +17.1 [+15.0, +19.3] | 1.41 | 0.02 |
| | Nexus | alone | 794 | 16.99 | – | 0.17 | 0.02 |
| | | +ST, on | 634 | 18.02 | +18.1 [+15.2, +21.2] | 0.97 | 0.02 |
| | | +ST, off | 634 | 19.27 | +10.4 [+6.0, +15.0] | 3.32 | 0.02 |

## D Batched Serving at Steady State

**Protocol.** A batch measurement is meaningful only if the decode batch actually holds $B$ requests. The server runs with at most $B$ running requests, and the client keeps $2B$ requests outstanding, sending the next case as soon as one returns, so a request is always queued and the batch refills as soon as one finishes. For each workload the client runs a warmup and then four timed passes over all cases, in the order EAGLE3, SemanticTree, SemanticTree, EAGLE3. The server logs, for every decode round, its GPU start and end times, the requests in it with their accepted tokens, and every prefill forward. We keep the rounds whose batch holds exactly $B$ requests and charge each the time until the next decode round starts, minus any prefill in between: decode time that includes the GPU's wait for the host between rounds but not prefill. A round's time is split evenly among its

requests, so a case is credited with its own tokens and its share of time. A case enters the comparison when its output length is identical in all four passes; this keeps 77–99% of the cases, since numerical differences between batch compositions occasionally change a greedy choice. Intervals are paired case bootstraps as in Appendix A. Full-batch rounds carry 66–88% of the decoded tokens at $B{=}4$ and 36–67% at $B{=}8$; the rest comes from the end of each pass, when the last long outputs finish alone.

**Results.** Tables 12 and 13 report the round change and the steady-state decode gain. Rounds fall by 15.3–29.6% at $B{=}4$ and 15.5–29.7% at $B{=}8$, as at batch size one. A full-batch round costs about the same in both arms ($-2.9$ to $+2.9\%$), so decode throughput rises by 19.1–41.6% at $B{=}4$ and 17.1–41.7% at $B{=}8$, in all 40 settings, with the lowest interval bound at $+11.4\%$. The gain computed from GPU time alone differs by less than 0.1 percentage points, so the host adds no measurable gap between rounds, and the two halves of the order-balanced passes agree in sign in every setting. The grammar time SemanticTree leaves on the critical path grows with the batch (Table 14): 0.37–0.47 ms per round at $B{=}4$ and 0.71–1.94 ms at $B{=}8$, above EAGLE3's 0.15–0.35 ms outside Nexus, against rounds of 35–216 ms.

Table 12: **Batched serving at steady state, $B{=}4$.** SemanticTree against EAGLE3 at the same budget with four concurrent requests: number of matched cases, verification-round change and steady-state decode gain with its paired 95% interval.

| Target | Workload | $n$ | $\Delta R$ (%) | decode gain (%) [95% CI] |
|---|---|---|---|---|
| Qwen3-4B | BFCL-simple | 178 | −19.5 | +23.3 [+20.9, +25.9] |
| | BFCL-multiple | 179 | −19.8 | +24.9 [+22.8, +27.2] |
| | BFCL-multiturn | 171 | −25.3 | +38.0 [+34.6, +41.7] |
| | Nexus | 56 | −19.2 | +28.4 [+25.1, +31.9] |
| | ToolHop | 53 | −27.8 | +37.5 [+34.0, +41.4] |
| Qwen3-8B | BFCL-simple | 190 | −28.7 | +36.9 [+34.3, +39.6] |
| | BFCL-multiple | 188 | −29.6 | +39.6 [+36.3, +43.1] |
| | BFCL-multiturn | 180 | −28.3 | +41.6 [+36.9, +46.4] |
| | Nexus | 57 | −26.0 | +39.7 [+34.2, +45.7] |
| | ToolHop | 57 | −28.2 | +38.3 [+33.3, +43.5] |
| Llama-3.1-8B | BFCL-simple | 190 | −27.1 | +34.6 [+31.5, +37.9] |
| | BFCL-multiple | 188 | −25.3 | +32.9 [+29.9, +36.2] |
| | BFCL-multiturn | 174 | −23.7 | +31.5 [+27.5, +35.7] |
| | Nexus | 57 | −15.3 | +19.1 [+15.8, +22.5] |
| | ToolHop | 55 | −22.5 | +32.0 [+27.9, +36.2] |
| Qwen3-14B | BFCL-simple | 189 | −17.8 | +21.6 [+18.7, +24.7] |
| | BFCL-multiple | 186 | −20.8 | +25.3 [+22.4, +28.3] |
| | BFCL-multiturn | 169 | −22.5 | +28.3 [+24.6, +32.3] |
| | Nexus | 52 | −19.7 | +25.4 [+21.3, +29.8] |
| | ToolHop | 49 | −16.8 | +20.0 [+16.1, +24.5] |

Table 13: **Batched serving at steady state,** $B{=}8$**.** As Table 12, with eight concurrent requests.

| Target | Workload | $n$ | $\Delta R$ (%) | decode gain (%) [95% CI] |
|---|---|---|---|---|
| Qwen3-4B | BFCL-simple | 177 | −19.6 | +23.3 [+20.0, +26.7] |
| | BFCL-multiple | 180 | −19.8 | +27.4 [+24.6, +30.3] |
| | BFCL-multiturn | 169 | −25.3 | +37.8 [+33.7, +42.4] |
| | Nexus | 56 | −20.2 | +27.4 [+22.2, +33.5] |
| | ToolHop | 52 | −28.2 | +36.7 [+31.2, +42.1] |
| Qwen3-8B | BFCL-simple | 187 | −28.6 | +36.9 [+33.3, +40.7] |
| | BFCL-multiple | 187 | −29.7 | +40.9 [+37.2, +44.8] |
| | BFCL-multiturn | 175 | −27.1 | +35.8 [+30.3, +41.6] |
| | Nexus | 56 | −26.2 | +31.0 [+23.1, +39.4] |
| | ToolHop | 56 | −29.2 | +41.7 [+35.4, +48.5] |
| Llama-3.1-8B | BFCL-simple | 190 | −26.9 | +37.4 [+32.8, +42.2] |
| | BFCL-multiple | 187 | −25.3 | +29.0 [+24.7, +33.5] |
| | BFCL-multiturn | 172 | −23.6 | +34.3 [+29.2, +39.6] |
| | Nexus | 61 | −15.5 | +17.1 [+11.4, +23.2] |
| | ToolHop | 53 | −22.3 | +32.4 [+28.5, +36.3] |
| Qwen3-14B | BFCL-simple | 190 | −17.7 | +23.2 [+19.6, +27.0] |
| | BFCL-multiple | 182 | −20.9 | +24.4 [+20.3, +28.6] |
| | BFCL-multiturn | 169 | −22.5 | +33.8 [+27.0, +40.7] |
| | Nexus | 52 | −20.0 | +23.8 [+17.5, +30.9] |
| | ToolHop | 50 | −18.2 | +21.5 [+16.1, +27.4] |

Table 14: **Grammar time left on the critical path per full-batch round** (ms; range over the four targets), measured as in Appendix C.

| | $B{=}4$ | | $B{=}8$ | |
|---|---|---|---|---|
| Workload | EAGLE3 | +SemanticTree | EAGLE3 | +SemanticTree |
| BFCL-simple | 0.15–0.18 | 0.37–0.47 | 0.29–0.35 | 0.82–1.94 |
| BFCL-multiple | 0.15–0.18 | 0.37–0.47 | 0.29–0.35 | 0.75–1.83 |
| BFCL-multiturn | 0.15–0.18 | 0.37–0.47 | 0.29–0.35 | 0.79–1.36 |
| Nexus | 0.18–3.17 | 0.39–0.47 | 0.30–7.17 | 0.71–1.92 |
| ToolHop | 0.15–0.18 | 0.39–0.47 | 0.29–0.35 | 0.72–1.78 |

## E THE PARSER-STATE HASH

The hash covers what determines XGrammar's next-token bitmask: the latest scannable Earley states and, recursively by content, every parent state they can complete into, found through each state's rule start position without hashing the position itself, together with the completion and stop-token flags. It returns no value when the reachable structure is cyclic. We validated it offline on Qwen3-4B's tokenizer over 12 cases per workload: at every position of each reference output, and along three random branches of up to four tokens from each position that are biased toward structural tokens so that they leave strings and diverge in history, we recorded the hash and the bitmask of every visited state. Over 16,884 states, no two states with equal hashes had different bitmasks and none was cyclic; the median hash takes 5.9 $\mu$s. On the same states, a cache keyed by the hash would skip 98.9% of the mask-filling time on Nexus and 84–94% on the other workloads. Online, we ran the evaluated implementation in a checking mode that recomputes the mask of every cache hit with the matcher and compares it bit for bit with the cached one, on 16 cases per workload: with EAGLE3 on the four targets at batch size one and at four concurrent requests (640 requests), and with DDTree on its three targets at batch size one (240 requests). Over 547,967 checked hits with EAGLE3 and 17,890 with DDTree, no cached mask differed from the recomputed one.

## F Implementation and Profile Details

**Captured frontier exchange.** At depth $i$, EAGLE3 publishes frontier $F_i$, with a token, validity bit and parent index for each of its $B \cdot K$ rows, where $B$ is the request batch size. A flag carries the round number. The host reads this frontier, forks each node's matcher from its parent's, accepts the node's token, and writes masks to pinned memory. A device kernel waits for the mask flag, a captured copy transfers the masks to the GPU, and masked Top-$K$ selects $F_{i+1}$. The wait kernel has a timeout and sticky error flag. Unlike captured host callbacks for chain-based guided speculation (NVIDIA TensorRT LLM Team and XGrammar Team, 2026), no graph node calls back into the host.

**Reuse and remaining device work.** In the EAGLE3 profile, $46$–$66\%$ of non-root verification rows reuse masks from expanded nodes; $14$–$23$ unexpanded leaf rows per round need mask lookup or computation, almost all cache hits. About $0.05$ ms of exposed grammar time is the verification-bitmask copy paid by both arms. Handshake kernels, mask copies, masked Top-$K$ and the root wait still cost device time. In one profiled Qwen3-4B/BFCL-simple case, the median draft phase increases from $4.44$ to $5.04$ ms, while target verification takes $13.70$ and $13.79$ ms. These single-case phase medians illustrate where the added work runs; the throughput comparisons use the full paired cohorts in Appendix B.