A SAMPLING-CENTRIC FRAMEWORK FOR TREE SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Tree speculative decoding accelerates large language model inference by drafting and verifying multiple candidate continuations in parallel. Top-candidate dynamic-tree methods such as EAGLE-2/3 follow a *topology-first* paradigm: they expand deterministic candidates and globally prune them by score, so draft–target overlap outside the retained set cannot contribute to acceptance. Introducing sampled candidates into this pipeline creates a selection problem: under global pruning, a sample's realized identity can affect its own survival. Existing sampling extensions such as RheoSampling retain EAGLE-style *expand-then-prune* and therefore must explicitly control how samples are scored and pruned to preserve losslessness. We introduce Solvo, a *sampling-centric* framework that replaces expand-then-prune with *allocate-then-sample*. At each layer, a global path-score threshold commits slots to parents before any token is drawn, and slots are never reranked afterward, so a sampled token can condition subsequent expansion but not its own retention. Under stochastic decoding, each non-root candidate group places top-ranked deterministic tokens in every slot except the last, which is filled with a sample from the remaining draft mass. Using a martingale argument over tree growth, we prove that Solvo preserves the exact target-model distribution under adaptive growth and cost-aware stopping. Across five target models and six tasks, Solvo improves end-to-end throughput by 7.3–10.2% over EAGLE-3 under stochastic decoding, and still by 2.9–7.8% under greedy decoding, where sampling plays no role.

1 INTRODUCTION

Large Language Models (LLMs) generate text autoregressively, requiring a target-model forward pass for each new token and making inference inherently sequential and slow. Speculative decoding mitigates this bottleneck through a draft-and-verify paradigm: a lightweight draft model proposes multiple future tokens, which are then verified in parallel by the target model while preserving the exact output distribution of standard autoregressive decoding (Leviathan et al., 2023; Chen et al., 2023). Tree speculative decoding further organizes candidate continuations into a draft tree, allowing a single target-model forward pass to verify multiple possible paths and thereby increasing the draft hit rate. The EAGLE series is a representative family of dynamic-tree methods that adapt the draft-tree structure to the current context (Li et al., 2024b; 2025).

However, EAGLE-style construction uses cumulative draft probability as a surrogate for a node's potential contribution to acceptance: under a limited tree budget, higher-scoring nodes are given priority. It consequently determines the tree topology through repeated Top-K frontier selection and global score-based reranking, retaining the highest-ranked deterministic candidates from the drafted pool. Draft probability mass outside this retained set cannot contribute to candidate acceptance, even when it overlaps substantially with the target distribution. In principle, this overlap can still be recovered: if the proposal set includes sampled candidates, advanced verification algorithms can exploit it and flexibly reallocate probability mass across candidates (Sun et al., 2023; 2024; Hu et al., 2025; Weng et al., 2026). A top-candidate dynamic tree thus does more than limit candidate coverage: it restricts the proposal family available to the verifier before verification even begins.

A natural remedy is to inject sampled candidates into this pipeline, but doing so naively creates a selection problem. If a sampled token is ranked by its realized draft probability, its identity can deter-

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.

mine whether it survives pruning. Conditioning on survival can then distort its proposal distribution, invalidating verification based on the original one. RheoSampling controls this sampling–pruning coupling through identity-independent proxy scores and constrained pruning (Hu et al., 2026), but retains *expand-then-prune* and requires a construction-specific dependency analysis to establish losslessness. We argue that sampling should shape tree construction: rather than retrofitting stochastic sampling into a deterministic expansion-and-pruning pipeline, tree construction should be designed around the sampling structure required by advanced verification from the outset.

We introduce **Solvo**, a *sampling-centric framework for tree speculative decoding* that replaces *expand-then-prune* with an *allocate-then-sample* procedure. At each layer, a global path-score threshold first commits structural slots to parent groups; only then are tokens materialized into these slots. Under stochastic decoding, each non-root group places top-ranked deterministic tokens in every slot except the last, which is filled with a sample from the remaining draft mass. Because these slots are fixed before sampling and never globally reranked afterward, a sampled token may condition subsequent expansion without affecting its own retention. Sampling thus becomes a native part of tree growth rather than an add-on to deterministic selection: the tree adapts where draft computation is spent, while sampled candidates let compatible verifiers exploit draft–target overlap beyond deterministic candidates.

We evaluate Solvo on five target models and six benchmarks spanning instruction following, dialogue, mathematical reasoning, code generation, question answering, and summarization. Under stochastic decoding ($T = 1$), the setting targeted by our design, Solvo improves aggregate end-to-end throughput by **7.3–10.2%** over EAGLE-3 and by **3.2–5.4%** over the strongest of the more recent baselines, TALON and RheoSampling (Liu et al., 2026; Hu et al., 2026). It also improves throughput by 2.9–7.8% under greedy decoding ($T = 0$), where sampling plays no role.

Our main contributions are summarized as follows:

- **A sampling-native tree construction.** We propose Solvo, which designs tree construction around stochastic proposals from the outset. Its allocate-then-sample procedure commits slots before sampling, so sampled tokens can guide subsequent growth without pruning-induced selection bias.
- **Losslessness under adaptive growth and stopping.** Using a martingale argument over tree growth, we prove that Solvo preserves the exact target-model distribution even though its topology adapts to sampled tokens and its depth is determined by cost-aware stopping.
- **Consistent end-to-end acceleration.** Solvo consistently improves throughput across all evaluated targets under stochastic decoding. Its cost-aware depth control additionally improves greedy decoding, where sampled candidates play no role.

2 PRELIMINARIES

2.1 CONTEXT-AWARE DYNAMIC DRAFT TREES

The EAGLE-style dynamic-tree construction (Li et al., 2024b; 2025) uses draft probability as a proxy for token acceptance probability. It assigns each drafted node a path score equal to the product of draft probabilities along its root-to-node path, approximating the probability that the entire path is accepted. At each expansion step, it selects the Top-K nodes with the largest path scores in the frontier layer and expands them with the draft model. After reaching a fixed depth, it reranks all drafted nodes by their path scores and retains the top-B nodes, which form a connected tree. OPT-Tree (Wang et al., 2025) shares essentially the same probability-coverage idea as EAGLE, but makes the optimization objective more explicit and adopts a different tree-construction procedure. Unless otherwise specified, we use EAGLE to refer to the dynamic-tree construction of EAGLE-2/3.

TALON (Liu et al., 2026) controls tree shape differently. After a Top-K first layer, it retains at each subsequent layer the candidates whose path scores exceed a fraction μ of that layer's maximum, and keeps expanding layer by layer until the node budget is exhausted. Rather than running a preset number of expansion layers, TALON thus lets budget exhaustion determine when construction ends: in uncertain contexts many candidates pass the relative threshold and the budget is spent on shallow, broad layers, while in confident contexts few pass and the tree grows deeper.

2.2 STOCHASTIC PROPOSALS AND DISTRIBUTIONAL OVERLAP

Both EAGLE and TALON adapt the tree topology while retaining deterministic token candidates. To isolate what stochastic proposals add, let $\mathcal{V}$ denote the vocabulary, and let $p(\cdot \mid u)$ and $q(\cdot \mid u)$ denote the target and draft next-token distributions at prefix u, respectively. For a single proposal drawn from q, standard speculative rejection sampling has marginal acceptance probability

$$\sum_{x \in \mathcal{V}} \min\{p(x \mid u), q(x \mid u)\}. \tag{1}$$

A deterministic top-1 proposal $x^\star = \arg\max_x q(x \mid u)$ is instead accepted with probability only $p(x^\star \mid u)$, whereas equation 1 aggregates draft–target overlap over the proposal's support. Rejection sampling can therefore achieve higher acceptance when substantial overlap lies outside the highest-ranked draft token. Sequoia (Chen et al., 2024) reports empirical acceptance gains from recursive rejection sampling without replacement over a deterministic top-K proposal baseline. In addition, RheoSampling (Hu et al., 2026) reports a controlled one-layer experiment using EAGLE-3, where verification using the three highest-ranked draft tokens achieves an acceptance rate of 80.0%. Replacing this candidate set with the highest-ranked token, one residual sample, and the highest-ranked remaining token raises the rate to 83.5% under recursive rejection sampling without replacement, and further with stronger verification rules.

Realizing this gain in existing dynamic trees is nontrivial, because under global pruning a sampled token's realized identity can determine its own survival. RheoSampling (Hu et al., 2026) addresses this problem by decoupling the structural score used for tree construction from the token's true sampling probability. It inserts one sampled token into each candidate group during EAGLE's expansion, forming a group of leading high-probability tokens, the sample, and fill tokens that follow it. During subsequent expansion and global pruning, the sample carries a prespecified, identity-independent proxy score, constrained to rank it ahead of every fill token in its group; verification uses its true sampling probability. The sampled token still affects the remainder of the tree, creating a complex coupling between its realized identity and subsequent construction; RheoSampling handles this coupling through an equivalence-class analysis.

The difficulty comes from the order of decisions in EAGLE-style construction, which is *topology-first*: tokens are first expanded and scored, and global pruning then decides which of them are retained. This order works for deterministic candidates, but RheoSampling keeps it, so pruning takes place *after* sampling. A sample's realized identity then affects its fill siblings, whose identities depend on it, and its descendants, whose path scores depend on it. Without explicit control, global pruning would select among these nodes by their realized scores and distort the proposal distribution assumed by the verifier; this is what RheoSampling's proxy constraint and equivalence-class analysis guard against. The difficulty therefore lies in pruning after sampling, not in sampling itself. If retention is decided *before* any token is sampled, this coupling does not arise and no such control is needed; we call such a construction *sampling-centric*.

3 METHOD

Solvo decides retention before sampling through *allocate-then-sample*: structural scores first allocate retained child slots, and tokens are then materialized into these slots. The scores that determine the allocation therefore need not be the probabilities of the tokens placed in those slots, and sampled tokens can guide subsequent growth without ever being subjected to post-sampling selection. Figure 1 contrasts this construction with EAGLE's token-coupled expansion and pruning.

3.1 SLOT-FIRST MIXED PROPOSAL CONSTRUCTION

Structural slot allocation. Let $\bar{q}$ denote the draft distribution used for structural scoring, without temperature scaling, and q the temperature-scaled proposal used for sampling and verification (Appendix D). Let $\Lambda(u)$ be the structural path score of node u, with $\Lambda(r) = 1$ at the root, and let $v_{u,j}$ denote the jth-ranked token under $\bar{q}(\cdot \mid u)$. For each parent u in the current frontier, candidate slot j has score $\Lambda(u)\bar{q}(v_{u,j} \mid u)$. Solvo compares these scores across all frontier parents and retains the slots scoring at least a global threshold θ, truncating them by score if they exceed the remaining

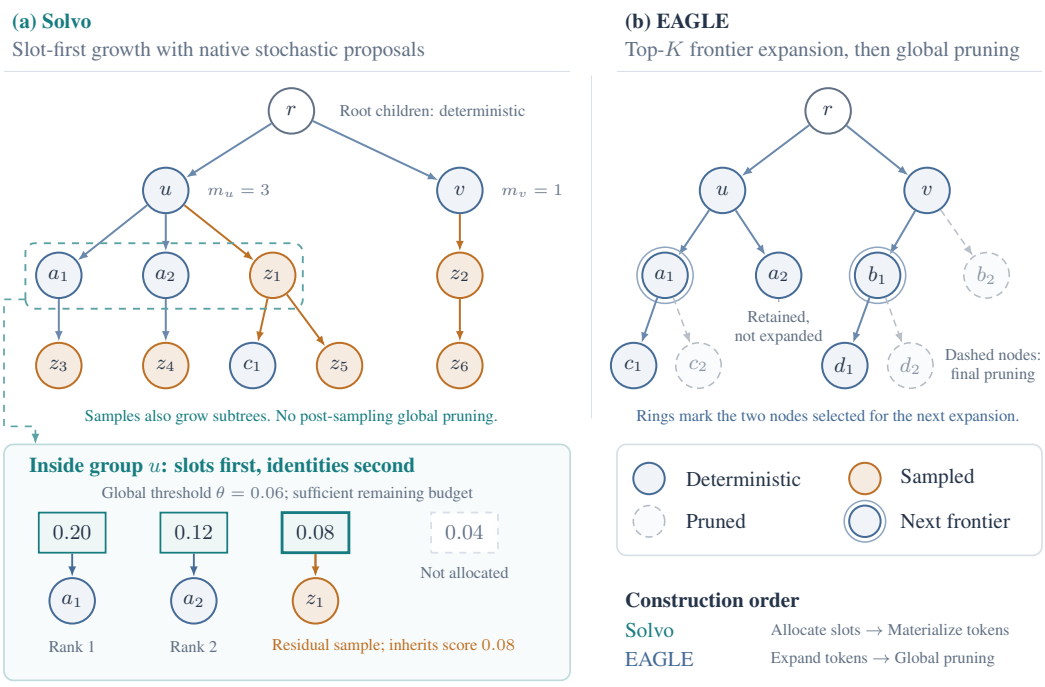

Figure 1: **Dynamic trees with deterministic candidates or native stochastic proposals.** (a) Solvo allocates slots to each group before any token is materialized. Each non-root group contains leading deterministic tokens and one residual sample (inset); for width one, the sample is drawn directly from q. Sampled nodes inherit their slots' structural scores independently of their identities, so they condition later subtrees without affecting their own retention. (b) EAGLE expands a Top-K frontier ($K = 2$ here) and globally prunes after a preset depth, so retention is decided only after tokens are expanded. Tree shapes and node counts are illustrative, not a matched-budget comparison.

node budget. The number of retained slots m_u for each parent is thus fixed before any token of the current layer is sampled.

Mixed proposal groups. For a non-root group with m_u allocated slots, the first $m_u - 1$ slots receive the deterministic tokens $v_{u,1}, \ldots, v_{u,m_u-1}$, and the boundary slot receives a sample z_u drawn from q after masking these tokens and renormalizing. Every token inherits its slot's structural score; in particular,

$$\Lambda(uz_u) = \Lambda(u)\bar{q}(v_{u,m_u} \mid u). \tag{2}$$

The sampled node is thus not rescored by $\bar{q}(z_u \mid u)$, although its identity conditions subsequent draft predictions. Root groups are filled entirely with top-ranked deterministic tokens, a non-root group with one slot samples directly from q, and at $T = 0$ all groups are filled deterministically.

Adaptive depth and budget refill. Every materialized node is retained and can be expanded in the next draft layer. Let L_b and L_d be the fitted base-cycle and marginal draft-layer costs (Appendix D.3). When the draft tree reaches depth d, its cycle cost is estimated as $D_d = L_b + dL_d$. Starting from $\widehat{E}_0 = 1$, we accumulate the acceptance-length proxy $\widehat{E}_d = \widehat{E}_{d-1} + \sum_{v \in F_d} \Lambda(v)$, where F_d is the set of retained nodes at depth d. We reuse this increment, $G_{d+1} = \sum_{v \in F_d} \Lambda(v)$, as an optimistic estimate of the next layer's gain, since the retained children of a node never exceed its score in total. Growth continues only if another layer is expected to increase the estimated acceptance length per unit cost:

$$\frac{\widehat{E}_d + G_{d+1}}{D_d + L_d} > \frac{\widehat{E}_d}{D_d} \quad \Longleftrightarrow \quad G_{d+1}D_d > \widehat{E}_d L_d. \tag{3}$$

Growth stops when the frontier is empty, the node budget is exhausted, or this test fails. Because this test ends growth once another forward no longer pays off, Solvo can afford a much larger depth

limit than EAGLE's fixed eight layers; we cap the draft depth at 20. Any remaining budget is then filled with the highest-probability unused root children, appended as leaves after the original root children so that the verification order and existing children are unchanged.

Verification. A single target-model forward pass evaluates all materialized nodes. Verification uses the actual proposal distributions of the materialized tokens, including the residual distribution of each boundary sample, rather than their structural scores. Any verifier with causally determined, locally lossless allocations preserves the target distribution (Section 4); we use UniVer by default (Weng et al., 2026), which supports groups that mix deterministic and sampled candidates.

3.2 Discussion

Proposal design. The Top-$(m_u - 1)$-plus-residual proposal combines deterministic coverage with access to residual draft–target overlap (Hu et al., 2025; Weng et al., 2026). It also aligns with slot allocation: each deterministic slot holds exactly the token whose score allocated it, and only the boundary slot explores the remaining mass. The same slots can also host other sampling schemes, each paired with a verifier that is lossless for it.

Why an absolute threshold. If path scores serve as the surrogate for acceptance, the best tree of size B consists of the B highest-scoring nodes, i.e., the result of global top-B pruning. Expand-then-prune approximates this target and recovers it exactly when the frontier and candidate pool are large enough. Sampling, however, requires that a candidate's realized identity not affect its own retention, so retention must be settled before tokens are materialized; under the same surrogate, the direct way to achieve this is to reverse the order and allocate slots before sampling. A threshold makes this reversal exact in principle. Because path scores never increase along a path, growing the tree layer by layer and retaining every slot scoring at least θ reaches exactly the slots above θ, and with θ set to the Bth largest score this is precisely the global top-B set. That score is only known once the full pool exists, so Solvo fixes θ in advance as an approximation: if θ admits too much, the budget is exhausted early; if it admits too little, the unspent budget is refilled at the root. We thus keep global top-B as the target and compromise only on how it is reached. This also distinguishes our threshold from TALON's relative, per-layer threshold (Liu et al., 2026), which treats adaptive tree shape as the objective rather than as a means to a fixed global criterion. Appendix A makes both points precise.

4 Theoretical Guarantees

4.1 Notations

Verification trees. Fix the input prefix, draft and target models, and all deterministic construction parameters (node budget, path-score threshold, and so on). Let ω denote a realization of all drafting randomness (e.g., a random seed). The tree retained for target-model verification is then completely determined by ω. We call it a *verification tree* and denote it by $T(\omega)$. When the dependence on ω is clear from context, we simply write T.

Tree extension. For two verification trees T and T', we say that T' *extends* T if the node set $V(T)$ is an ancestor-closed subset of $V(T')$ and

$$T = T'|_{V(T)},$$

where $T'|_{V(T)}$ denotes the restriction of the rooted, ordered, token-labeled tree T' to the node set $V(T)$. We then write $T \preceq T'$.

Equivalently, T' may add new descendants to T, but cannot alter the existing tree structure, sibling order, or token labels.

Definition 4.1 (Stochastic tree growth)**.** Let $T_1(\omega), T_2(\omega), \ldots, T_n(\omega)$ be verification-tree-valued random variables generated from the same underlying drafting randomness ω. We call $\{T_k(\omega)\}_{k=1}^{n}$ a *stochastic tree growth* if, for every realization ω,

$$T_1(\omega) \preceq T_2(\omega) \preceq \cdots \preceq T_n(\omega).$$

The index k denotes the growth stage, and each stage may add finitely many nodes.

Target-completed output M_k. Following the standard target-completion argument in block verification (Sun et al., 2025, Lemma 2), after a verification cycle terminates we continue generation with the target model. For any finite token prefix x, define

$$M_k(x) := \Pr(\text{the target-completed output on } T_k \text{ starts with } x \mid \sigma(T_k)) .$$

4.2 Losslessness of Solvo

Solvo as a stochastic tree growth. Solvo verification uses UniVer's causal Allocation–Decision rule with deterministic root allocation and leaf-only root refill, as specified in Appendix B.1. For Solvo, let T_1 be the root-only verification tree. Each subsequent growth stage materializes one child group at a retained parent according to the construction rules in Section 3. If the kth child group $C(u_k)$ contains m_k children, we set

$$T_{k+1} = T_k \cup C(u_k), \qquad |V(T_{k+1})| = |V(T_k)| + m_k.$$

The group may be the initial deterministic root group, a mixed group, or the final root-refill group. Since Solvo never removes or modifies already retained nodes after a group is materialized,

$$T_k \preceq T_{k+1}, \quad \forall k \in \mathbb{N}^+.$$

Hence every Solvo construction induces a stochastic tree growth in the sense of Definition 4.1. Under Solvo's fixed group serialization and deterministic construction rules, the current tree also determines its preceding construction history; Appendix B.1 gives the formal justification.

Lemma 4.2 (One-step preservation)**.** *For every Solvo tree-growth step $T_k \to T_{k+1}$,*

$$\mathbb{E}_{\textit{Solvo}}[M_{k+1}(x) \mid \sigma(T_k)] = M_k(x).$$

By Lemma 4.2, $\{M_k(x)\}$ is a bounded martingale with respect to $\{\sigma(T_k)\}$, and each Solvo growth step is unbiased with respect to the target-completed output law. The proof uses the same local mass-preservation idea as UniVer (Weng et al., 2026), and details are deferred to Appendix B.

Theorem 4.3 (Losslessness of Solvo)**.** *Let N be the bounded terminal stopping stage of Solvo's stochastic tree growth, including the final root refill. Under the construction and verification conditions above, $\{M_k(x)\}$ is a bounded martingale with respect to $\{\sigma(T_k)\}$. Consequently, for every stopping time $\tau \le N$,*

$$\mathbb{E}_{\textit{Solvo}}[M_\tau(x)] = M_1(x) = P_{\text{target}}(x).$$

In particular, $\tau = N$ corresponds to a complete Solvo verification cycle, so Solvo preserves the exact target autoregressive distribution under stochastic decoding.

4.3 Connection to RheoSampling

For RheoSampling, let $\mathcal{U}$ be the underlying drafted tree and $T_k^{\mathrm{R}} := \mathrm{Prune}_k(\mathcal{U})$ the retained tree at pruning budget k; the sequence $\{T_k^{\mathrm{R}}\}_k$ plays the role of its stochastic growth. The equivalence class

$$[\mathcal{U}]_k := \{\mathcal{U}' : \mathrm{Prune}_k(\mathcal{U}') = T_k^{\mathrm{R}}\}$$

contains all drafting realizations yielding the same current tree, so probability-weighted averaging over this class is conditioning on T_k^{R}. Writing M_k^{R} for the corresponding target-completed output, the key induction of RheoSampling (Hu et al., 2026, Eq. (8)) also has the form

$$\mathbb{E}_{\text{Rheo}}[M_{k+1}^{\mathrm{R}}(x) \mid \sigma(T_k^{\mathrm{R}})] = M_k^{\mathrm{R}}(x).$$

Thus RheoSampling and Solvo share the same conditional one-step preservation structure.

5 Experiments

5.1 Experimental setup

Models and tasks. We evaluate five target models: Llama-3.1-8B-Instruct (Grattafiori et al., 2024), DeepSeek-R1-Distill-Llama-8B (DSL-8B) (Guo et al., 2025), Vicuna-13B-v1.3 (Zheng et al., 2023), Qwen3-8B, and Qwen3-32B (Yang et al., 2025). Each target is paired with the EAGLE-3 draft checkpoint trained for that model. We use 80 prompts from each of six benchmarks: Alpaca (Taori et al., 2023) instruction following, GSM8K mathematical reasoning (Cobbe et al., 2021), HumanEval code generation (Chen et al., 2021), MT-Bench dialogue (Zheng et al., 2023), Natural Questions open-domain question answering (Kwiatkowski et al., 2019), and CNN/DailyMail summarization (Nallapati et al., 2016). The maximum-new-token parameter is set to 512.

Table 1: Results on six benchmarks. τ is macro acceptance length and SR is speedup over the matched autoregressive reference. Entries are means over six runs; bold marks the best SR at the displayed precision. Six-task aggregates and their uncertainty are reported in Appendix C.

(a) Stochastic decoding ($T = 1$)

Model	Method	Alpaca		GSM8K		HumanEval		MT-Bench		QA		Summ.	
		τ	SR	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR
Llama3.1-8B	EAGLE-3	5.60	4.32×	5.46	4.29×	6.09	4.85×	4.93	3.89×	3.90	3.05×	4.47	3.50×
	TALON	5.54	4.36×	5.48	4.36×	6.52	5.08×	4.96	4.00×	3.82	3.14×	4.39	3.56×
	RheoSampling	5.96	4.59×	5.58	4.39×	6.21	4.95×	5.20	4.11×	4.19	3.25×	4.68	3.66×
	Solvo	6.04	**4.69**×	5.83	**4.59**×	6.82	**5.26**×	5.36	**4.27**×	4.19	**3.40**×	4.77	**3.79**×
DSL-8B	EAGLE-3	4.48	3.36×	6.42	4.77×	5.45	4.03×	4.91	3.63×	4.03	2.99×	4.33	3.19×
	TALON	4.39	3.46×	6.82	4.99×	5.36	4.09×	5.01	3.80×	3.99	3.13×	4.18	3.24×
	RheoSampling	4.76	3.59×	6.56	4.90×	5.75	4.28×	5.14	3.83×	4.34	3.24×	4.50	3.34×
	Solvo	4.87	**3.77**×	7.23	**5.22**×	5.87	**4.40**×	5.38	**4.03**×	4.39	**3.39**×	4.50	**3.41**×
Vicuna-13B	EAGLE-3	5.54	3.16×	5.86	3.50×	6.42	3.82×	5.71	3.36×	4.93	2.91×	5.91	3.26×
	TALON	5.52	3.23×	6.07	3.61×	7.45	4.25×	5.91	3.48×	4.87	2.95×	6.19	3.35×
	RheoSampling	5.87	3.35×	5.95	3.53×	6.60	3.92×	5.81	3.41×	5.04	2.96×	5.91	3.25×
	Solvo	5.96	**3.48**×	6.35	**3.74**×	7.58	**4.33**×	6.26	**3.66**×	5.20	**3.12**×	6.48	**3.47**×
Qwen3-8B	EAGLE-3	3.37	2.52×	3.82	2.85×	3.81	2.85×	3.45	2.59×	3.20	2.40×	3.17	2.37×
	TALON	3.32	2.65×	3.76	2.96×	3.74	2.97×	3.40	2.71×	3.16	2.52×	3.11	2.47×
	RheoSampling	3.30	2.48×	3.72	2.79×	3.68	2.76×	3.39	2.55×	3.20	2.40×	3.11	2.32×
	Solvo	3.41	**2.74**×	3.83	**3.04**×	3.79	**3.02**×	3.49	**2.79**×	3.31	**2.66**×	3.19	**2.54**×
Qwen3-32B	EAGLE-3	2.64	1.89×	3.25	2.31×	2.96	2.11×	2.79	1.99×	2.47	1.77×	2.46	1.76×
	TALON	2.58	1.92×	3.18	2.34×	2.89	2.14×	2.75	2.04×	2.41	1.80×	2.41	1.81×
	RheoSampling	2.64	1.89×	3.17	2.26×	2.85	2.03×	2.79	1.99×	2.50	1.79×	2.45	1.76×
	Solvo	2.74	**2.06**×	3.27	**2.43**×	2.94	**2.19**×	2.88	**2.15**×	2.60	**1.96**×	2.53	**1.91**×

(b) Greedy decoding ($T = 0$)

Model	Method	Alpaca		GSM8K		HumanEval		MT-Bench		QA		Summ.	
		τ	SR	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR
Llama3.1-8B	EAGLE-3	6.84	5.20×	6.31	4.93×	6.83	5.38×	6.17	4.84×	5.26	4.00×	5.38	4.14×
	TALON	6.71	5.22×	6.31	4.99×	7.37	5.67×	6.21	4.95×	5.12	4.07×	5.25	4.19×
	Solvo	6.90	**5.33**×	6.51	**5.11**×	7.54	**5.76**×	6.37	**5.04**×	5.28	**4.15**×	5.46	**4.29**×
DSL-8B	EAGLE-3	5.64	4.16×	7.10	5.23×	6.58	4.86×	5.87	4.34×	5.06	3.77×	5.03	**3.74**×
	TALON	5.42	4.17×	7.66	5.53×	6.39	4.83×	5.81	4.40×	4.85	3.80×	4.83	**3.74**×
	Solvo	5.59	**4.24**×	7.87	**5.62**×	6.62	**4.93**×	6.03	**4.50**×	4.99	**3.84**×	4.93	**3.74**×
Vicuna-13B	EAGLE-3	6.59	3.78×	6.57	3.88×	7.82	4.65×	6.49	3.81×	5.33	3.13×	6.65	3.64×
	TALON	6.56	3.83×	6.79	3.95×	9.62	5.34×	6.77	3.96×	5.26	3.18×	7.32	3.87×
	Solvo	6.72	**3.93**×	7.01	**4.08**×	9.73	**5.41**×	6.93	**4.04**×	5.43	**3.26**×	7.52	**3.94**×
Qwen3-8B	EAGLE-3	3.43	2.57×	3.95	2.95×	3.91	2.93×	3.64	2.73×	3.47	2.61×	3.28	2.44×
	TALON	3.42	2.72×	3.89	3.04×	3.83	3.03×	3.59	2.85×	3.44	2.74×	3.21	2.52×
	Solvo	3.47	**2.80**×	3.94	**3.14**×	3.89	**3.13**×	3.63	**2.93**×	3.49	**2.83**×	3.27	**2.61**×
Qwen3-32B	EAGLE-3	2.83	2.02×	3.37	2.41×	2.99	2.14×	2.99	2.13×	2.68	1.91×	2.57	1.84×
	TALON	2.78	2.07×	3.31	2.44×	2.91	2.16×	2.92	2.16×	2.63	1.96×	2.53	1.88×
	Solvo	2.84	**2.13**×	3.38	**2.51**×	3.01	**2.25**×	2.98	**2.24**×	2.71	**2.04**×	2.59	**1.95**×

Baselines and protocol. We compare against EAGLE-3 and TALON under stochastic ($T = 1$) and greedy ($T = 0$) decoding, and additionally against RheoSampling on all five targets at $T = 1$. Our implementation builds on the EAGLE-3 repository. All methods share the same CUDA Graph-enabled drafting backend. Please refer to Appendix D for more details.

Metrics. We report acceptance length τ and speedup ratio (SR). For each prompt, acceptance length is the mean number of tokens generated per drafting–verification cycle; τ averages this quantity across prompts. For each task and run, SR is the method's mean per-prompt generation rate divided by the matched autoregressive mean for that task. A six-task aggregate SR is the arithmetic

mean of the six task-wise SRs, and reported values are then averaged over six runs. Table 1 reports the task-wise values, while all cross-task aggregates throughout the paper use this six-task mean.

5.2 Main results

Table 1 reports the full model–task matrix: Solvo is fastest or tied for fastest in every evaluated configuration. It also achieves the highest six-task mean SR for all five targets under both decoding settings; these aggregates are reported in Appendix C.

Stochastic decoding: exploiting residual overlap. At $T = 1$, Solvo improves aggregate SR over EAGLE-3 by 7.3–10.2% and over the strongest reported baseline by 3.2–5.4%. Stochastic decoding flattens the target distribution, so every method accepts fewer tokens than at $T = 0$; the size of this drop reflects how well its candidates cover the flattened distribution. With deterministic candidates, aggregate acceptance length drops by 17.2% for EAGLE-3 and 17.0% for TALON on Llama3.1-8B, and by 16.0% and 15.0% on DSL-8B. Solvo drops by only 13.3% and 10.6%, as its residual samples recover part of the draft–target overlap outside the deterministic candidates. Correspondingly, the average gain over EAGLE-3 rises from 5.6% at $T = 0$ to 8.6% at $T = 1$; the component analysis below isolates their individual contributions.

Greedy decoding: adaptive depth. At $T = 0$, sampled candidates play no role, yet Solvo still improves aggregate SR over EAGLE-3 by 2.9–7.8% and over the strongest reported baseline by 1.4–3.7%. These gains come mainly from depth control: cost-aware stopping avoids low-yield draft forwards, while the absence of a fixed depth allows deeper growth when continuation remains promising. On Vicuna-13B HumanEval, for example, Solvo reaches $\tau = 9.73$, beyond EAGLE's fixed-depth ceiling and above EAGLE's 7.82 and TALON's 9.62.

5.3 Component analysis

Table 2 evaluates cumulative additions to EAGLE-3. Threshold construction, including root refill, and cost-aware stopping raise SR from $3.99\times$ to $4.16\times$, despite a slight reduction in τ at the stopping stage. The sampling-related components add a comparable gain: mixed proposals with the single-layer OT verifier of Hu et al. (2025) reach $4.21\times$, and an advanced verifier (UniVer) completes Solvo at $4.33\times$.

Table 2: Cumulative components on Llama3.1-8B ($T = 1$). τ is macro acceptance length and SR is the six-task mean of task-wise speedup ratios; entries are means over six runs.

Configuration	τ	SR
EAGLE-3	5.08	$3.99\times$
+ Threshold tree	5.24	$4.10\times$
+ Cost-aware stopping	5.23	$4.16\times$
+ Mixed proposals	5.34	$4.21\times$
+ Advanced verifier	**5.50**	**4.33**$\times$

The verifier's gain depends on sampled candidates. On deterministic-only trees such as EAGLE-3 and TALON, sampling-aware lossless verifiers reduce to the same deterministic rule. UniVer therefore has zero theoretical acceptance gain on EAGLE-3, and its heavier verification tends to lower the speedup ratios (Appendix F). On Solvo's trees, by contrast, it raises SR from $4.21\times$ to $4.33\times$, since residual samples give it probability mass to reallocate. Its benefit thus relies on a construction that admits sampled candidates losslessly (Section 4).

5.4 Effect of sampling temperature

Figure 2 varies T from 0.2 to 1.0 on MT-Bench, GSM8K, and HumanEval. Solvo has the highest mean speedup at every tested temperature in all three tasks. Its relative margin over the deterministic baselines grows at higher temperature, and RheoSampling likewise separates from EAGLE-3 only at higher temperatures, consistent with sampled candidates becoming more valuable as the target distribution flattens.

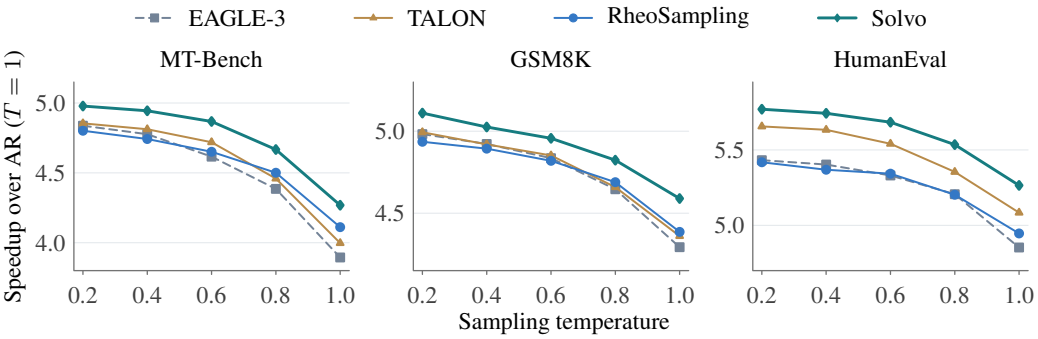

Figure 2: Sampling-temperature sweep on Llama3.1-8B with four methods. Each task uses its fixed autoregressive reference at $T = 1$ for all temperatures; points are means over six runs.

6 Related Work

Tree construction. Early methods such as SpecInfer and the original EAGLE use predefined branching structures (Miao et al., 2024; Li et al., 2024a), while Sequoia optimizes structures offline (Chen et al., 2024). Predetermined positions accommodate sampled candidates without identity-dependent retention. EAGLE-2/3 and OPT-Tree instead select tokens by context-dependent scores (Li et al., 2024b; 2025; Wang et al., 2025), coupling topology with candidate identity and complicating direct stochastic sampling. Further developments include budget-driven growth in TALON, classifier-based expansion and pruning in C2T, and cost-aware expansion control in SMART (Liu et al., 2026; Huo et al., 2025; Wang & Zhou, 2026).

Proposal design and verification. Standard speculative sampling uses token-level rejection sampling to preserve the target distribution (Leviathan et al., 2023; Chen et al., 2023). Along the vertical dimension, ASpS and Block Verification exploit sequence-level dependence beyond locally decided token acceptance (Hu & Huang, 2024; Sun et al., 2025). Horizontally, recursive rejection sampling (RRS) and its without-replacement variant (RRSw) verify sibling candidates, as in SpecInfer, Sequoia, the original EAGLE, and RSD (Miao et al., 2024; Chen et al., 2024; Li et al., 2024a; Jeon et al., 2024); SpecTr, SpecHub, and Greedy further improve multi-candidate proposal design and probability allocation (Sun et al., 2023; 2024; Hu et al., 2025). Traversal Verification and UniVer combine horizontal candidate verification with vertical cross-step coordination at the tree level (Weng et al., 2025; 2026).

Stochastic proposals in context-aware trees. RheoSampling introduces stochastic proposals into EAGLE-style dynamic trees while controlling sampling–pruning dependencies (Hu et al., 2026). Solvo instead makes proposal-law preservation a construction requirement through allocate-then-sample, supporting stochastic proposals without exposing them to subsequent global selection (Section 3.2).

7 Conclusion

We presented Solvo, a sampling-centric framework for tree speculative decoding. EAGLE-style dynamic trees decide retention by global pruning after tokens are expanded, so admitting sampled candidates requires controlling how a sample's identity affects its own survival and that of its siblings and descendants. We showed that this complexity is an artifact of pruning after sampling rather than inherent to sampling in dynamic trees: by allocating slots before materializing candidates and never reranking them, Solvo lets sampled tokens guide subsequent growth without pruning-induced selection bias. Using a martingale argument over tree growth, we proved that Solvo preserves the exact target-model distribution under adaptive growth and cost-aware stopping. Across five target models, Solvo consistently improves throughput under stochastic decoding, and it remains faster under greedy decoding. These results suggest that sampling should be treated as a design principle of tree construction rather than an addition to it.

AI USE STATEMENT

In this work, we used generative AI tools to assist in writing the proofs, to implement parts of the experimental system, to help summarize and interpret experimental results, and to assist with translation. All proof ideas and key steps were developed by the authors, and AI tools were used to write them up. AI tools contributed to the implementation of the baselines (e.g., TALON) and of the CUDA Graph-based drafting backend. The research questions, the proposed method, and all mathematical claims were developed by the authors without generative AI. The experiments were designed by the authors. Generating synthetic data, cleaning or reformatting datasets, and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to draft and edit parts of the manuscript for readability, to search for and summarize related literature, to format references, and to organize raw experiment logs into tables and summary statistics. All AI-assisted proofs, code, and text were reviewed and checked by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The experimental protocol, including target–draft pairs, datasets, prompt counts, decoding settings, metrics, and aggregation, is specified in Section 5. Appendix D reports implementation details, hardware, tree configurations, thresholds, and latency calibration. Section 4 states the assumptions underlying our theoretical results, and Appendix B provides the complete proofs. The evaluated datasets and pretrained models are publicly available from the sources listed in Appendix I; the appendices additionally provide hyperparameter sweeps, ablations, and run-to-run uncertainty.

REFERENCES

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, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 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 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687.

Qiao Hu, Yepeng Weng, Bo Zhang, and Takehisa Yairi. RheoSampling: Resolving the one-hot dilemma in stochastic dynamic-tree speculative decoding. *arXiv preprint arXiv:2609.21827*, 2026.

Zhengmian Hu and Heng Huang. Accelerated speculative sampling based on tree monte carlo. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.

Zhengmian Hu, Tong Zheng, Vignesh Viswanathan, Ziyi Chen, Ryan A. Rossi, Yihan Wu, Dinesh Manocha, and Heng Huang. Towards optimal multi-draft speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025.

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.

Wonseok Jeon, Mukul Gagrani, Raghavv Goel, Junyoung Park, Mingu Lee, and Christopher Lott. Recursive speculative decoding: Accelerating LLM inference via sampling without replacement. *arXiv preprint arXiv:2402.14160*, 2024.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 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, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, 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 *Annual Conference on Neural Information Processing Systems*, 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 large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, 2024.

Ramesh Nallapati, Bowen Zhou, Cícero Nogueira dos Santos, Çaglar Gülçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, 2016.

Ryan Sun, Tianyi Zhou, Xun Chen, and Lichao Sun. SpecHub: Provable acceleration to multi-draft speculative decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, 2024.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix X. Yu. SpecTr: Fast speculative decoding via optimal transport. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Ziteng Sun, Uri Mendlovic, Yaniv Leviathan, Asaf Aharoni, Jae Hun Ro, Ahmad Beirami, and Ananda Theertha Suresh. Block verification accelerates speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An instruction-following LLaMA model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

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. doi: 10.1162/tacl_a_00735.

Lifu Wang and Pan Zhou. SMART: When is it actually worth expanding a speculative tree? *arXiv preprint arXiv:2604.09731*, 2026.

Yepeng Weng, Qiao Hu, Xujie Chen, Li Liu, Dianwen Mei, Huishi Qiu, Jiang Tian, and Zhongchao Shi. Traversal verification for speculative tree decoding. In *Advances in Neural Information Processing Systems*, 2025.

Yepeng Weng, Qiao Hu, and Takehisa Yairi. UniVer: A unified perspective for multi-step and multi-draft speculative decoding. *arXiv preprint arXiv:2605.04543*, 2026.

An Yang et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

A The Threshold as an Approximation of Global Pruning

Exact case. Fix a set of expanded candidate slots and let $s_{(B)}$ be the B-th largest structural score among them. Retaining the slots with score at least $s_{(B)}$ returns exactly the node set of global top-B pruning over the same slots. Thresholding and ranking therefore agree whenever the threshold is placed at the rank-B boundary, and a threshold construction gives up nothing relative to global pruning, provided it can be placed there.

Why it cannot be placed there. Locating $s_{(B)}$ requires the pool to exist first: one must expand a wide frontier, score the resulting candidates, and rank them, which is precisely EAGLE-style expand-then-prune. Doing so after sampling is what couples a sample's retention to its realized identity (Section 2.2). Solvo therefore fixes θ in advance, which decouples retention from sampling at the cost of an approximation.

The two directions of slack. For a fixed expanded pool, when $\theta < s_{(B)}$, the construction admits slots that global pruning would discard, and the budget is exhausted earlier than it otherwise would be; the layer-wise truncation by score bounds how far this can go, but coverage is lost relative to a better-placed boundary. When $\theta > s_{(B)}$, the construction admits fewer slots than the budget allows, and the surplus is refilled with unused root children. Cost-aware stopping or the depth cap can also leave budget unspent, and the same refill applies. Refilled nodes are appended as leaves, so they add coverage without disturbing the existing tree. Appendix G.1 reports the empirical cost of this slack: performance is relatively flat near the selected defaults, with smaller θ widening the frontier and raising drafting overhead, and larger θ leaving more of the budget to refill.

Relationship to other dynamic-tree constructions. EAGLE does perform global top-B pruning, but over a pool drawn from a preset number of expansion layers; its weakness under greedy decoding stems mainly from this fixed depth rather than from global pruning itself. TALON (Liu et al., 2026) replaces the preset depth with a relative threshold and budget exhaustion, and its gains over EAGLE-3 confirm that adapting the tree shape to the context is worth having. Solvo's construction resembles it in form, and our greedy-decoding gains are consistent with the same observation. The motivation, however, is different. For TALON the tree is the object of study, and adaptive shape is the goal. For Solvo the tree is a means: the threshold exists so that retention can be committed before sampling, and the resulting tree shape follows from that commitment rather than from a target shape.

B PROOFS OF THE THEORETICAL GUARANTEES

We prove Lemma 4.2 and Theorem 4.3 using the stochastic tree growth and target-completed output $M_k(x)$ defined in Section 4. First, we establish the conditional local mass identity used by Solvo's verifier. We then show that each growth step preserves the target-completed output law, including the final root refill, and conclude losslessness under adaptive stopping and repeated verification cycles.

B.1 PROOF SETUP AND LOCAL VERIFICATION

Throughout this appendix, $p(\cdot \mid u)$ and $q(\cdot \mid u)$ denote the target and draft next-token distributions, respectively, at token prefix u. The notation used in the subsequent proofs is summarized in Table 3, which serves as a quick reference for the objects introduced below.

Table 3: Notation reference for the theoretical proofs.

Symbol	Meaning
Tree growth and target completion	
$k,\ T_k,\ T_{k+1}$	Growth-stage index, current verification tree, and tree after adding one child group.
$p(\cdot \mid u),\ q(\cdot \mid u)$	Target and draft next-token distributions at u.
$r,\ u,\ ut$	Root, node/token prefix, and prefix u extended by t.
$P_u(x)$	Probability that continuing the generated string u with the target model yields a sequence with prefix x.
$P_{\mathrm{target}}(x)$	The probability that target generation starts with x, or equivalently, $P_r(x)$.
$M_k(x)$	Given T_k, probability that running the verifier and then continuing its output with the target model yields a sequence with prefix x.
$\sigma(T_k)$	Information contained in T_k; it also determines the preceding construction history.
Local verification	
$C(u),\ c_j,\ m,\ S_u$	Ordered child group, its jth token label, group width, and set of child labels.
η_u	Conditional acceptance probability upon reaching node u; $\eta_r = 1$.
$a_u(t),\ d_u$	Local mass assigned to token t and to rejection of the subtree at u.
$w_{u,j},\ \eta_{uc_j}$	Mass remaining before child j and the normalized acceptance probability of that child.
$b_u,\ U_u$	Local stopping probability and its independent uniform decision variable.
Sampled boundary allocation	
$H_u,\ Z_u,\ s_u$	Leading deterministic tokens, sampled boundary token, and its conditional proposal distribution.
$R_u(t),\ \kappa_u,\ h_u(z)$	Residual target mass, its normalization constant, and acceptance factor for sample z.
$a_u(t; z),\ d_u(z)$	Realized allocation and rejection masses conditional on $Z_u = z$.
$\mathbf{1}\{\cdot\},\ [z]_+$	Indicator function and positive part $\max\{z, 0\}$.
One-step preservation and stopping	
$L_k(u, x),\ D_k(u)$	Local target-completed success mass and rejection probability before expansion.
$A_k(x),\ \rho_k,\ B_k(x)$	Success mass before reaching u, probability of reaching u, and success mass after its subtree rejects.
$N,\ \tau$	Bounded terminal growth stage and an arbitrary stopping time satisfying $\tau \le N$.

Information carried by T_k. Start with the root-only tree T_1. Adding the initial deterministic root-child group gives T_2; adding the next child group at a retained leaf gives T_3, and so on. *Each growth step adds one group without changing earlier nodes or their token labels*. The input prefix, draft model, target model, and construction rules are fixed, including tie-breaking and the order of groups within each layer.

Now read this process backward. Given T_k, we can identify the last added group and remove it to recover T_{k-1}, then repeat to recover the entire history. The group order is recoverable because the fixed construction rules determine each successive allocation from the preceding tree, while all realized tokens remain in T_k. Thus T_k records not just the current tree but enough information to reconstruct how it grew; structural scores and other deterministic construction quantities can also be recomputed. In information terms,

$$\sigma(T_1, \ldots, T_k) = \sigma(T_k),$$

where $\sigma(\cdot)$ denotes the generated σ-algebra. We therefore write conditioning on the construction history simply as conditioning on T_k. Given T_k, the parent and the slot allocation for the new child group are already determined. Any random token in this group is then sampled conditional on T_k, and adding the group gives T_{k+1}.

Target completion. The root r represents the empty continuation after the fixed input prefix. We identify a node u with its root-to-node token sequence and write ut for this sequence extended by token t, or for the corresponding child when present in the tree. Given a generated token string u, let $P_u(x)$ denote the probability that continuing u with the target model yields a sequence with prefix x. It is 1 if x is already a prefix of u, 0 if u and x are incompatible, and otherwise the probability of generating the remaining tokens of x. Conditioning on the first target token after u gives

$$P_u(x) = \sum_{t \in \mathcal{V}} p(t \mid u) P_{ut}(x), \qquad P_r(x) = P_{\text{target}}(x). \tag{4}$$

After drawing t, the fixed prefix becomes ut, and the same prefix event has probability $P_{ut}(x)$.

We apply the same completion procedure to the output of a verification cycle on T_k: first run the verifier, then continue from its output using the target model. As in Section 4.1, define

$$M_k(x) := \Pr(\text{the target-completed output on } T_k \text{ starts with } x \,|\, \sigma(T_k)) .$$

Here T_k is fixed, and the probability averages over the verifier's random decisions and subsequent target-model generation. Thus, $P_u(x)$ *starts from a fixed token sequence* u*, whereas* $M_k(x)$ *starts by running the verifier on a fixed tree* T_k.

An example. Treat each word as one token and let $x =$ `The cat sat`. For $u =$ `The cat`, $P_u(x)$ is simply the target model's probability of generating `sat` next: `The cat` is already fixed. In contrast, $M_k(x)$ first runs the verifier on T_k. Depending on the verifier's output, there are three cases:

1. **A matching but incomplete prefix.** If the verifier outputs `The cat`, the target model must next generate `sat` to match x. If it outputs only `The`, the target model must next generate `cat` and then `sat`. The respective probabilities are $p($`sat` $\mid$ `The cat`$)$ and $p($`cat` $\mid$ `The`$)p($`sat` $\mid$ `The cat`$)$.
2. **A mismatching prefix.** If the verifier outputs `The dog`, no continuation can make the sequence start with x. Conditional on this output, the probability of matching x is 0.
3. **An already complete match.** If the verifier outputs `The cat sat`, or a longer sequence starting with these words, the probability of matching x is 1. No further target completion is needed to determine this event.

$M_k(x)$ is **the weighted average** of these matching probabilities over all possible verifier outputs, with each output weighted by its probability given T_k.

Local probability allocation. We use the Allocation–Decision formulation of UniVer (Weng et al., 2026). Let $\eta_u \in [0, 1]$ denote the conditional acceptance probability at node u, with $\eta_r = 1$. It corresponds to UniVer's effective conditional acceptance probability $\widetilde{p}_u$, rather than the marginal probability of generating prefix u.

Consider an ordered child group with distinct labels,

$$C(u) = (uc_1, \ldots, uc_m), \qquad S_u = \{c_1, \ldots, c_m\}.$$

Write $a_u(t)$ for the local mass assigned to token $t \in \mathcal{V}$, and d_u for the mass assigned to rejection of the entire local subtree. Feasibility requires

$$a_u(t) \geq 0, \qquad d_u \geq 0, \qquad \sum_{t \in \mathcal{V}} a_u(t) + d_u = 1. \tag{5}$$

For $t \in S_u$, the mass is assigned to the corresponding child; for $t \notin S_u$, it is assigned to fallback at u. Define

$$\begin{aligned} w_{u,j} &= 1 - \sum_{i<j} a_u(c_i), && j = 1, \ldots, m+1, \\ \eta_{uc_j} &= \begin{cases} a_u(c_j)/w_{u,j}, & w_{u,j} > 0, \\ 0, & w_{u,j} = 0, \end{cases} && j = 1, \ldots, m. \end{aligned} \tag{6}$$

After one target-model pass on T, allocation proceeds top-down, in parallel across nodes at each depth, as in UniVer (Weng et al., 2026). The Decision phase follows the post-order traversal of Traversal Verification and UniVer (Weng et al., 2025; 2026), respecting the child order above. Each node has a stopping probability

$$b_u = \begin{cases} \eta_u, & u \text{ is an unexpanded leaf}, \\ \dfrac{\sum_{t \notin S_u} a_u(t)}{w_{u,m+1}}, & u \text{ is expanded and } w_{u,m+1} > 0, \\ 0, & \text{otherwise}. \end{cases}$$

Using independent uniforms $(U_u)_{u \in T}$, test $U_u < b_u$ in parallel and return the first success in post-traversal order: its prefix plus a token drawn from $p(\cdot \mid u)$ at a leaf, or proportionally to $a_u(t)$ over $t \notin S_u$ otherwise.

B.2 Conditional Local Mass Preservation

Lemma B.1 (Conditional local mass identity)**.** *For every Solvo growth step $T_k \to T_{k+1}$, let u be the parent whose child group is materialized or extended, and write $a_u(t)$ and d_u for its post-step local allocation. Then, for every $t \in \mathcal{V}$,*

$$\mathbb{E}_{\text{Solvo}}[a_u(t) \mid \sigma(T_k)] = \eta_u p(t \mid u), \qquad \mathbb{E}_{\text{Solvo}}[d_u \mid \sigma(T_k)] = 1 - \eta_u. \tag{7}$$

These identities hold for the initial root-group expansion, every standard non-root expansion, and terminal root refill.

Proof. Condition on T_k. If u is the root r, then $\eta_r = 1$ and the deterministic root allocation is

$$a_r(t) = p(t \mid r), \qquad d_r = 0. \tag{8}$$

This proves equation 7 pointwise for the initial root expansion. Terminal refill only appends deterministic root leaves, transferring their labels from root fallback to explicit children without changing equation 8; hence the same pointwise identity holds after refill.

It remains to consider a non-root expansion. The conditional acceptance probability η_u is fixed through equation 6. For the child group $C(u)$ of width m, let $H_u = \{v_{u,1}, \ldots, v_{u,m-1}\}$ be its leading deterministic tokens. The boundary token Z_u has conditional distribution

$$s_u(t) := \frac{q(t \mid u)\mathbf{1}\{t \notin H_u\}}{1 - \sum_{h \in H_u} q(h \mid u)}, \qquad \Pr(Z_u = t \mid \sigma(T_k)) = s_u(t). \tag{9}$$

For $m = 1$, $s_u = q(\cdot \mid u)$.

Equation 9 is the conditional sampling law. Once drawn, Z_u is fixed: subsequent growth only adds nodes and never changes existing token labels.

For $[z]_+ = \max\{z, 0\}$, define

$$R_u(t) = [\eta_u p(t \mid u) - s_u(t)]_+,$$

$$\kappa_u = 1 - \sum_{t \in \mathcal{V}} \min\{\eta_u p(t \mid u), s_u(t)\} = 1 - \eta_u + \sum_{t \in \mathcal{V}} R_u(t), \tag{10}$$

$$h_u(z) = \min\left\{1, \frac{\eta_u p(z \mid u)}{s_u(z)}\right\}, \qquad z \in \operatorname{supp} s_u. \tag{11}$$

These quantities depend only on the current state, except for the argument z of h_u. For $\kappa_u > 0$, UniVer's allocation can be written compactly as

$$a_u(t; z) = h_u(z)\mathbf{1}\{t = z\} + \frac{1 - h_u(z)}{\kappa_u} R_u(t), \tag{12}$$

$$d_u(z) = \frac{1 - h_u(z)}{\kappa_u}(1 - \eta_u). \tag{13}$$

Since $(1 - h_u(z))R_u(z) = 0$, the sampled child receives exactly $h_u(z)$. The other masses are assigned to deterministic children or fallback according to their labels. By equation 10, all masses are nonnegative and

$$\sum_t a_u(t; z) + d_u(z) = h_u(z) + \frac{1 - h_u(z)}{\kappa_u} \left(\sum_t R_u(t) + 1 - \eta_u \right) = 1.$$

Moreover, equation 9 gives

$$\mathbb{E}_{\text{Solvo}}[1 - h_u(Z_u) \mid \sigma(T_k)] = \kappa_u.$$

Hence, for every $t \in \mathcal{V}$,

$$\begin{aligned} \mathbb{E}_{\text{Solvo}}[a_u(t; Z_u) \mid \sigma(T_k)] &= \min\{s_u(t), \eta_u p(t \mid u)\} + R_u(t) = \eta_u p(t \mid u), \\ \mathbb{E}_{\text{Solvo}}[d_u(Z_u) \mid \sigma(T_k)] &= 1 - \eta_u. \end{aligned} \tag{14}$$

If $\kappa_u = 0$, equation 10 implies $\eta_u = 1$ and $s_u = p(\cdot \mid u)$. Set $a_u(t; z) = \mathbf{1}\{t = z\}$ and $d_u(z) = 0$; the same conditional identities hold without division by zero.

For $\kappa_u > 0$, equation 14 proves equation 7; the $\kappa_u = 0$ case gives the same identity. Together with the deterministic root cases above, this covers every Solvo growth step. □

Causality and edge cases. The conditional acceptance probability η_u is determined entirely by the fixed allocations along the path from the root to u. In a growth step $T_k \to T_{k+1}$, the only new target distributions are those associated with the newly added child group. The corresponding parameter update is local to that group and its parent: all other parameters remain unchanged, and removing the group recovers the preceding state. Thus T_{k+1} adds only the information carried by the new child group.

If $\eta_u = 0$, the node and all its descendants receive zero acceptance mass, so the local subtree rejects. If $\eta_u = 1$, then $d_u = 0$, so the subtree cannot reject: either fallback succeeds or a child with parameter one succeeds recursively. Finally, $w_{u,j} = 0$ means that an earlier child succeeds with probability one. The current and all later children are therefore unreachable, so assigning their parameters zero has no effect.

B.3 Proof of Lemma 4.2: One-step Preservation

Proof. Fix a finite token prefix x and condition on T_k. By the locality established above, a growth step changes only the affected parent and its newly added children. All verifier behavior outside this local part of the tree remains fixed. We distinguish leaf expansion from terminal root refill.

Leaf expansion. Let u be the expanded leaf. Before expansion, conditional on the traversal reaching u, its target-completed success mass is

$$L_k(u, x) = \eta_u P_u(x),$$

and its rejection probability is $D_k(u) = 1 - \eta_u$.

After expansion, ordered normalization in equation 6 assigns total probability $a_u(c_j)$ to child uc_j, while fallback assigns mass $a_u(t)$ to each $t \notin S_u$. Thus the corresponding local quantities become

$$L_{k+1}(u, x) = \sum_{t \in \mathcal{V}} a_u(t) P_{ut}(x), \qquad D_{k+1}(u) = d_u. \tag{15}$$

Here an explicit child labeled t and fallback to token t have the same target-completed continuation probability $P_{ut}(x)$.

Lemma B.1 and the target-completion identity equation 4 give

$$\begin{aligned} \mathbb{E}_{\text{Solvo}}[L_{k+1}(u, x) \mid \sigma(T_k)] &= \eta_u \sum_{t \in \mathcal{V}} p(t \mid u) P_{ut}(x) \\ &= \eta_u P_u(x) = L_k(u, x), \\ \mathbb{E}_{\text{Solvo}}[D_{k+1}(u) \mid \sigma(T_k)] &= 1 - \eta_u = D_k(u). \end{aligned} \tag{16}$$

To place this local replacement back into the full traversal, let $A_k(x)$ be the target-completed success probability accumulated before reaching u, ρ_k the probability of reaching u, and $B_k(x)$ the target-completed success probability after the local subtree rejects. These quantities are fixed by T_k and unchanged by the expansion. Therefore

$$M_{k+1}(x) = A_k(x) + \rho_k\big[L_{k+1}(u,x) + D_{k+1}(u)B_k(x)\big],$$

whereas $M_k(x)$ has the same expression with $L_k(u,x)$ and $D_k(u)$. Taking the conditional expectation and applying equation 16 yields

$$\begin{aligned}\mathbb{E}_{\mathrm{Solvo}}[M_{k+1}(x) \mid \sigma(T_k)] &= A_k(x) + \rho_k\Big(\\ &\qquad \mathbb{E}_{\mathrm{Solvo}}[L_{k+1}(u,x) \mid \sigma(T_k)] \\ &\qquad + \mathbb{E}_{\mathrm{Solvo}}[D_{k+1}(u) \mid \sigma(T_k)]B_k(x)\Big) \\ &= A_k(x) + \rho_k\big[L_k(u,x) + D_k(u)B_k(x)\big] \\ &= M_k(x). \end{aligned} \tag{17}$$

This covers both the initial deterministic root-group expansion and every non-root mixed-group expansion. The equality is immediate if u is unreachable.

Terminal root refill. Because refill leaves are appended after all existing root children, they are reached only if every pre-refill root-child subtree rejects. For each refilled token t, the deterministic root allocation assigns mass $p(t \mid r)$: before refill it is emitted through root fallback, and after refill through the explicit leaf rt. Both contribute the same target-completed mass $p(t \mid r)P_{rt}(x)$. All other outcomes are unchanged, so

$$M_{k+1}(x) - M_k(x) = \sum_{t:\,\text{refilled}} \big[p(t \mid r)P_{rt}(x) - p(t \mid r)P_{rt}(x)\big] = 0. \tag{18}$$

Thus root refill preserves $M_k(x)$ pointwise and completes the proof. □

B.4 Proof of Theorem 4.3: Losslessness of Solvo

Proof. Fix a finite token prefix x. Since $0 \le M_k(x) \le 1$, Lemma 4.2 and $\sigma(T_1, \ldots, T_k) = \sigma(T_k)$ show that $\{M_k(x)\}$ is a bounded martingale with respect to $\{\sigma(T_k)\}$.

Adaptive stopping. Whether growth stops after T_k is determined by T_k itself. Because the node budget is finite and every nonempty growth step strictly increases the number of retained nodes, the terminal stage N is a bounded stopping time. The optional stopping theorem therefore applies to any stopping time $\tau \le N$:

$$\mathbb{E}_{\mathrm{Solvo}}[M_\tau(x)] = M_1(x) = P_{\mathrm{target}}(x). \tag{19}$$

The last equality follows because T_1 is the root-only tree with $\eta_r = 1$, so its target-completed output is direct target generation. Taking $\tau = N$ gives a complete Solvo verification cycle and proves its losslessness, while smaller stopping times show that Solvo may stop losslessly at any earlier growth stage.

Repeated verification cycles. The same argument applies after any fixed generated prefix, following the cycle-composition argument for block verification (Sun et al., 2025, Lemma 2). Thus equation 19 allows a target continuation to be replaced by a Solvo cycle without changing the output law. Since $\eta_r = 1$, every cycle emits at least one token, so this replacement can be repeated for any finite output prefix. Therefore

$$P_{\mathrm{Solvo}}(x) = P_{\mathrm{target}}(x) = \prod_{i=1}^{|x|} p(x_i \mid x_{<i}),$$

with the fixed input prefix implicit. EOS is treated as absorbing, with the generated sequence truncated at its first occurrence. □

Other proposal rules and greedy decoding. The proof uses only the conditional local mass identity in Lemma B.1, the ordered Decision rule, and deterministic root behavior. It therefore also applies to causally generated child groups satisfying the same identity. At temperature zero, all groups are deterministic, and setting $a_u(t) = \eta_u p(t \mid u)$ and $d_u = 1 - \eta_u$ gives the same losslessness guarantee for greedy decoding.

C Run-to-Run Uncertainty

For each configuration, we conduct six runs with different seeds on the fixed prompt set. For each run, the six-task SR is the arithmetic mean of the six task-wise SRs. Table 4 reports its mean, sample standard deviation s, and two-sided 95% Student-t confidence-interval half-width $t_{0.975,5}s/\sqrt{6}$ across six runs, together with the corresponding aggregate of τ. The experimental unit is a full run, and the intervals quantify run-to-run variability on the fixed prompt set. At $T = 0$, acceptance lengths are identical across runs, so their standard deviations and interval half-widths are omitted.

Table 4: Six-run uncertainty of the six-task aggregates. CI is the 95% Student-t half-width across six runs.

Model	Method	T	τ	SD	CI	SR	SD	CI
Llama3.1-8B	EAGLE-3	1	5.075	0.196	0.206	3.986	0.111	0.116
	TALON	1	5.117	0.194	0.203	4.084	0.138	0.145
	RheoSampling	1	5.304	0.117	0.122	4.159	0.088	0.092
	Solvo	1	5.500	0.031	0.032	4.334	0.043	0.045
	EAGLE-3	0	6.132	–	–	4.748	0.116	0.121
	TALON	0	6.163	–	–	4.849	0.042	0.044
	Solvo	0	6.345	–	–	4.945	0.019	0.020
DSL-8B	EAGLE-3	1	4.937	0.113	0.119	3.662	0.093	0.097
	TALON	1	4.956	0.141	0.148	3.784	0.119	0.125
	RheoSampling	1	5.176	0.074	0.077	3.863	0.070	0.074
	Solvo	1	5.371	0.105	0.111	4.035	0.094	0.099
	EAGLE-3	0	5.879	–	–	4.351	0.035	0.036
	TALON	0	5.827	–	–	4.414	0.024	0.025
	Solvo	0	6.006	–	–	4.478	0.072	0.075
Vicuna-13B	EAGLE-3	1	5.727	0.170	0.178	3.333	0.094	0.099
	TALON	1	6.000	0.241	0.252	3.479	0.119	0.125
	RheoSampling	1	5.863	0.134	0.141	3.406	0.073	0.076
	Solvo	1	6.304	0.148	0.155	3.632	0.076	0.079
	EAGLE-3	0	6.576	–	–	3.814	0.005	0.005
	TALON	0	7.054	–	–	4.023	0.002	0.002
	Solvo	0	7.222	–	–	4.110	0.004	0.005
Qwen3-8B	EAGLE-3	1	3.471	0.024	0.025	2.595	0.040	0.042
	TALON	1	3.418	0.023	0.024	2.715	0.039	0.041
	RheoSampling	1	3.400	0.017	0.018	2.549	0.045	0.047
	Solvo	1	3.501	0.010	0.011	2.801	0.054	0.057
	EAGLE-3	0	3.615	–	–	2.706	0.042	0.044
	TALON	0	3.563	–	–	2.817	0.061	0.064
	Solvo	0	3.614	–	–	2.907	0.052	0.054
Qwen3-32B	EAGLE-3	1	2.760	0.023	0.024	1.972	0.017	0.018
	TALON	1	2.703	0.021	0.022	2.008	0.015	0.016
	RheoSampling	1	2.733	0.014	0.014	1.953	0.010	0.010
	Solvo	1	2.825	0.004	0.004	2.116	0.003	0.004
	EAGLE-3	0	2.906	–	–	2.075	0.000	0.000
	TALON	0	2.847	–	–	2.111	0.001	0.001
	Solvo	0	2.918	–	–	2.188	0.001	0.001

D Implementation Details

D.1 Structural scores and sampling temperature

Solvo separates the probabilities used for structural allocation from those used for sampling and verification. Let $\ell_x(u)$ be the draft logit of token x at prefix u, and let S_u contain the Top-128 draft tokens. For $x \in S_u$ and $T > 0$, we use

$$\bar{q}(x \mid u) = \frac{\exp(\ell_x(u))}{\sum_{y \in S_u} \exp(\ell_y(u))}, \qquad q_T(x \mid u) = \frac{\exp(\ell_x(u)/T)}{\sum_{y \in S_u} \exp(\ell_y(u)/T)}, \tag{20}$$

with both distributions zero outside S_u. The proposal q in the sampling and verification formulas denotes q_T. Structural path scores, slot allocation, and frontier-score sums instead use $\bar{q}$, without temperature scaling. This follows the unscaled scoring convention in the EAGLE-2/3 implementation and is intended to prevent low sampling temperatures from concentrating structural scores and collapsing the available branching structure. Sampled identities can still affect subsequent draft predictions. The Top-128 truncation is for acceleration and affects only the draft proposals; losslessness remains exact because verification uses the same truncated q_T that generated the samples.

For $T > 0$, $\bar{q}$ and q_T have the same support and token ordering, but generally different probabilities. The leading deterministic tokens are therefore unchanged by temperature scaling at a fixed prefix; the boundary token is sampled from q_T after masking and renormalizing away these tokens. Verification uses this actual residual proposal distribution, not $\bar{q}$ or the inherited slot score. At $T = 1$, $q_T = \bar{q}$. At $T = 0$, candidate groups are deterministic while structural allocation still uses $\bar{q}$.

D.2 Hardware and Tree configurations

All experiments are performed on an NVIDIA RTX PRO 6000 Blackwell Server Edition GPU (96 GB). For the main comparisons, EAGLE-3 uses frontier width $K = 10$ and eight draft expansion layers; TALON uses the official relative threshold $\mu = 0.03$. These baseline settings follow the defaults recommended in the original papers and are kept fixed across target models. Appendix G.2 additionally reports TALON with μ tuned per target model. RheoSampling uses its recommended rheostat parameter $m = 1$ with its original verification rule. All methods share the tree-token budget of 60 and the same CUDA Graph-enabled drafting backend, which keeps layer-wise expansion and control on the GPU. This matters most for dynamic-width expansion methods such as TALON, whose per-layer width would otherwise require a host synchronization at every draft layer. Because CUDA Graphs require static shapes, dynamic-width methods (TALON and Solvo) capture one graph per frontier width, using 16 width buckets in total to bound graph-capture memory. All evaluations use FP16 inference with batch size 1 and no concurrent requests. Solvo uses path-score threshold $\theta = 0.01$ for Llama3.1-8B, DSL-8B, and Vicuna-13B, and $\theta = 0.005$ for Qwen3-8B and Qwen3-32B. Appendix G.1 varies Solvo's θ.

D.3 Cycle-level latency calibration

For each target–draft pair, we run 128-token generation on four Alpaca prompts at $T = 1$, using the same CUDA Graph drafting path with early stopping disabled. From CUDA-synchronized cycle timings and actual draft-forward counts d, we robustly fit $\widehat{C}(d) = L_b + dL_d$. Here L_b includes target computation, verification, and fixed bookkeeping, while L_d estimates the marginal cost of another draft layer. These fitted costs parameterize Equation 3; Table 5 reports the calibrated costs and ratios.

Table 5: Cycle-level calibration on RTX PRO 6000. L_b and L_d are fitted costs in milliseconds; $\overline{C}$ is the mean measured calibration-cycle latency.

Model	L_b (ms)	L_d (ms)	L_d/L_b	$L_d/\overline{C}$
Llama3.1-8B	22.947	0.717	3.13%	1.97%
DSL-8B	22.363	0.777	3.47%	2.20%
Vicuna-13B	27.164	0.960	3.53%	2.82%
Qwen3-8B	22.599	0.732	3.24%	2.75%
Qwen3-32B	62.212	1.167	1.88%	1.71%

E ROOT SAMPLING ABLATION

The default Solvo configuration uses only deterministic root candidates. A sampled candidate would necessarily occupy the tail of the initially allocated root group, while root refill subsequently appends additional unused deterministic candidates. In the final first layer, the sampled candidate would therefore appear between two deterministic segments. Fully deterministic root children keep this special first-layer construction and its ordering cleaner. This choice does not make root sampling infeasible: we also evaluate a group-tail variant that replaces the last initially allocated root child with a residual sample, retains its structural score, and handles refill by deduplicating against its realized identity. We compare the two root policies at $T = 1$ on the same five target-draft pairs.

Table 6: Root-policy ablation. τ is macro acceptance length and SR is the six-task mean of task-wise speedup ratios; entries are means over six runs. Relative changes compare sampled with deterministic root children.

Model	Deterministic root		Sampled root		Relative change	
	τ	SR	τ	SR	$\Delta\tau$	ΔSR
Llama3.1-8B	5.50	4.33×	5.53	4.33×	$+0.49\%$	-0.07%
DSL-8B	5.37	4.04×	5.40	4.04×	$+0.48\%$	$+0.09\%$
Vicuna-13B	6.30	3.63×	6.36	3.64×	$+0.92\%$	$+0.17\%$
Qwen3-8B	3.50	2.80×	3.50	2.79×	$+0.05\%$	-0.58%
Qwen3-32B	2.82	2.12×	2.84	2.12×	$+0.39\%$	$+0.10\%$

Root sampling increases mean acceptance length on all five targets, but only by 0.05–0.92%. Handling the sampled root candidate and its residual refill can also introduce additional latency, leaving SR nearly unchanged: the relative differences range from -0.58% to $+0.17\%$. Both root policies therefore perform similarly; the main configuration uses deterministic root children for a simpler first layer.

F APPLYING UNIVER TO EAGLE-3

For a tree containing only deterministic candidates, such as EAGLE-3 or TALON, sampling-aware lossless verifiers reduce to the same deterministic-candidate rule. For any fixed tree and context, they therefore induce the same acceptance distribution, leaving zero theoretical acceptance gain from replacing the verifier. The small aggregate differences in τ arise because the two verifiers consume randomness differently, causing their stochastic generation trajectories to diverge even under matched initial seeds. Table 7 compares EAGLE-3's original verifier with UniVer (Weng et al., 2026) on five target models. UniVer adds verification overhead without an acceptance benefit and consequently yields slightly lower SR on four of the five targets.

Table 7: Applying UniVer to EAGLE-3 at $T = 1$. τ is macro acceptance length and SR is the six-task mean of task-wise speedup ratios; entries are means over six runs. Parentheses give relative changes from EAGLE-3's original verifier.

	Original		+ UniVer	
Model	τ	SR	τ	SR
Llama3.1-8B	5.08	3.99×	5.06 (-0.39%)	3.95× (-1.01%)
DSL-8B	4.94	3.66×	4.93 (-0.04%)	3.67× ($+0.19\%$)
Vicuna-13B	5.73	3.33×	5.69 (-0.72%)	3.29× (-1.24%)
Qwen3-8B	3.47	2.59×	3.47 ($+0.03\%$)	2.59× (-0.27%)
Qwen3-32B	2.76	1.97×	2.76 ($+0.01\%$)	1.97× (-0.14%)

G THRESHOLD SENSITIVITY

G.1 THRESHOLD SENSITIVITY OF SOLVO

We vary the path-score threshold θ at $T = 0$, using five settings for each of the five target–draft pairs. Each setting has six runs (seeds 0–5) on the same 480 prompts, with the tree-token budget held at 60. The default-threshold entries reuse the corresponding main-experiment runs. For each run, SR is the arithmetic mean of the six task-wise speedup ratios; we then average SR across runs. We sweep at $T = 0$ so that the threshold's effect on tree construction is isolated from sampling.

Table 8 reports the full greedy sweep using the six-task mean SR. We evaluate five settings per model: 0.002–0.020 for the Llama-family and Vicuna targets, and 0.001–0.010 for Qwen3. The selected defaults, 0.01 for the Llama-family and Vicuna targets and 0.005 for Qwen3, are underlined. The maximum improvement over the selected default among the tested settings is 0.85% on Qwen3-8B, at threshold 0.002. Nearby settings also perform similarly: increasing the threshold from 0.01 to 0.015 changes mean speed by less than 0.8% on each Llama-family and Vicuna target. These results indicate a relatively flat neighborhood around the defaults in the evaluated greedy configurations.

The same thresholds are used at T = 1 without further tuning. For the Qwen3 targets, the default was not chosen as the best-performing value in this sweep; selecting it would only increase Solvo's greedy speedups, so the reported Qwen3 results are slightly conservative.

Table 8: Greedy threshold sensitivity. SR is the six-task mean of task-wise speedup ratios, averaged over six runs. Underlining marks the selected default.

Model / θ	0.002	0.005	<u>0.01</u>	0.015	0.02
Llama3.1-8B	4.60×	4.79×	4.94×	4.91×	4.78×
DSL-8B	4.21×	4.35×	4.48×	4.47×	4.40×
Vicuna-13B	3.80×	3.99×	4.11×	4.11×	4.04×
Model / θ	0.001	0.002	<u>0.005</u>	0.008	0.01
Qwen3-8B	2.90×	2.93×	2.91×	2.87×	2.80×
Qwen3-32B	2.18×	2.19×	2.19×	2.18×	2.15×

G.2 THRESHOLD SENSITIVITY OF TALON

To test whether the comparison in Table 1 depends on TALON's shared default $\mu = 0.03$, we tune its relative path-score threshold separately for each target model (Liu et al., 2026). At $T = 0$, we sweep $\mu \in \{0.01, 0.02, 0.03, 0.04, 0.05\}$ for Llama3.1-8B and Qwen3-8B, extend the sweep to 0.06 and 0.07 for DSL-8B and Vicuna-13B, and use $\{0.002, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05\}$ for Qwen3-32B. Every setting has six runs on the same 480 prompts. We select the μ with the highest six-run mean per-prompt generation rate at $T = 0$ and reuse it at $T = 1$; the stochastic setting is not tuned independently.

Table 9 reports the full $T = 0$ scan. Increasing μ generally raises acceptance length by retaining a narrower, deeper tree, but throughput eventually saturates or declines. Bold entries mark the setting selected by mean generation rate.

Table 9: TALON threshold sweep under greedy decoding. Each cell reports τ/SR, averaged over six runs; dashes denote settings not evaluated. SR is the six-task mean of task-wise speedup ratios. Bold marks the selected $\mu^\star$.

Model / μ	0.002	0.005	0.01	0.02	0.03	0.04	0.05	0.06	0.07
Llama3.1-8B	–	–	5.80/4.69	6.04/4.80	**6.16/4.85**	6.23/4.84	6.28/4.83	–	–
DSL-8B	–	–	5.49/4.25	5.70/4.34	5.83/4.41	5.90/4.42	**5.97/4.44**	6.00/4.43	6.04/4.39
Vicuna-13B	–	–	6.57/3.88	6.87/3.97	7.05/4.02	7.16/4.03	**7.26/4.03**	7.30/4.02	7.33/4.00
Qwen3-8B	–	–	3.56/2.84	**3.57/2.85**	3.56/2.82	3.54/2.78	3.52/2.75	–	–
Qwen3-32B	2.85/2.13	**2.86/2.13**	2.86/2.12	2.85/2.12	2.85/2.11	2.83/2.09	2.82/2.08	–	–

Table 10 compares tuned TALON with the default Solvo configuration. Tuning selects the original $\mu = 0.03$ for Llama3.1-8B and different values for the other four targets. Solvo achieves higher mean SR on every target: its gain over tuned TALON is 0.83–2.88% at $T = 0$ and 2.54–7.12% at $T = 1$.

Table 10: Comparison with TALON tuned per target model. The selected $\mu^\star$ maximizes the six-run mean generation rate at $T = 0$ and is reused at $T = 1$. SR is the six-task mean of task-wise speedup ratios; entries are means over six runs, and Δ is Solvo's relative SR improvement over tuned TALON.

			TALON		Solvo		
Model	$\mu^\star$	T	τ	SR	τ	SR	Δ
Llama3.1-8B	0.03	0	6.16	4.85×	6.35	**4.94**×	+1.98%
		1	5.12	4.08×	5.50	**4.33**×	+6.21%
DSL-8B	0.05	0	5.97	4.44×	6.01	**4.48**×	+0.83%
		1	5.01	3.77×	5.37	**4.04**×	+7.12%
Vicuna-13B	0.05	0	7.26	4.03×	7.22	**4.11**×	+1.95%
		1	5.99	3.41×	6.30	**3.63**×	+6.54%
Qwen3-8B	0.02	0	3.57	2.85×	3.61	**2.91**×	+1.92%
		1	3.43	2.73×	3.50	**2.80**×	+2.54%
Qwen3-32B	0.005	0	2.86	2.13×	2.92	**2.19**×	+2.88%
		1	2.72	2.03×	2.82	**2.12**×	+4.27%

H LIMITATIONS

Model-dependent hyperparameters. Solvo requires selecting a path-score threshold for each target–draft pair. We use different defaults for the Llama-family/Vicuna and Qwen3 targets rather than a universal setting. Appendix G.1 shows a relatively flat neighborhood around these defaults in the evaluated greedy configurations, but automatic threshold selection remains future work.

Large-batch serving. In large-batch serving, concurrent requests already provide substantial parallelism, and verifying discarded branches competes with useful work on other requests. Solvo is designed for latency-oriented decoding at batch size 1, and we leave high-concurrency serving to future work.

I ARTIFACT LICENSES

Target models. Llama-3.1-8B-Instruct uses the Llama 3.1 Community License. DeepSeek-R1-Distill-Llama-8B is released under MIT, with the underlying Llama 3.1 terms also applying. Vicuna-13B-v1.3 uses a non-commercial license. Qwen3-8B and Qwen3-32B are released under Apache-2.0.

Evaluation data. Alpaca is released under CC BY-NC 4.0; GSM8K and HumanEval under MIT; and Natural Questions under CC BY-SA 3.0. MT-Bench questions are distributed in the Apache-2.0-licensed FastChat repository. The CNN/DailyMail dataset card lists Apache-2.0 for its version-1.0.0 release.

Code. The EAGLE codebase is licensed under Apache-2.0.