TreeGraft: Adaptive Multi-Drafter Grafting for Tree-Based Speculative Decoding

Jiaming Fan[1,2,3*] **Daming Cao**[4*‡] **Canchen Huang**[1,2] **Jiale Fu**[1,2] **Jin Zhang**[3]
Junjie Gao[3] **Kai Yang**[3†] **Xiangzhong Luo**[1,2] **Xu Yang**[1,2]
[1]Key Laboratory of New Generation Artificial Intelligence Technology and
Its Interdisciplinary Applications (Southeast University), Ministry of Education, China
[2]Southeast University [3]Ant Group
[4]Nanjing University of Information Science and Technology
[*]Equal Contribution [†]Project Leader [‡]Corresponding Author
jiaming.fan@seu.edu.cn dmcao@nuist.edu.cn
chengyue.yk@antgroup.com xuyang_palm@seu.edu.cn

Abstract

Speculative decoding accelerates large language model inference through a draft-then-verify paradigm. Building on this, tree-structured methods improve inference by organizing proposals into multiple candidate paths, increasing the accepted length. However, existing tree-structured methods use a single drafter for all drafting steps, creating a dilemma: a smaller drafter is fast but yields lower-quality trees, whereas a larger drafter improves tree quality but suffers from high latency. To address this, we propose *TreeGraft*, a multi-drafter framework in which drafters of different costs jointly construct a shared draft tree. TreeGraft uses the stronger drafter to rescore candidates by updating scores assigned by the weaker drafter, reselect grafting positions, and recover promising paths left unexplored. It also integrates stronger drafter expansions non-destructively, preserving existing branches that may still be accepted by the target model. Together, these designs improve the quality of the shared draft tree. To control the drafting cost, TreeGraft introduces a lightweight scheduler distilled from an offline value system to decide when to call the stronger drafter. Across 10 model pairs and 6 benchmarks, TreeGraft outperforms the better of the two fixed single-drafter endpoint strategies by 15.1% on average, reaching a maximum gain of 26.6%. Our code is available at https://github.com/fjm9933/TreeGraft.

1 Introduction

Autoregressive large language models (LLMs) achieve strong performance [1–3] across a wide range of tasks [4–17], yet their token-by-token decoding makes inference inherently slow. Early speculative decoding (SD) [18, 19] accelerates this process by using a smaller model as the drafter, which constructs drafts that the target model then verifies in parallel. However, since the speedup is bounded by the accepted length, early SD faces a structural limitation: it restricts proposals to a single sequence. Consequently, once the target model rejects a token, all subsequent tokens are invalidated, limiting the accepted length. Tree-based SD [20–22] overcomes this by organizing proposals into a draft tree of multiple paths, utilizing alternative branches to bypass local rejections and thereby extend the accepted length. A long accepted length, however, requires a high-quality tree whose paths cover what the target model would accept, and producing such a tree incurs a substantial drafting cost. The central objective is therefore to build a high-quality tree at a low drafting cost.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

To achieve this objective, we first examine the draft tree construction process. A general paradigm [21] is to build it incrementally over a series of drafting steps. At each step, the drafter forms a candidate pool from the current tree—typically the newly generated nodes from the previous step—and ranks these nodes according to a scoring criterion. The highest-ranked nodes are then expanded with new child nodes. However, existing tree-based methods mostly rely on a single drafter across all drafting steps [20, 21], which forces a binary trade-off between tree quality and per-step cost: a drafter with few parameters yields a lower-quality tree, while simply switching to a larger one raises tree quality but incurs higher latency at every step (Figure 1(a)). This raises a natural question: *must the same drafter be used at every drafting step*? If a stronger drafter were invoked only at the steps where its quality gain matters most, the tree could be improved without paying its cost at all steps.

Motivated by this observation, we introduce two drafters of different costs—a lightweight small drafter and a stronger middle drafter—that jointly build a single draft tree, with each drafter handling a subset of drafting steps. We refer to this scheme as *grafting*: nodes produced by one drafter can be added beneath nodes produced by the other on a shared draft tree. Making grafting work requires answering three questions. Within a single step, *where* on the tree should new nodes be attached, and *how* should they be combined with the existing tree? Across steps, *when* should the middle drafter be called? The first two questions concern how grafting improves tree quality, and the third concerns how to control the drafting cost incurred by middle drafter calls.

Where to graft. In common single-drafter tree construction [21], the candidate pool is restricted to the leaf nodes generated in the previous step. For a single drafter, this rule is natural: earlier candidates have already been evaluated by the same model, so tree construction advances from the newly generated frontier. However, directly applying this to a multi-drafter setting restricts the stronger middle drafter to the newest nodes, forcing it to blindly trust the existing scores on the shared tree. If a promising node was earlier under-ranked by the small drafter, that node is permanently ignored. We therefore expand the candidate pool beyond the previous step at middle drafter steps, allowing the middle drafter to rescore earlier nodes, reselect grafting positions, and therefore reopen prematurely ignored paths (Figure 1(b)).

How to graft. Existing hierarchical multi-drafter methods are largely sequence-based [23, 24]: when a stronger drafter generates new children under a parent, the parent's existing children are overwritten. In a tree, however, overwriting becomes more harmful under our *where* design above: since middle drafter nodes may be attached at earlier positions that already carry entire subtrees, overwriting would discard whole subtrees, including branches that the target model may still accept. We therefore need a non-destructive grafting rule, in which middle drafter nodes are attached beneath existing parents without removing any existing branches (Figure 1(c)).

When to graft. A middle drafter call pays off only at steps where its quality gain justifies the extra latency. Whether a step meets this condition mainly depends on the current tree, which is only known during decoding. We therefore need an online scheduler: at each step, it can read a few signals from the current tree and decide whether a middle drafter call is worth its cost; if so, the step is handed to the middle drafter, otherwise it stays with the small drafter (Figure 1(d)).

Building on this view, we propose *TreeGraft*, a multi-drafter framework for tree-based speculative decoding. TreeGraft focuses on how multiple drafters jointly construct a shared draft tree and schedule its expansions. Our contributions are as follows: (1) **Expanded grafting-position selection**, which lets the middle drafter rescore previously under-ranked nodes and choose grafting positions; (2) **Non-destructive grafting**, which integrates middle drafter child nodes into the shared tree without discarding existing branches; (3) **Value-guided online scheduling**, which fits an offline value system to select value-predictive tree signals and distills its call/skip margin into a lightweight runtime scheduler; (4) Extensive experiments across 10 model pairs and 6 benchmarks, showing that TreeGraft outperforms the better of the two fixed single-drafter endpoint strategies by 15.1% on average, reaching a maximum gain of 26.6%.

2 Related Work

Tree-based speculative decoding. Existing work on tree-based speculative decoding and its draft sources mainly improves candidate-tree quality from two perspectives. The first is tree-side optimization, which improves tree construction [21, 22], node allocation or pruning [25, 26], and token or

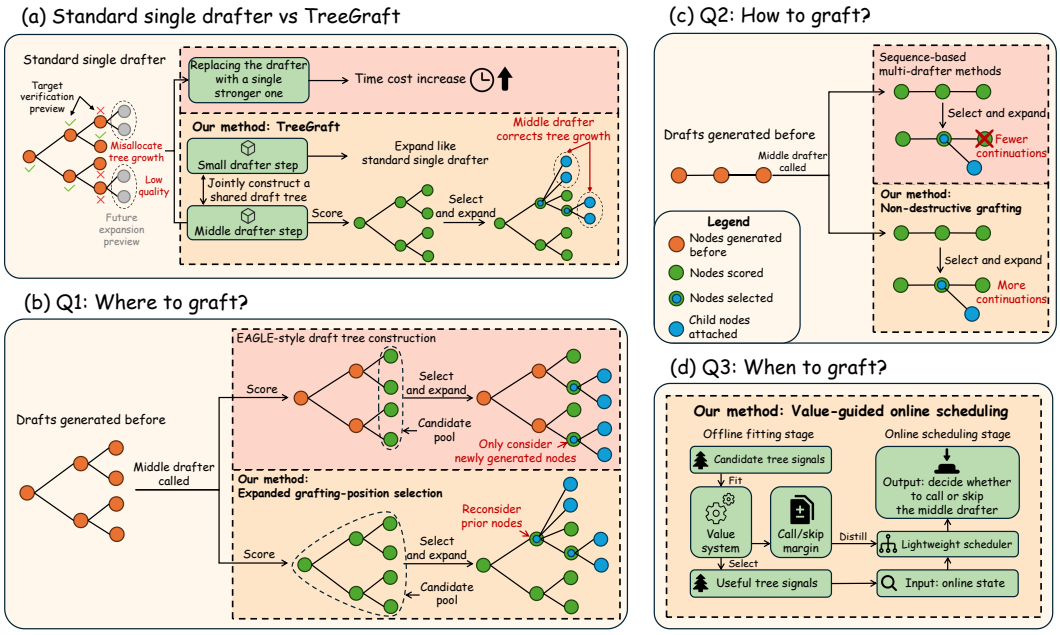

Figure 1: **Overview of TreeGraft.** (a) Standard single-drafter expansion is contrasted with TreeGraft. (b–d) The resulting design is organized around three questions: where to graft, how to graft, and when to graft.

cache-tree verification [20, 27] under a given draft source. The second is draft-source improvement, where representative training-based methods [28–34] improve draft quality by training auxiliary draft modules or draft heads, including target-internal draft heads [31], sequentially dependent draft heads [32], feature-level self-drafting [33], and direct token prediction with multi-layer feature fusion [34]. TreeGraft extends the first direction from single-drafter tree construction to multi-drafter shared-tree construction. At the same time, TreeGraft is largely orthogonal to the second direction, including training-based draft-source improvement, because it operates at the tree-construction level and does not constrain how each draft source is obtained.

Hierarchical multi-drafter speculative decoding. Hierarchical multi-drafter speculative decoding augments the draft–target pipeline with a middle drafter, but existing methods are mainly sequence-based, such as Cascade Speculative Drafting [23] and TriForce [24]. In contrast, TreeGraft studies hierarchical multi-drafter speculative decoding on draft trees.

3 Shared-Tree Construction with Grafting

3.1 Standard Single-Drafter Tree Expansion

Before introducing grafting, we first describe the standard single-drafter expansion rule. At each drafting step, the drafter chooses a candidate pool $\mathcal{C}_k$ from the current draft tree and selects a frontier $\mathcal{F}_k \subseteq \mathcal{C}_k$ for expansion. In common single-drafter tree-SD methods [21], this candidate pool is restricted to the newly generated nodes from the previous step. We denote these nodes by $\mathcal{N}_k$, with $\mathcal{N}_1 = \{r\}$ at the first drafting step, where r is the root representing the current verified prefix. Thus, the standard single-drafter rule uses $\mathcal{C}_k = \mathcal{N}_k$. The drafter then ranks nodes in $\mathcal{C}_k$ using a cumulative path score $s(v)$ and selects the top-W nodes as the expansion frontier, namely $\mathcal{F}_k = \mathrm{Top}_W(\mathcal{C}_k, s)$. Only nodes in $\mathcal{F}_k$ are expanded. For each frontier node, the drafter performs a forward pass on the prefix ending at that node, obtains the next-token distribution, and attaches the top-m tokens as child nodes. These newly attached child nodes form $\mathcal{N}_{k+1}$ for the next step. By sharing prefixes among different paths, the resulting tree can be verified efficiently by the target model through tree attention.

The score $s(v)$ is defined as a cumulative probability along the path to node v. At drafting step k, let $\mathcal{T}_k$ denote the current tree and $V(\mathcal{T}_k)$ its node set. Each non-root node $u \in V(\mathcal{T}_k) \setminus \{r\}$ corresponds to a draft token generated under the prefix ending at its parent $\mathrm{pa}(u)$. When u is generated, the drafter

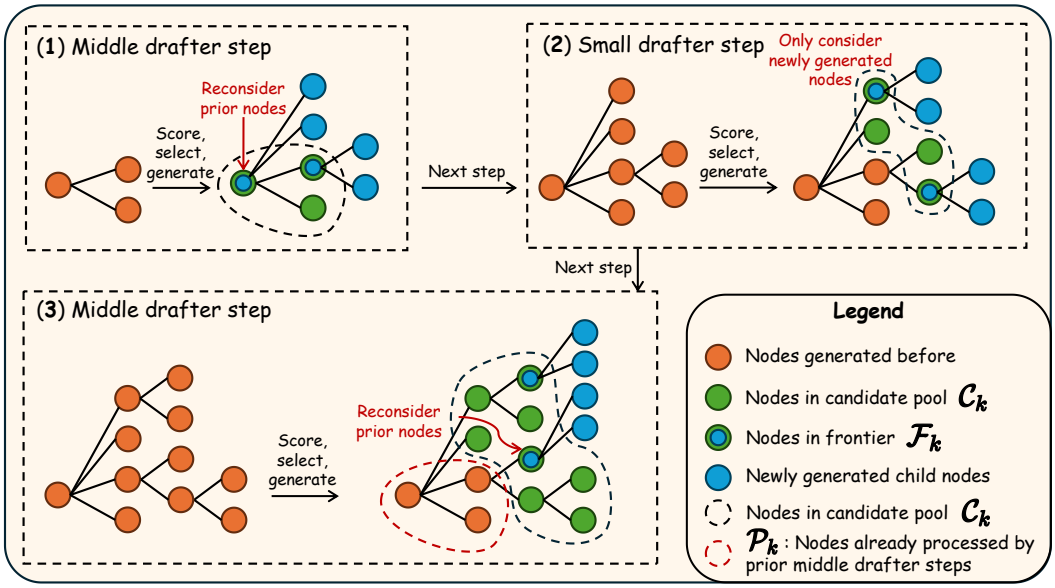

Figure 2: Illustration of expanded grafting-position selection under a middle–small–middle schedule. Panels **(1)** and **(3)** show that middle drafter steps reconsider earlier nodes in the shared tree, while panel **(2)** shows that the small drafter step only expands nodes generated in the previous step.

assigns a probability to this token; we store this probability as the edge score $p(u)$ of the parent-child edge leading to u. For a candidate node v, let $\pi(v)$ denote the nodes on the path from r to v. The cumulative path score is $s(v) = \prod_{u \in \pi(v) \setminus \{r\}} p(u)$. For the root node, $s(r) = 1$.

3.2 Grafting on a Shared Draft Tree

The expansion process described in Section 3.1 assumes a single drafter. When multiple drafters with varying capabilities collaborate, they iteratively build upon a shared draft tree. We frame this multi-drafter collaborative expansion as a *grafting* process. To formalize this operation, we must first address two critical questions: *where* to graft and *how* to graft, which together determine how TreeGraft improves tree quality.

Expanded grafting-position selection. To decide where to graft, TreeGraft looks at which nodes have been considered by the active drafter. We say that a drafter has *visited* a node if the node has entered that drafter's candidate pool in a previous step. This definition recovers the standard rule in the single-drafter case: since the same drafter has processed all earlier candidate pools, the only nodes *unvisited* by that drafter are the nodes generated in the preceding step.

In TreeGraft, the two drafters are treated asymmetrically. The middle drafter follows the same *visited*-node rule. Let $\mathcal{P}_k$ denote the set of nodes that have already entered a middle drafter candidate pool before step k, with $\mathcal{P}_1 = \emptyset$. At a middle drafter step, TreeGraft uses $\mathcal{C}_k = V(\mathcal{T}_k) \setminus \mathcal{P}_k$. Thus, the middle drafter can reconsider any node in the current shared tree that it has not yet *visited*, including historical nodes produced or ranked by the small drafter. Since the small drafter has lower capacity than the middle drafter, TreeGraft does not use it to *revisit* older nodes for re-ranking. Instead, the small drafter treats all previous candidate pools, including those from the middle drafter, as already evaluated. Therefore, at a small drafter step, we set its candidate pool exclusively to the newly generated nodes from the previous step, i.e., $\mathcal{C}_k = \mathcal{N}_k$. After a middle drafter step, we update $\mathcal{P}_{k+1} = \mathcal{P}_k \cup \mathcal{C}_k$; after a small drafter step, $\mathcal{P}_{k+1} = \mathcal{P}_k$. Combining the two cases gives

$$\mathcal{C}_k = \begin{cases} \mathcal{N}_k, & \text{small drafter step,} \\ V(\mathcal{T}_k) \setminus \mathcal{P}_k, & \text{middle drafter step.} \end{cases} \tag{1}$$

Figure 2 illustrates this rule under a middle–small–middle schedule. Each middle drafter step considers the portion of the shared tree that has not yet been *visited* by the middle drafter, while the

small drafter expands only the most recently generated nodes. As a result, the middle drafter can select historical grafting positions and correct earlier tree-growth decisions.

After the candidate pool is formed, we enforce the same asymmetry on scoring: the small drafter trusts and directly reuses the stored scores, whereas the middle drafter updates candidate scores using its own distribution. For each non-root candidate node $v \in \mathcal{C}_k$, the middle drafter updates the edge score $p(v)$ assigned to the token at v, and then recomputes the cumulative path score $s(v)$. The selection of the expansion frontier and the generation of child nodes then follow the same procedure as in Section 3.1.

Non-destructive grafting. Once the frontier $\mathcal{F}_k$ is selected, the drafter generates child nodes for each frontier node. A replacement rule is natural in sequence-based hierarchical drafting [23, 24], where the stronger drafter refines a single continuation. In a shared tree, however, a selected historical node may already have children and descendants. Replacing them would remove existing subtrees before target verification and may reduce the accepted length (see Appendix F). TreeGraft therefore attaches newly generated children under the frontier nodes without overwriting any existing node or branch.

Non-destructive grafting may temporarily make the constructed tree larger, but the final verification tree is pruned to the same verification budget B_{ver}. Once construction is complete, we prune the whole constructed tree before target verification. The pruning step ranks non-root nodes by cumulative path score. It then greedily keeps high-scoring nodes together with the ancestor chains needed to connect them to the root, until the verification budget is reached. Thus, old branches are not discarded during construction, while the final tree remains a valid budgeted tree for target-side verification.

Distribution preservation. TreeGraft changes only the proposal tree supplied to the target model. The final output is still produced by the same target-side tree verification and rejection-sampling rule as in common single-drafter tree-SD methods. Therefore, TreeGraft improves proposal-construction efficiency without changing the target-model output distribution. Appendix E gives the detailed pseudocode for standard single-drafter tree construction and TreeGraft shared-tree construction.

4 Online Scheduling via Value-Guided Distillation

Section 3 addresses where and how to graft middle drafter nodes onto the shared draft tree. We now address when to call the middle drafter. The value of a middle drafter call is state-dependent: in some tree states, the middle drafter can correct poor growth and increase the accepted length; in others, the same call adds latency without enough quality gain. TreeGraft therefore learns an online scheduler that maps the current online state to a call/skip decision. To obtain this scheduler, we first define the scheduling target, then fit an offline value system that connects trajectory-level outcomes to step-wise decisions, and finally distill its call/skip comparison into a lightweight runtime predictor, as summarized in Figure 1(d).

4.1 Scheduling Target and Observability Gap

A middle drafter call is useful only if the accepted-length gain it brings is worth the extra latency it adds. This makes throughput a natural objective for scheduling. The difficulty is that, when the scheduler makes a step-wise decision, this throughput is not yet available. It is observed only after the remaining tree construction and target-side verification are completed. Thus, the supervision for a step is tied to an entire scheduling trajectory rather than to an isolated local decision. To obtain step-wise supervision despite this observability gap, we fit an offline value system over complete trajectories and then distill its scheduling preference into a lightweight online scheduler.

Formally, turn-level throughput is written as $\eta = A/t$, where A denotes the accepted length and $t = t_{\text{D}} + t_{\text{V}}$ is the total latency of one speculative-decoding turn, consisting of draft-construction latency t_{D} and target-verification latency t_{V}. At drafting step k, the scheduler observes an online state z_k and chooses a call/skip action $a_k \in \{0, 1\}$: $a_k = 1$ calls the middle drafter, while $a_k = 0$ leaves the step to the small drafter. The desired scheduler chooses the action with the larger predicted turn-level throughput.

4.2 Fitting an Offline Value System

Since the final throughput depends on the entire sequence of scheduling decisions, we define a scheduling trajectory as the binary call/skip sequence $\tau_{1:D} \in \{0,1\}^D$, where D is the maximum number of drafting steps and τ_k denotes the action taken at step k. For a step k, this trajectory can be decomposed into the past scheduling prefix $\tau_{1:k-1}$, the current action $a_k = \tau_k$, and the future suffix $\tau_{k+1:D}$. During offline data generation, we evaluate complete call/skip trajectories by executing the TreeGraft construction and target verification; Appendix B.1 gives the data-collection details. Rather than compressing these evaluated outcomes into a single ratio $\eta = A/t$, we independently record the trajectory-level accepted length $A(\tau_{1:D})$ and latency $t(\tau_{1:D})$ as separate labels to provide more fine-grained supervision. To obtain step-wise supervision from these trajectory-level labels, we associate each complete-trajectory outcome with the pre-action state, current action, and future suffix observed at a step. Each fitting sample has the form $(z_k, a_k, \tau_{k+1:D}, A, t)$, where A and t are the complete-trajectory outcome labels paired with that step. We implement the value system f_θ as a lightweight MLP that predicts accepted length and latency from the current online state, the current action, and the future suffix, with architecture and training details in Appendix B.3:

$$(\hat{A}, \hat{t}) = f_\theta(z_k, a_k, \tau_{k+1:D}), \qquad \hat{\eta} = \hat{A}/\hat{t}. \tag{2}$$

Including the suffix is important: the same current state and current action can lead to different turn-level outcomes under different future schedules. Thus, f_θ serves as a future-aware estimator of the trajectory-level outcome induced by a local scheduling decision and its subsequent suffix.

The online state is written as $z_k = (\mathbf{c}, k, \tau_{1:k-1}, \phi_k)$, where $\mathbf{c}$ is the runtime context, k is the drafting-step index, $\tau_{1:k-1}$ is the previous call/skip history, and ϕ_k contains tree signals computed from the current draft tree before taking action. The first three terms provide basic runtime and scheduling information, but they are insufficient to distinguish different tree states. Therefore, we additionally incorporate the tree signals ϕ_k, which expose the current tree state to the value system.

Since the offline fitting samples are constructed from a limited set of models and datasets, directly using all candidate tree signals can introduce extra noise and reduce fitting accuracy. Not all tree signals ϕ_k are equally useful: some improve predictive accuracy, while others are redundant or noisy. To prevent this, we collect ten tree signals into a candidate set Φ_k and enumerate candidate subsets $\phi_k \subseteq \Phi_k$; Appendix B.2 defines the candidate signals. We combine each subset with the basic signals to formulate z_k, and fit the value system f_θ. We select the subset $\phi_k^\star$ with the lowest out-of-fold identification score (detailed in Appendix B.3) as the final tree signals. This gives the final online state $z_k^\star$ and the corresponding fitted value system $f_{\theta^\star}$. With the selected state representation and fitted value system in place, the remaining question is how to convert this offline estimator into a runtime scheduling rule.

4.3 Margin Distillation for Runtime Scheduling

One possible scheduler would query the fitted value system online for both actions and choose the larger predicted throughput; Appendix B.6 ablates this direct value-system planning alternative. We avoid this direct use for two reasons. First, the value system predicts the absolute accepted length and latency, while scheduling only needs the relative preference between call and skip. Small absolute prediction errors can therefore flip the action ordering. Second, the value system conditions on future suffixes, but future decisions are not known during online decoding.

TreeGraft instead uses the fitted value system as an offline teacher that converts trajectory-level outcomes into a call/skip margin. For each selected online state $z_k^\star$, we compare call and skip after optimizing over all binary future suffixes $\tau_{k+1:D} \in \{0,1\}^{D-k}$, as detailed in Appendix B.4:

$$y_k = \max_{\tau_{k+1:D}} \hat{\eta}(z_k^\star, 1, \tau_{k+1:D}) - \max_{\tau_{k+1:D}} \hat{\eta}(z_k^\star, 0, \tau_{k+1:D}). \tag{3}$$

The scalar y_k is a call/skip margin. A positive margin means that calling the middle drafter is predicted to yield higher future-optimized throughput; a negative margin means that skipping is preferred. Since this target encodes the relative preference between call and skip at the same online state, it can provide a more stable supervision signal than directly regressing complete-trajectory throughput labels. We train a lightweight predictor $g_\psi(z_k^\star)$ to regress this margin. Unlike the value system, g_ψ does not take the future suffix as input; the suffix-dependent reasoning has already been

absorbed into the offline margin target. During inference, the scheduler uses only the sign of the predicted margin:

$$\pi_\psi(z_k^\star) = \begin{cases} 1 \text{ (call)}, & g_\psi(z_k^\star) > 0, \\ 0 \text{ (skip)}, & \text{otherwise.} \end{cases} \tag{4}$$

Thus, online scheduling requires a single lightweight forward pass at each drafting step, while the expensive future-aware comparison is performed offline. Offline results of scheduler distillation are provided in Appendix B.5. The runtime overhead of the scheduler is analyzed in Appendix C.2.

5 Experiments

5.1 Experimental Setup

Model pairs. We evaluate TreeGraft with off-the-shelf drafters in a setting that requires *no additional drafter training*. The target model and the middle drafter are pretrained checkpoints from LLaMA 3 [35] and Qwen3 [36]. *For the small drafter, we use a training-free n-gram drafter.* This choice is based on our preliminary experiments (Appendix C.1): even the smallest neural LLM in each model family introduces nontrivial forwarding cost in our implementation, making it unsuitable as the low-cost small drafter. Among the preliminary configurations we tested, using a training-free n-gram drafter as the small drafter and a pretrained neural model as the middle drafter achieves the best speedup, while replacing the n-gram small drafter with a pretrained neural small model leads to lower speedup (Appendix G.1). The offline stage described in Section 4 (state selection, value-system fitting, and scheduler training) uses 6 target–middle model pairs. In the online evaluation of the scheduler, we report results not only on these 6 seen pairs (white rows in Table 1), but also on 4 held-out pairs that are unseen during the offline stage (blue rows in Table 1), to verify the generalization.

Datasets. We evaluate on 6 benchmarks: GSM8K [37], Alpaca [38], NQ [39], HumanEval [40], CNN/DailyMail (CNN/DM) [41], and MT-Bench [42]. For the first five benchmarks, the offline stage and online evaluation use disjoint subsets, so no evaluation sample is used during offline fitting. MT-Bench, shown as the green columns in Table 1, is entirely held out from the offline stage and used only for cross-task generalization.

Baselines and metrics. Under the standard dynamic-tree construction paradigm in tree-based speculative decoding [21], we construct two single-drafter endpoint baselines: All Small uses the small drafter throughout tree construction, while All Mid uses the middle drafter throughout. Comparing TreeGraft with these baselines tests whether selectively using multiple drafters to expand a shared tree improves over common single-drafter dynamic-tree construction. For fair comparison, all methods use the same tree budget, decoding parameters, and target verification procedure. In Section 5.3, we further compare with Cascade Speculative Drafting (CSD) [23], a sequence-based multi-drafter baseline, and the standard dynamic-tree candidate-pool variant [21] adapted to the multi-drafter setting. We report the speedup ratio relative to standard autoregressive decoding with the target model. Section 5.3 additionally reports the average accepted length A.

Runtime configuration. All online evaluations share the same tree-construction hyperparameters: maximum number of drafting steps $D{=}5$, expansion frontier size $W{=}10$, and $m{=}10$ child tokens per expanded node. Additional decoding parameters are provided in Appendix D.

5.2 Main Results

Overall performance. Table 1 compares TreeGraft with All Small and All Mid, the two single-drafter dynamic-tree endpoints. Across 10 model pairs and 6 datasets, TreeGraft achieves an average speedup of $1.60\times$, compared with $1.32\times$ for All Small and $1.39\times$ for All Mid. This represents a 15.1% improvement in average speedup over All Mid, the stronger of the two fixed endpoint strategies by overall average speedup. The largest per-setting gain over the stronger of the two endpoints is 26.6%, attained on Qwen3-32B/0.6B with Alpaca ($1.38\times$ vs. $1.09\times$). Compared with the stronger endpoint selected separately for each setting, TreeGraft performs better in 46 of the 60 model-pair–benchmark settings, ties in two, and performs worse in 12. In these 12 settings, its mean relative shortfall is 1.8%. Together, these results support the benefit of TreeGraft's multi-drafter shared-tree construction over the two single-drafter dynamic-tree endpoints in this controlled comparison. Figure 3 visualizes the gains over each endpoint, and Table 11 in Appendix G.3 reports the complete results. Timing stability is examined in Appendix G.2.

Table 1: Online evaluation results. Values are speedups over autoregressive target decoding. TG denotes TreeGraft; S and M denote All Small and All Mid. The five-task average is the arithmetic mean over Alpaca, GSM8K, HumanEval, NQ, and CNN/DM; the overall average includes MT-Bench. Blue rows denote held-out target–middle pairs, green columns denote the held-out MT-Bench task, and orange cells denote their intersection. For LLaMA pairs, 70B, 8B, 3B, and 1B refer to LLaMA 3.3-70B, LLaMA 3.1-8B, LLaMA 3.2-3B, and LLaMA 3.2-1B, respectively. The full speedups are in Table 11.

Target–Middle	Five-task average			MT-Bench			Overall average			Improvement	
	TG	S	M	TG	S	M	TG	S	M	TG vs S	TG vs M
Qwen3-32B/1.7B	**1.49**	1.20	1.27	**1.51**	1.26	1.21	**1.49**	1.21	1.26	23.1%	18.3%
Qwen3-32B/0.6B	**1.47**	1.20	1.19	**1.46**	1.26	1.18	**1.47**	1.21	1.19	21.5%	23.5%
Qwen3-32B/8B	**1.39**	1.20	1.06	**1.43**	1.26	1.08	**1.40**	1.21	1.06	15.7%	32.1%
LLaMA-70B/1B	2.30	1.44	**2.32**	**2.26**	1.44	2.21	**2.30**	1.44	**2.30**	59.7%	0.0%
LLaMA-70B/3B	1.95	1.44	**1.96**	**1.91**	1.44	**1.91**	**1.95**	1.44	**1.95**	35.4%	0.0%
LLaMA-8B/1B	**1.51**	1.40	1.31	**1.47**	1.40	1.26	**1.50**	1.40	1.30	7.1%	15.4%
Qwen3-32B/4B	**1.38**	1.20	1.15	**1.41**	1.26	1.15	**1.39**	1.21	1.15	14.9%	20.9%
Qwen3-8B/0.6B	**1.26**	1.23	0.85	**1.29**	1.28	0.84	**1.27**	1.24	0.85	2.4%	49.4%
LLaMA-70B/8B	1.83	1.44	**1.87**	**1.83**	1.44	1.81	1.83	1.44	**1.86**	27.1%	-1.6%
LLaMA-8B/3B	**1.42**	1.40	0.94	**1.42**	1.40	0.94	**1.42**	1.40	0.94	1.4%	51.1%
Average	**1.60**	1.31	1.39	**1.60**	1.34	1.36	**1.60**	1.32	1.39	21.2%	15.1%

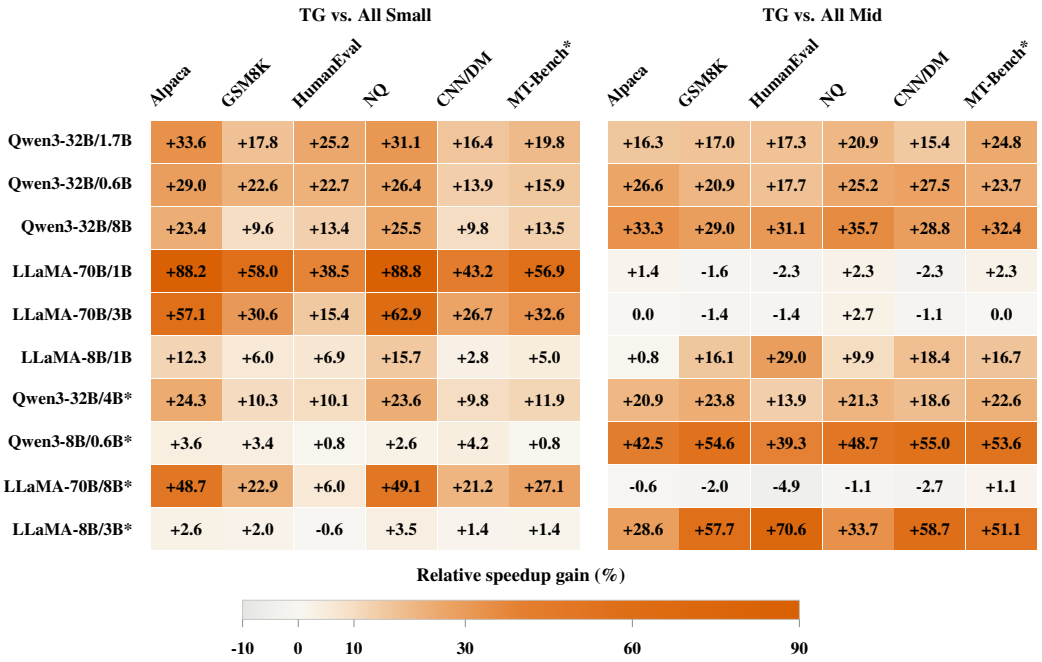

Figure 3: Setting-wise percentage speedup gains of TreeGraft over All Small (left) and All Mid (right). Cell labels show gains computed from the reported two-decimal speedups. An asterisk (*) marks held-out model pairs and MT-Bench.

Generalization. TreeGraft maintains speedup on unseen model pairs and tasks. The 4 held-out model pairs (blue rows in Table 1) achieve an average speedup of $1.48\times$, higher than All Small ($1.32\times$) and All Mid ($1.20\times$). On MT-Bench, shown as the green columns in Table 1 and excluded from the offline stage, TreeGraft achieves an average speedup of $1.60\times$, higher than All Small ($1.34\times$) and All Mid ($1.36\times$). These results suggest that the scheduler captures cost–quality trade-off patterns that transfer across model pairs and tasks, rather than being tied only to the fitting configurations. We also evaluate scheduler transfer without refitting across sampling temperature ($T = 0 \rightarrow 1$), tree width, model family, and hardware (Appendix G.4).

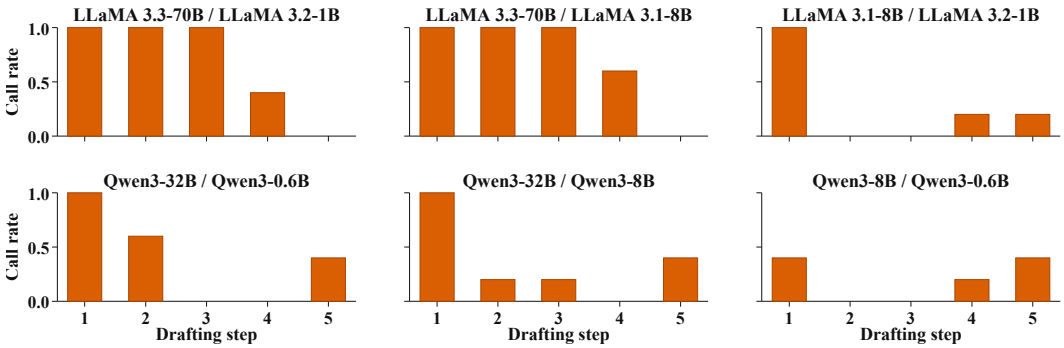

Figure 4: Step-wise middle drafter usage learned by the TreeGraft online scheduler under the value system setting. Each subplot corresponds to one target–middle model pair. The y-axis shows the middle drafter call rate at each drafting step, averaged over the five fit datasets. Higher values indicate a stronger tendency to call the middle drafter at that step.

Scheduling behavior analysis. At the model-pair-average level, TreeGraft improves over the better endpoint by up to 21.5% (Qwen3-32B/0.6B, TreeGraft $1.47\times$ vs. All Small $1.21\times$), while falling behind the better endpoint by at most 1.6% (LLaMA 3.3-70B/3.1-8B, TreeGraft $1.83\times$ vs. All Mid $1.86\times$). The former indicates that the scheduler effectively captures gains when improvement is available; the latter indicates that the scheduler introduces almost no loss when improvement is unavailable. Together, these results indicate that the scheduler adaptively finds a favorable cost–quality trade-off across settings, thereby achieving the goal of building high-quality trees at low cost. The following analysis explains this adaptive behavior through three representative regimes:

When All Mid significantly outperforms All Small (e.g., LLaMA 3.3-70B/3.2-1B, All Mid $2.30\times$ vs. All Small $1.44\times$), TreeGraft achieves $2.30\times$ speedup, closely tracking the All Mid endpoint. As shown in Figure 4, the learned scheduler calls the middle drafter frequently for most drafting steps in this regime, yielding a call pattern close to All Mid.

When All Small significantly outperforms All Mid (e.g., Qwen3-8B/0.6B, All Small $1.24\times$ vs. All Mid $0.85\times$), TreeGraft achieves $1.27\times$ speedup, slightly exceeding All Small while avoiding the severe slowdown of All Mid. In this case, Figure 4 shows that the scheduler uses the middle drafter sparsely, keeping the call rate much lower than an All Mid policy.

When the two endpoints achieve similar and modest speedups (e.g., Qwen3-32B/0.6B, All Small $1.21\times$ vs. All Mid $1.19\times$), TreeGraft achieves $1.47\times$ speedup, substantially surpassing both endpoints. Here, Figure 4 shows a selective call pattern: middle drafter calls are concentrated on selected steps rather than applied uniformly.

Overall, these call-rate patterns are consistent with the intended adaptive cost–quality trade-off: the scheduler adjusts middle-drafter usage across model pairs and drafting steps, increasing calls when stronger drafting is more beneficial and reducing them when its cost is less justified.

5.3 Alternative Designs and Ablations

Table 2: Ablation of grafting mechanism components. All variants use the same fixed middle drafter schedule $[1, 0, 1, 0, 1]$, i.e., the middle drafter participates only at drafting steps 1, 3, and 5. Speedup is relative to autoregressive decoding with the target model. A denotes average accepted length.

Method	Q32B/0.6B GSM8K		Q32B/0.6B CNN/DM		L70B/1B GSM8K		L70B/1B CNN/DM		Average	
	Speedup (×)	A	Speedup (×)	A	Speedup (×)	A	Speedup (×)	A	Speedup (×)	A
CSD	0.75	0.82	0.78	0.72	1.07	0.90	1.14	0.78	0.94	0.81
w/o EG	1.49	1.76	0.85	1.56	1.72	1.92	1.34	1.97	1.35	1.80
w/o ND	1.44	1.69	0.82	1.18	1.33	1.64	1.21	1.37	1.20	1.47
TreeGraft	**1.55**	**1.84**	**0.95**	**2.15**	**1.80**	**2.18**	**1.42**	**2.52**	**1.43**	**2.17**

This section ablates TreeGraft's three components: expanded grafting-position selection and non-destructive grafting (how middle drafter outputs are integrated) and the online scheduler (which steps invoke the middle drafter). We ablate the first two components under a fixed schedule, and then assess the scheduler by comparing the fixed-schedule results with the learned-scheduler results in Table 1.

Setup. All variants share a fixed middle drafter schedule $[1, 0, 1, 0, 1]$, where the middle drafter participates at steps 1, 3, and 5, so grafting is tested after small drafter expansion at multiple tree depths. We use two model pairs (Qwen3-32B/0.6B, LLaMA 3.3-70B/3.2-1B), each selected from a representative regime where middle drafter calls are effective (Section 5.2). Experiments are conducted on GSM8K and CNN/DM. The four methods are: **Cascade Speculative Drafting (CSD)** [23], a sequence-based multi-drafter baseline reproduced under the same fixed middle drafter schedule; **w/o EG** (w/o expanded grafting-position selection), which retains non-destructive grafting but restricts the middle drafter candidate pool to the nodes generated in the previous step (this structure effectively implements the EAGLE candidate-pool design within a multi-drafter setting); **w/o ND** (w/o non-destructive grafting), which retains expanded grafting-position selection but overwrites existing children when attaching middle drafter nodes; and the full **TreeGraft**.

Table 2 reports the speedup and average accepted length (tokens accepted per verification round). TreeGraft achieves the best results across all settings.

Tree-based grafting outperforms CSD. Under the same fixed middle drafter schedule, replacing TreeGraft with CSD reduces the overall speedup from $1.43\times$ to $0.94\times$ and the average accepted length from 2.17 to 0.81, showing that shared-tree grafting is more effective than sequence-based multi-drafter methods when call positions are controlled.

Expanded grafting-position selection provides an additional, independent gain. Removing it (w/o EG) reduces the overall speedup from $1.43\times$ to $1.35\times$ and the average accepted length from 2.17 to 1.80. To check whether this gain indeed comes from revisiting earlier nodes, we report detailed statistics at middle drafter steps: TreeGraft's candidate pool averages 112.54 nodes ($2.38\times$ that of w/o EG), and 61.47% of its frontier selections fall on historical positions (vs. 0% for w/o EG). Since both methods select the same top-W nodes per step, the accepted-length improvement stems from better positions, not more selections.

Non-destructive grafting provides the primary gain. Removing it (w/o ND) reduces the overall speedup from $1.43\times$ to $1.20\times$ and the average accepted length from 2.17 to 1.47, showing that overwriting existing branches can remove useful paths and thus reduce target acceptance.

Contribution of the online scheduler. Comparing the fixed schedule with the same settings in Table 1, the online scheduler raises the average speedup from $1.43\times$ to $1.94\times$. The gap is sharpest on CNN/DM, where it improves Qwen from $0.95\times$ to $1.39\times$ and LLaMA from $1.42\times$ to $2.09\times$, confirming that adaptive scheduling provides a gain beyond the grafting mechanism itself.

Scheduler-state ablation. Removing runtime context, scheduling history, or tree signals separately reduces performance, showing that call cost, earlier drafting choices, and current frontier structure each inform the decision. In particular, tree signals contribute information beyond timing and call history (Appendix G.5).

Heuristic scheduler comparison. We test simple policies based on step position, confidence, score gap, or latency, tuning each over a predefined parameter grid. The learned scheduler attains the highest average speedup among these tested policies (Appendix G.6).

6 Conclusion

This paper introduced TreeGraft, a multi-drafter framework for tree-based speculative decoding organized around three questions: *where* to graft, *how* to graft, and *when* to graft. Expanded grafting-position selection and non-destructive grafting address the first two questions by improving tree quality, while value-guided online scheduling addresses the third by controlling drafting cost. Together, these components turn shared-tree grafting into a controlled quality–cost trade-off. Experiments show that TreeGraft outperforms both single-drafter endpoints on average and closely tracks the better endpoint when one endpoint dominates, while ablations confirm that the components contribute complementary gains. We discuss limitations and future directions in Appendix A.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 62301266), the National Natural Science Foundation of China (Grant No. 62576091), and the Jiangsu Province Carbon Peak Carbon Neutrality Science and Technology Innovation Special Fund Project (Grant No. BT2025029). We also acknowledge support from the Southeast University Big Data Computing Center and the Southeast University Kunpeng & Ascend Center of Cultivation.

Competing Interests. The authors declare no competing interests.

References

[1] Yingzhe Peng, Gongrui Zhang, Miaosen Zhang, Zhiyuan You, Jie Liu, Qipeng Zhu, Kai Yang, Xingzhong Xu, Xin Geng, and Xu Yang. Lmm-r1: Empowering 3b lmms with strong reasoning abilities through two-stage rule-based rl, 2025. URL https://arxiv.org/abs/2503.07536.

[2] Li Li, Ziyi Wang, Yongliang Wu, Jianfei Cai, and Xu Yang. Cot vectors: Transferring and probing the reasoning mechanisms of llms, 2025. URL https://arxiv.org/abs/2510.00579.

[3] Ziyi Wang, Li Li, Aolin Zhou, Yankun Shen, Chonghan Liu, Shuxia Lin, and Xu Yang. Can linguistic reasoning vectors enhance multimodal reasoning ability?, 2026. URL https://arxiv.org/abs/2609.31140.

[4] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[6] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[7] Jian-Dong Liu, Zi-Chen Zhao, Hao Sun, Lin-Xing Wu, Huan Zhang, Pengyuan Wang, Ming Zhao, Xinyu Chu, Shu Yan, Yongbei Zhu, Weijun Zhong, Zhi-Hao Tan, Jing Shang, Yang Yu, and Zhi-Hua Zhou. Constructive specification for plug-and-play learnware agents. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, page 3222–3233. ACM, August 2026. doi: 10.1145/3770855.3817996. URL http://dx.doi.org/10.1145/3770855.3817996.

[8] Guannan Lai, Gelin Bian, Hao-Xuan Ma, Jun-Peng Jiang, Long Chen, Jian-Dong Liu, Zhi-Hao Tan, and Han-Jia Ye. Routing should pay for itself: Sparse supervision for economical llm routing, 2026. URL https://arxiv.org/abs/2609.37402.

[9] Jiale Fu, Yuchu Jiang, Peijun Wu, Chonghan Liu, Joey Tianyi Zhou, and Xu Yang. Rethinking llm ensembling from the perspective of mixture models, 2026. URL https://arxiv.org/abs/2605.00419.

[10] Shuo Xu, Yu Chen, Shuxia Lin, Xin Geng, and Xu Yang. Flowprune: Accelerating attention flow calculation by pruning flow network. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 134236–134261. Curran Associates, Inc., 2025. doi: 10.52202/085713-4475. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/c2cd7fa98591516c025d7632b2586ca1-Paper-Conference.pdf.

[11] Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, and Xu Yang. d^2cache: Accelerating diffusion-based llms via dual adaptive caching, 2026. URL https://arxiv.org/abs/2509.23094.

[12] Jiale Fu, Yuchu Jiang, Junkai Chen, Jiaming Fan, Xin Geng, and Xu Yang. Fast large language model collaborative decoding via speculation, 2025. URL https://arxiv.org/abs/2502.01662.

[13] Huiyi Chen, Jiawei Peng, Kaihua Tang, Xin Geng, and Xu Yang. Enhancing multimodal in-context learning for image classification through coreset optimization, 2025. URL https://arxiv.org/abs/2504.14200.

[14] Yuchu Jiang, Jiale Fu, Chenduo Hao, Xinting Hu, Yingzhe Peng, Xin Geng, and Xu Yang. Mimic in-context learning for multimodal tasks, 2025. URL https://arxiv.org/abs/2504.08851.

[15] Yingzhe Peng, Chenduo Hao, Xu Yang, Jiawei Peng, Xinting Hu, and Xin Geng. Live: Learnable in-context vector for visual question answering, 2024. URL https://arxiv.org/abs/2406.13185.

[16] Xu Yang, Yingzhe Peng, Haoxuan Ma, Shuo Xu, Chi Zhang, Yucheng Han, and Hanwang Zhang. Lever lm: Configuring in-context sequence to lever large vision language models, 2024. URL https://arxiv.org/abs/2312.10104.

[17] Li Li, Jiawei Peng, Huiyi Chen, Chongyang Gao, and Xu Yang. How to configure good in-context sequence for visual question answering, 2023. URL https://arxiv.org/abs/2312.01571.

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

[19] 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.

[20] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 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*, pages 932–949, 2024.

[21] 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*, pages 7421–7432, 2024.

[22] Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. *Advances in Neural Information Processing Systems*, 37:129531–129563, 2024.

[23] Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Kevin C Chang, and Jie Huang. Cascade speculative drafting for even faster llm inference. *Advances in Neural Information Processing Systems*, 37:86226–86242, 2024.

[24] Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024.

[25] 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.

[26] 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.

[27] 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. *Advances in Neural Information Processing Systems*, 37:16342–16368, 2024.

[28] Zihua Wang, Ruibo Li, Haozhe Du, Joey Tianyi Zhou, Yu Zhang, and Xu Yang. Specflash: A latent-guided semi-autoregressive speculative decoding framework for efficient multimodal generation, 2026. URL `https://arxiv.org/abs/2505.12728`.

[29] Kaiwen Liu, Yangkai Xie, Shuxia Lin, Chonghan Liu, Xu Yang, and Yiguo Qiao. CATS: Acceptance-oriented critical token adaptive selection for multimodal speculative decoding. Accepted at NeurIPS, 2026. URL `https://neurips.cc/virtual/2026/loc/sydney/poster/154883`.

[30] Jiaming Fan, Daming Cao, Xiangzhong Luo, Jiale Fu, Chonghan Liu, and Xu Yang. Flatter tokens are more valuable for speculative draft model training, 2026. URL `https://arxiv.org/abs/2601.18902`.

[31] 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. *arXiv preprint arXiv:2401.10774*, 2024.

[32] Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. *arXiv preprint arXiv:2402.05109*, 2024.

[33] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

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

[35] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[36] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[37] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[38] 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, 2023.

[39] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

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

[41] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015.

[42] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[43] Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026.

A Limitations

TreeGraft is drafter training-free and does not train additional draft heads or auxiliary draft modules. This makes it easy to apply with off-the-shelf draft sources, but also means that it does not replace training-based draft-source improvement methods, such as EAGLE-style draft heads [34]. As discussed in Section 2, these methods are largely orthogonal to TreeGraft, since TreeGraft operates at the shared-tree construction and scheduling level. Combining TreeGraft with trained draft sources is therefore a natural direction for future work.

In addition, because the smallest neural LLMs in our implementation still introduce nontrivial forwarding latency, we use an n-gram drafter as the low-cost small drafter. This lookup-based drafter is one practical instantiation rather than an exhaustive choice, and future work can explore stronger lightweight draft sources, such as Engram-style conditional-memory lookup [43] or suffix-tree-based drafters, within the same TreeGraft framework.

B Details of Online Scheduling

This appendix provides the technical implementation details, hyperparameters, network architectures, candidate signals, and offline data construction procedure for the online scheduling method introduced in Section 4.

B.1 Fitting Data Collection

The offline fitting data is collected by executing complete scheduling trajectories. We set the maximum number of drafting steps to $D = 5$. A complete scheduling trajectory is denoted by

$$\tau_{1:D} \in \{0, 1\}^D,$$

where $\tau_k = 1$ means that the middle drafter is called at the k-th drafting step, and $\tau_k = 0$ means that the middle drafter is skipped.

Offline reports. The offline stage uses 6 training target–middle model pairs:

LLaMA 3.3-70B-Instruct / LLaMA 3.2-1B-Instruct,
LLaMA 3.3-70B-Instruct / LLaMA 3.2-3B-Instruct,
LLaMA 3.1-8B-Instruct / LLaMA 3.2-1B-Instruct,
Qwen3-32B / Qwen3-0.6B, Qwen3-32B / Qwen3-8B, Qwen3-32B / Qwen3-1.7B.

It uses 5 fitting datasets:

`Alpaca_fit`, `GSM8K_fit`, `HumanEval_fit`, `NQ_fit`, `CNN/DM_fit`.

This gives $6 \times 5 = 30$ fitting reports. The online evaluation uses disjoint evaluation subsets for the first five datasets, and additionally includes MT-Bench as a fully held-out task. We also evaluate on 4 held-out target–middle model pairs that are not used in the offline stage:

LLaMA 3.3-70B-Instruct / LLaMA 3.1-8B-Instruct,
LLaMA 3.1-8B-Instruct / LLaMA 3.2-3B-Instruct,
Qwen3-8B / Qwen3-0.6B, Qwen3-32B / Qwen3-4B.

Model-family coverage. The offline value system and the distilled online scheduler are evaluated across two representative mainstream open-source model families, LLaMA 3 and Qwen3, covering multiple target–middle scale combinations. Importantly, the online evaluation includes both model pairs used during offline fitting and held-out target–middle pairs that are never used for value-system fitting or scheduler distillation. Together with the fully held-out MT-Bench evaluation, this setup suggests that the learned scheduling behavior captures general cost–quality trade-off patterns in shared-tree construction, rather than being tied to a single model size, model family, or benchmark.

Scheduling traces. For each fitting report, we enumerate the complete binary scheduling space. Since $D = 5$, this gives $2^5 = 32$ complete scheduling trajectories per report and $30 \times 32 = 960$ trajectory-level records in total. Executing one complete scheduling trajectory yields a scheduling trace, which contains: (1) the binary scheduling trajectory $\tau_{1:D}$, (2) the trajectory-level accepted length $A(\tau_{1:D})$ and latency $t(\tau_{1:D})$, and (3) the pre-action online states $z_1, \ldots, z_D$ observed along the trajectory. Collecting these trajectories for all six fitting model pairs costs approximately 160 A100 GPU-hours in total (about 20 hours of wall-clock time with eight A100 GPUs).

During execution, latency is measured as part of the trajectory outcome. In the saved logs, the same information is stored through the accepted length $A(\tau_{1:D})$ and throughput $\eta(\tau_{1:D})$, where throughput is reported in tokens per second. Since the saved latency labels are represented in milliseconds, we reconstruct the trajectory-level latency as

$$t_{\mathrm{ms}}(\tau_{1:D}) = 1000 \cdot \frac{A(\tau_{1:D})}{\eta(\tau_{1:D})}.$$

Thus, the latency label used in the value system is a trajectory-level latency in milliseconds reconstructed from the trajectory-level accepted length and throughput. This is only a unit convention: the main-text definition $\eta = A/t$ corresponds to using latency t in seconds.

At each step k, we record the pre-action state z_k, the current action a_k, the future suffix $\tau_{k+1:D}$, and the trajectory-level labels $A(\tau_{1:D})$ and $t(\tau_{1:D})$. This gives fitting samples of the form

$$(z_k,\ a_k,\ \tau_{k+1:D},\ A,\ t).$$

The labels A and t are always defined at the complete-trajectory level, rather than as local one-step quantities.

Online state and runtime context. Following Section 4, the online state is written as

$$z_k = (\mathbf{c},\ k,\ \tau_{1:k-1},\ \phi_k),$$

where $\mathbf{c}$ is the runtime context, k is the drafting-step index, $\tau_{1:k-1}$ is the scheduling history, and ϕ_k denotes pre-action tree signals computed from the current draft tree.

The runtime context $\mathbf{c}$ contains four features:

$$\mathbf{c} = \big[\log(1 + \text{prompt length}),\ t_{\mathrm{tgt}},\ t_{\mathrm{mid}},\ \log(t_{\mathrm{tgt}}/t_{\mathrm{mid}})\big].$$

Here t_{tgt} and t_{mid} denote the per-call timing of the target model and the middle drafter, respectively. In offline fitting, t_{tgt} and t_{mid} are read from each fitting report using the all-call warm-up trajectory $\tau_{1:D}^{\mathrm{warm}} = (1, 1, 1, 1, 1)$. They correspond to the recorded average per-call target forward time and middle drafter forward time. The prompt-length feature is computed as the model-family–dataset average prompt length and then transformed by $\log(1 + \cdot)$. This report-level averaging is used because the offline trajectory labels are also aggregated at the report level; during online evaluation, $\log(1 + \text{prompt length})$ is computed from the actual input prompt.

The scheduling history is encoded as a 7-dimensional vector:

$$h_k = \Big[k/D,\ (k-1)/D,\ \big(\mathbf{1}[j < k \text{ and } a_j = 1]\big)_{j=1}^{D}\Big].$$

Since $D = 5$ is fixed in the main experiments, no additional padding is required. Positions that are not in the prefix are naturally set to zero by the condition $j < k$.

Canonical online states. When extracting training samples from scheduling traces, we first construct canonical online states. Each canonical state is identified by the tuple

$$(\text{report_id},\ k,\ \tau_{1:k-1}).$$

Different complete scheduling trajectories may visit the same canonical state. We merge these repeated occurrences into a single canonical state. The tree signals of the merged state are averaged over its occurrences, and the state weight is set to

$$w = \sqrt{\text{count}},$$

Table 3: Candidate tree signals used for state selection.

Signal	Definition	Meaning
depth_before	$\frac{1}{\lvert\mathcal{F}_k\rvert}\sum_{v\in\mathcal{F}_k} d(v)$	Average depth of the current frontier
leaf_ratio_before	$\frac{1}{\lvert\mathcal{F}_k\rvert}\sum_{v\in\mathcal{F}_k} \mathbf{1}[\lvert\mathrm{child}(v)\rvert = 0]$	Ratio of leaf nodes in the current frontier
pre_reselect_depth_delta	$\frac{1}{\lvert\widehat{\mathcal{F}}_k\rvert}\sum_{v\in\widehat{\mathcal{F}}_k} d(v) - \frac{1}{\lvert\mathcal{F}_k\rvert}\sum_{v\in\mathcal{F}_k} d(v)$	Change in average frontier depth after reselection preview
pre_reselect_cands	$\lvert\mathcal{R}_k\rvert$	Number of candidate nodes in the reselection preview
max_depth_before	$\max_{v\in\mathcal{F}_k} d(v)$	Maximum depth of the current frontier
collapse_debt	$\max(0,\ -\mathrm{pre_reselect_depth_delta})$	Amount by which the preview frontier becomes shallower
pre_reselect_past_ratio	$\frac{1}{\lvert\widehat{\mathcal{F}}_k\rvert}\sum_{v\in\widehat{\mathcal{F}}_k} \mathbf{1}[d(v)\notin D_F]$	Ratio of preview-frontier nodes from past depths
pre_reselect_overlap	$\frac{\lvert\widehat{\mathcal{F}}_k\cap\mathcal{F}_k\rvert}{\lvert\widehat{\mathcal{F}}_k\rvert}$	Overlap between the current and preview frontiers
pre_reselect_score	$\frac{1}{\lvert\{v\in\widehat{\mathcal{F}}_k : s(v)\text{ finite}\}\rvert}\sum_{\substack{v\in\widehat{\mathcal{F}}_k\\ s(v)\text{ finite}}} s(v)$	Average cumulative score of finite-score nodes in the preview frontier
finite_score_ratio_before	$\frac{1}{\lvert\mathcal{F}_k\rvert}\sum_{v\in\mathcal{F}_k} \mathbf{1}[s(v)\text{ finite}]$	Ratio of finite-score nodes in the current frontier

where count is the number of occurrences of the canonical state in the corresponding report. In the main experiments, this procedure yields 930 canonical online states, which are later used as the teacher states for constructing margin targets.

Each complete scheduling trajectory contributes $D = 5$ action-conditioned return rows. Therefore, the 960 trajectory-level records yield

$$30 \times 32 \times 5 = 4800$$

action-conditioned rows. Among them, 2400 rows correspond to $a_k = 0$ and 2400 rows correspond to $a_k = 1$, forming the skip-return and call-return datasets, respectively.

B.2 Candidate Tree Signals

Table 3 defines the ten candidate tree signals used for state selection. All signals are computed before the action at the k-th drafting step, and therefore correspond to pre-action observations of the current draft tree.

Notation. Let $\mathcal{T}_k$ denote the shared draft tree before the k-th drafting step, and $\mathcal{F}_k$ denote the current frontier. For a node v, let $d(v)$ be its depth, $s(v)$ its cumulative path score, and $\mathrm{child}(v)$ its children. We construct a reselection preview frontier to measure structural change potential. The depth set of the current frontier is

$$D_F = \{d(v) : v \in \mathcal{F}_k\}.$$

Given a lookback window L, the candidate depth set is

$$D_C = \bigcup_{d\in D_F} \{x : \max(0, d - L) \le x \le d\}.$$

The corresponding candidate set is

$$\mathcal{R}_k = \{v \in V(\mathcal{T}_k) : d(v) \in D_C\}.$$

We sort $\mathcal{R}_k$ by cumulative path score $s(v)$ and take the top-W nodes to form the preview frontier:

$$\widehat{\mathcal{F}}_k = \mathrm{Top}_W(\mathcal{R}_k, s).$$

The main experiments set $L = 5$. A pre-action state is kept only when all ten candidate tree signals are finite. If any candidate signal is missing or non-finite, the corresponding state is discarded.

Selected signal subset. The final 6-dimensional signal subset identified in the main experiments is

$$B^\star = \{\mathrm{depth_before},\ \mathrm{leaf_ratio_before},\ \mathrm{collapse_debt},\ \mathrm{pre_reselect_past_ratio},\ \mathrm{pre_reselect_overlap},\ \mathrm{pre_reselect_score}\}.$$

Thus, the tree-state part of the final online scheduler consists of these six pre-action observations.

B.3 Value System Architecture and Training

We denote the selected signal subset by $B^\star$, and the resulting tree-signal vector at step k by $\phi_k^\star$. For each candidate subset $B \subseteq \Phi_k$, the value system f_θ^B predicts the accepted length and log-latency of the complete scheduling trajectory, conditioned on the current online state, the current action, and the future suffix:

$$f_\theta^B(z_k^B,\ a_k,\ \tau_{k+1:D}).$$

The future suffix is explicitly included because z_k and a_k alone do not determine the final trajectory-level outcome.

Input encoding. For a candidate subset B, the value-system input dimension is $17 + |B|$. The input consists of

$$\underbrace{\mathbf{c}}_{4\text{ dims}},\quad \underbrace{h_k}_{7\text{ dims}},\quad \underbrace{\phi_k^B}_{|B|\text{ dims}},\quad \underbrace{u_{k+1:D}}_{6\text{ dims}}.$$

Here $\mathbf{c}$ is the 4-dimensional runtime context, h_k is the 7-dimensional scheduling-history encoding, ϕ_k^B is the selected tree-signal subset, and $u_{k+1:D}$ encodes the future suffix. The future suffix encoding is

$$u_{k+1:D} = \Big[(D-k)/D,\ \big(\mathbf{1}[j > k \text{ and } a_j = 1]\big)_{j=1}^{D}\Big].$$

Positions that are not in the suffix are set to zero by the condition $j > k$. After subset selection, the final selected subset has $|B^\star| = 6$, so the final value-system input has $17 + 6 = 23$ dimensions.

The current action a_k is not concatenated as an additional binary input. Instead, the value system uses action-conditioned heads, so the action information is represented by the identity of the head.

Architecture. For each candidate subset B, the value system consists of four independent MLP heads:

$$(A \mid a_k = 0),\quad (\log t \mid a_k = 0),\quad (A \mid a_k = 1),\quad (\log t \mid a_k = 1).$$

For the final selected subset $B^\star$, each MLP uses the architecture

$$\mathrm{Linear}(23, 128) \to \mathrm{ReLU} \to \mathrm{Linear}(128, 64) \to \mathrm{ReLU} \to \mathrm{Linear}(64, 1),$$

comprising 11,393 parameters per head and 45,572 parameters in total. Inputs and outputs are standardized to zero mean and unit variance. Each head computes its own input and output standardization statistics on the corresponding training data.

The latency head predicts log-latency in milliseconds. When computing predicted throughput, we first recover

$$\hat{t}_{\mathrm{ms}} = \exp(\widehat{\log t_{\mathrm{ms}}}),$$

and then compute

$$\hat{\eta} = 1000 \cdot \frac{\hat{A}}{\hat{t}_{\mathrm{ms}}},$$

where the factor 1000 converts millisecond latency to the token-per-second throughput used in evaluation. This is consistent with the main-text definition $\eta = A/t$ when t is measured in seconds.

Training samples. For a fixed canonical state z_k, current action a_k, and future suffix $\tau_{k+1:D}$, we recover the corresponding complete scheduling trajectory and read its trajectory-level labels A and t. Therefore, the value-system fitting data is arranged as action-conditioned return rows rather than as report-level aggregate regression. Although transition-style rows can be constructed in implementation, the main experiments use only action-conditioned return datasets for system fitting.

Training objective. Each head is trained using weighted MSE:

$$\mathcal{L}_h = \frac{\sum_i w_i \big(f_h(x_i) - y_i\big)^2}{\sum_i w_i}, \tag{5}$$

where w_i is the canonical-state weight inherited from the merged state. Each value-system head is trained with AdamW using learning rate 10^{-2}, weight decay 10^{-4}, maximum 160 epochs, early-stopping patience 20, and minimum improvement 10^{-4}. The four heads use the same base seed with different offsets. If a head has at least 40 training samples, we further split a small validation subset from its training data for early stopping; otherwise, no additional validation split is used.

Tree-signal subset selection. Subset selection evaluates every non-empty subset of the 10 candidate tree signals, resulting in

$$2^{10} - 1 = 1023$$

candidate bases. For each candidate subset B, we fit a separate value system and evaluate it using five-fold grouped out-of-fold validation. The grouping unit is the fitting report: all trajectory rows and canonical states from the same report are assigned to the same fold. Reports are assigned to folds in a round-robin order, which prevents rows from the same report from appearing on both the training and validation sides.

For each candidate subset B, we compute out-of-fold predictions on the skip-return and call-return datasets. On each action branch, we evaluate the normalized mean absolute error (NMAE) for accepted length, latency, and throughput. We define

$$\mathrm{NMAE}(y, \widehat{y}) = \frac{\frac{1}{n}\sum_{i=1}^{n} |y_i - \widehat{y}_i|}{\mathrm{std}(y)}.$$

The NMAE is computed on out-of-fold rows using ordinary MAE and ordinary standard deviation, without additional sample weights.

For the skip branch, we compute

$$E_{\mathrm{skip}}(B) = \frac{1}{3}\left(\mathrm{NMAE}_{A,0}(B) + \mathrm{NMAE}_{t,0}(B) + \mathrm{NMAE}_{\eta,0}(B)\right),$$

and for the call branch,

$$E_{\mathrm{call}}(B) = \frac{1}{3}\left(\mathrm{NMAE}_{A,1}(B) + \mathrm{NMAE}_{t,1}(B) + \mathrm{NMAE}_{\eta,1}(B)\right).$$

The identification score is

$$\mathrm{id_score}(B) = \frac{1}{2}\left(E_{\mathrm{skip}}(B) + E_{\mathrm{call}}(B)\right).$$

We select the 6-dimensional subset $B^\star$ that minimizes this score. In the main experiments, the selected subset achieves

$$\mathrm{id_score}(B^\star) = 0.1863.$$

B.4 Margin Predictor Architecture

The lightweight runtime scheduler $g_\psi(z_k^\star)$ regresses the optimal action-value margin y_k. After fitting the selected value system, we construct this margin by comparing the best predicted future throughput under call and skip:

$$y_k = \max_{\tau_{k+1:D}} \widehat{\eta}_{\theta^\star}(z_k^\star,\ 1,\ \tau_{k+1:D}) - \max_{\tau_{k+1:D}} \widehat{\eta}_{\theta^\star}(z_k^\star,\ 0,\ \tau_{k+1:D}). \tag{6}$$

Since $D = 5$, the maximization over future suffixes is implemented by exhaustive enumeration over all 2^{D-k} binary suffixes. Thus, the margin target is produced by an exact suffix planner within the enumerated call/skip space, rather than by approximate search.

Architecture. The predictor g_ψ is a three-layer MLP:

$$\mathrm{Linear}(17, 64) \to \mathrm{ReLU} \to \mathrm{Linear}(64, 32) \to \mathrm{ReLU} \to \mathrm{Linear}(32, 1),$$

with 3,265 parameters. Its 17-dimensional input is

$$17 = 4 + 7 + 6,$$

consisting of the 4-dimensional runtime context, the 7-dimensional scheduling-history encoding, and the selected 6-dimensional tree-signal subset. Unlike the value system, the margin predictor explicitly excludes the future suffix $\tau_{k+1:D}$. The suffix-dependent reasoning is handled offline by the value system and exhaustive suffix maximization.

Training. After constructing margin teachers for the 930 canonical states, the margin predictor is trained using unweighted MSE:

$$\mathcal{L}_{\text{margin}} = \frac{1}{n}\sum_{i=1}^{n}\big(g_\psi(z_i^\star) - y_i\big)^2.$$

The margin-predictor training objective does not use sample weights. However, when summarizing teacher quality, such as teacher MAE or sign accuracy, we use canonical-state weights for weighted reporting.

During online inference, the policy maps the sign of the predicted margin to the scheduling action:

$$\pi_\psi(z_k^\star) = \begin{cases} 1 \text{ (call)}, & g_\psi(z_k^\star) > 0, \\ 0 \text{ (skip)}, & \text{otherwise.} \end{cases}$$

Thus, online scheduling only requires a single lightweight forward pass at each drafting step.

B.5 Offline Evaluation Results

Table 4 summarizes the offline evaluation metrics for the final scheduler distillation on scheduling traces.

During the state selection phase, the selected 6-dimensional state minimizes the grouped out-of-fold identification score compared with using either the single best signal, which lacks sufficient context, or all 10 candidate signals, which introduces redundancy and noise.

Building upon this optimized state, we evaluate the offline scheduler solving result. As shown in Table 4, the TreeGraft online scheduler achieves 24.30 tok/s on the scheduling traces, close to the best enumerated scheduling trajectory in the call/skip space, which achieves 24.78 tok/s. This demonstrates that the action-value margin constructed from the offline value system effectively supervises the lightweight online scheduler, enabling it to approximate the best scheduling trajectory within the enumerated call/skip space.

Table 4: Scheduler distillation evaluation on scheduling traces. The online scheduler closely approximates the highest throughput achieved by the enumerated scheduling trajectories in the call/skip space.

Method	Throughput (tok/s)
Best enumerated scheduling trajectory in the call/skip space	**24.78**
TreeGraft online scheduler	24.30

B.6 Ablation on the Value System

This appendix analyzes why TreeGraft does not directly deploy the fitted value system as the online scheduler. We compare the TreeGraft scheduler used in the main experiments with a direct value-system planner.

Direct value-system planner. The direct planner uses the fitted value system online. At each drafting step, it compares the predicted throughput of *call* and *skip*, after optimizing over future suffixes, and chooses the action with the higher predicted value. This is the most direct way to use the value system, but it also relies on accurate absolute predictions of accepted length and latency during online decoding.

Offline mean actual gap. We first define the offline metric. For each fitting report, let η_{chosen} denote the true throughput of the trajectory selected by a method, and let η_{oracle} denote the highest true throughput among the enumerated trajectories in the call/skip space of that report. This oracle is not a theoretical global optimum, but the best trajectory within the enumerated call/skip space. The actual gap on this report is defined as

$$\eta_{\text{oracle}} - \eta_{\text{chosen}}.$$

Table 5: Ablation on direct value-system planning. A smaller offline mean actual gap indicates a closer match to the offline enumerated oracle. Overall speedup is measured in real online evaluation relative to autoregressive target decoding.

Method	Offline mean actual gap	Overall speedup
Direct value-system planner	0.3718	0.95×
TreeGraft scheduler	0.4826	**1.60**×

The offline mean actual gap is the average actual gap over all fitting reports. A smaller value means that the method is closer to the enumerated oracle on the offline scheduling traces. However, this metric is not equivalent to the speedup obtained in real online evaluation.

Results. Table 5 shows that a smaller offline gap does not necessarily translate into better online speedup. The direct value-system planner achieves a smaller offline gap than the TreeGraft scheduler, but its online speedup drops to only $0.95\times$. This indicates that directly using the fitted value system for online planning is unreliable, because small prediction errors in accepted length and latency can be amplified when the planner optimizes over future suffixes.

TreeGraft therefore uses the value system only as an offline teacher. It first identifies useful tree signals and then converts trajectory-level accepted length and latency outcomes into a relative call/skip margin. The lightweight runtime scheduler is trained on this margin and only uses the current online state during inference. This design avoids deploying the future-conditioned value system directly online, while achieving substantially better end-to-end speedup.

C Runtime Analysis

C.1 Small Drafter Implementation and Cost Analysis

In TreeGraft, the *small drafter* is the low-cost proposal source used when the scheduler skips the middle drafter. In our implementation, it is a training-free n-gram drafter rather than a neural LLM. It performs no neural-network forward pass; instead, it retrieves candidate tokens and proposal scores through n-gram lookup over the available context, serving as the small drafter expansion step in Section 3.

During prefill, we cache the top-m candidate tokens at each context position together with their target-model probabilities from the prefill logits. Concretely, for a context position i, the target prefill provides a next-token distribution $p_{\mathrm{tar}}(\cdot \mid x_{\leq i})$; we store its top-$m$ tokens and the corresponding probabilities as the cached proposal candidates and scores for that position. We also build an n-gram index over the context. At drafting step k, the small drafter matches the suffix of the current prefix to the context using the longest available n-gram, retrieves the cached candidates at the matched position, and uses the cached probabilities as the edge scores $p(u)$ for expansion. If no match is found, it gradually shortens the n-gram and finally falls back to the cached candidates associated with the position of the most frequent token in the prompt. These cached scores come only from target-model outputs already available during prefill, so the small drafter introduces no additional target forward pass and does not use future verification outcomes or post-prefill target logits. Its lookup cost is included in the reported runtime.

The use of an n-gram drafter is motivated by the latency of real small LLMs. Table 6 reports the per-call latency of representative neural drafters and target models under the online evaluation setting. Even the smallest neural model in each family incurs non-negligible forwarding cost: LLaMA 3.2-1B-Instruct requires 29.1 ms per call, and Qwen3-0.6B requires 44.8 ms per call. Therefore, using a neural model as the small drafter would make the low-cost endpoint substantially more expensive.

Table 7 directly compares the candidate small drafter choices. The n-gram drafter costs only 0.92 ms per call, while the smallest neural alternatives are tens of milliseconds per call. This latency gap is the reason we use the n-gram drafter to define the low-cost endpoint in the main experiments.

Therefore, the goal of All Small is not to represent the strongest possible small drafter, but to provide a genuinely low-cost endpoint for tree construction. This endpoint makes the scheduler's role clear:

Table 6: Per-call latency of representative neural drafters and target models under the online evaluation setting in Section 5.

Model family	Model	Per-call latency (ms)
LLaMA	LLaMA 3.3-70B-Instruct target	122.2
LLaMA	LLaMA 3.2-1B-Instruct	29.1
LLaMA	LLaMA 3.1-8B-Instruct	48.9
Qwen	Qwen3-32B target	77.6
Qwen	Qwen3-0.6B	44.8
Qwen	Qwen3-8B	55.0

Table 7: Per-call latency of candidate small drafter choices under the online evaluation setting in Section 5.

Small drafter	Type	Per-call latency (ms)
N-gram (ours)	Lookup-based	0.92
LLaMA 3.2-1B-Instruct	Neural LLM	29.1
Qwen3-0.6B	Neural LLM	44.8

it decides when a real middle drafter call is worth paying for, rather than comparing two already expensive neural proposal sources.

TreeGraft itself is not restricted to n-gram small drafters. In principle, the same shared-tree grafting and scheduling framework can be combined with stronger neural small drafters or stronger lightweight lookup-based proposers. However, any such replacement requires a separate cost–quality study. We leave this direction for future work.

C.2 Online Scheduler Cost Analysis

The online scheduler is called once per drafting step, so its overhead must be small enough not to become a runtime bottleneck. Table 8 compares the scheduler latency with the main runtime components under the online evaluation setting.

The scheduler costs only 0.318 ms per call, which is lower than the n-gram lookup cost and far smaller than neural forwarding components. With $D = 5$, its total cost per decoding turn is at most $5 \times 0.318 \approx 1.59$ ms. Therefore, the scheduler is not a runtime bottleneck; the end-to-end speedup mainly comes from selectively avoiding unnecessary middle drafter calls rather than from ignoring scheduling overhead.

D Online Evaluation Setting Details

The formal online evaluation is conducted on the 10 target–middle model pairs reported in Table 1. The blue rows in Table 1 correspond to held-out model pairs, which are not used during offline state selection, value-system fitting, or scheduler distillation. The green MT-Bench column corresponds to held-out task evaluation, since no MT-Bench samples are used in the offline stage. The shortened LLaMA names in Table 1 refer to instruct checkpoints: 70B denotes LLaMA 3.3-70B-Instruct, 8B denotes LLaMA 3.1-8B-Instruct, 3B denotes LLaMA 3.2-3B-Instruct, and 1B denotes LLaMA 3.2-1B-Instruct. For Qwen pairs, all models are from the Qwen3 family.

All online evaluations are conducted on two NVIDIA A100 GPUs with batch size 1. All compared methods use the same hardware, batching, tree-construction budget, decoding parameters, and target-verification procedure. For each benchmark, we randomly sample 80 evaluation examples, and the same sampled examples are used for TreeGraft, All Small, All Mid, and all ablation variants.

All online evaluations use the same tree-construction hyperparameters and verification budget. We set the maximum number of drafting steps to $D = 5$, the expansion frontier size to $W = 10$, the number of child tokens generated per expanded node to $m = 10$, and the verification-tree budget to $B_{\text{ver}} = 63$. Here, B_{ver} denotes the maximum number of non-root draft nodes retained after final pruning and submitted to the target model for tree verification. We also set *max_new_tokens* $= 256$

Table 8: Per-call latency of the online scheduler and the main runtime components under the online evaluation setting in Section 5.

Component	Per-call latency (ms)	Ratio vs. target
Target-side verification	69.39	1.000×
Middle-model correction	44.23	0.637×
N-gram small drafter	0.92	0.013×
Online scheduler	0.318	0.005×

and *temperature* $= 0$. In the main online evaluation, we do not impose a fixed middle drafter schedule; instead, the trained scheduler decides at each drafting step whether to call the middle drafter.

The runtime context $\mathbf{c}$ contains four features: the logarithm of the prompt length, t_{tgt}, t_{mid}, and $\log(t_{\mathrm{tgt}}/t_{\mathrm{mid}})$. The prompt length is directly computed from the input sample. The timing features t_{tgt} and t_{mid} denote per-step average latency and are measured during online warm-up. Specifically, for each model pair and benchmark, we run three warm-up rounds on the first evaluation sample with middle drafter calls forced at all five drafting steps, and use the average timing from the third warm-up round to compute t_{tgt}, t_{mid}, and their log ratio. The warm-up sample is used only to measure timing features; its accepted length, verification outcome, throughput, and scheduler decision are not used as supervision.

E Algorithmic Details of Tree Construction

This appendix gives the step-by-step pseudocode corresponding to the tree-construction procedure in Section 3. Algorithm 1 summarizes the standard single-drafter tree construction rule, and Algorithm 2 summarizes the TreeGraft shared-tree construction rule. Both algorithms use the same tree budget and the same target-side verification procedure; the difference lies only in how the draft tree is constructed before verification. Algorithm 3 gives the final global pruning procedure used by both methods.

Notation. Let r denote the root node representing the current verified prefix. At drafting step k, let $\mathcal{T}_k$ be the current draft tree and $V(\mathcal{T}_k)$ its node set. Let $\mathcal{N}_k$ denote the nodes newly generated at the previous step, with $\mathcal{N}_1 = \{r\}$. For each non-root node v, $p(v)$ denotes the stored edge score of the token at v, and $s(v)$ denotes its cumulative path score. The operation $\mathrm{Top}_W(\mathcal{C}, s)$ returns the W highest-scoring nodes in candidate pool $\mathcal{C}$, and $\mathrm{Top}_m(\ell(v))$ returns the top-m child tokens from the next-token logits $\ell(v)$ under the prefix ending at v. The final operation $\mathrm{GlobalPrune}(\mathcal{T}, B_{\mathrm{ver}})$, detailed in Algorithm 3, prunes the constructed tree to the fixed verification budget B_{ver} and returns the pruned verification tree $\mathcal{T}_{\mathrm{verify}}$ together with the verification buffers.

Global score-based pruning. Both Algorithm 1 and Algorithm 2 call the same final pruning procedure before target verification. This step ranks non-root nodes in the constructed draft tree by cumulative path score $s(v)$, using depth $d(v)$ and node id $\mathrm{id}(v)$ only to break ties. It then greedily keeps high-scoring nodes together with the ancestor chains needed to connect them to the root r. The resulting $\mathcal{T}_{\mathrm{verify}}$ satisfies the verification budget and remains a valid tree for building the tree mask, position ids, and retrieval paths.

Algorithm 1 Standard single-drafter tree construction

Require: Verified prefix, single drafter q, target model p_{tar}, maximum drafting steps D, frontier size W, branching size m, verification budget B_{ver}
Ensure: Target-verified output tokens

Initialize the draft tree with root r: $\mathcal{T}_1 \leftarrow \{r\}$
Initialize newly generated nodes: $\mathcal{N}_1 \leftarrow \{r\}$
Set root score $s(r) \leftarrow 1$
for $k = 1, \ldots, D$ **do**
    Form the candidate pool from newly generated nodes: $\mathcal{C}_k \leftarrow \mathcal{N}_k$
    Select the expansion frontier: $\mathcal{F}_k \leftarrow \mathrm{Top}_W(\mathcal{C}_k, s)$
    Initialize the newly generated set for the next step: $\mathcal{N}_{k+1} \leftarrow \emptyset$
    for each node $v \in \mathcal{F}_k$ **do**
        Run the drafter on the prefix ending at v, and obtain next-token logits $\ell_q(v)$
        Select child tokens: $\mathcal{Y}_v \leftarrow \mathrm{Top}_m(\ell_q(v))$
        for each token $y \in \mathcal{Y}_v$ **do**
            Create a child node $u = (v, y)$ under parent v
            Store the edge score $p(u) \leftarrow q(y \mid \pi(v))$
            Compute the cumulative path score $s(u) \leftarrow s(v) \cdot p(u)$
            Attach u to $\mathcal{T}_k$ and add it to $\mathcal{N}_{k+1}$
        end for
    end for
    Set $\mathcal{T}_{k+1} \leftarrow \mathcal{T}_k$
end for
Apply final global pruning: $\mathcal{T}_{\mathrm{verify}} \leftarrow \mathrm{GlobalPrune}(\mathcal{T}_{D+1}, B_{\mathrm{ver}})$
Verify $\mathcal{T}_{\mathrm{verify}}$ with p_{tar} using tree attention and the standard rejection-sampling rule
return Target-verified output tokens

Algorithm 2 TreeGraft shared-tree construction with grafting

Require: Verified prefix, small drafter q_{s}, middle drafter q_{m}, target model p_{tar}, schedule actions $a_{1:D}$, maximum drafting steps D, frontier size W, branching size m, verification budget B_{ver}
Ensure: Target-verified output tokens

Initialize the shared draft tree with root r: $\mathcal{T}_1 \leftarrow \{r\}$
Initialize newly generated nodes: $\mathcal{N}_1 \leftarrow \{r\}$
Initialize the set of nodes visited by the middle drafter: $\mathcal{P}_1 \leftarrow \emptyset$
Set root score $s(r) \leftarrow 1$
for $k = 1, \ldots, D$ **do**
  if $a_k = 0$ **then** ▷ small drafter step
    Form the candidate pool using the standard rule: $\mathcal{C}_k \leftarrow \mathcal{N}_k$
    Reuse the stored edge scores $p(\cdot)$ and cumulative scores $s(\cdot)$
    Select the expansion frontier: $\mathcal{F}_k \leftarrow \mathrm{Top}_W(\mathcal{C}_k, s)$
    Initialize the newly generated set: $\mathcal{N}_{k+1} \leftarrow \emptyset$
    for each node $v \in \mathcal{F}_k$ **do**
      Run the small drafter on the prefix ending at v, and obtain next-token logits $\ell_{\mathrm{s}}(v)$
      Select child tokens: $\mathcal{Y}_v \leftarrow \mathrm{Top}_m(\ell_{\mathrm{s}}(v))$
      for each token $y \in \mathcal{Y}_v$ **do**
        Create a child node $u = (v, y)$ under parent v
        Store the edge score $p(u) \leftarrow q_{\mathrm{s}}(y \mid \pi(v))$
        Compute the cumulative path score $s(u) \leftarrow s(v) \cdot p(u)$
        Attach u to $\mathcal{T}_k$ and add it to $\mathcal{N}_{k+1}$
      end for
    end for
    Keep the middle drafter visited set unchanged: $\mathcal{P}_{k+1} \leftarrow \mathcal{P}_k$
  else ▷ middle drafter step
    Form the expanded grafting-position candidate pool: $\mathcal{C}_k \leftarrow V(\mathcal{T}_k) \setminus \mathcal{P}_k$
    Run one middle drafter forward pass over $\mathcal{C}_k$
    Use this forward pass to update candidate edge scores $p(v)$ for each non-root $v \in \mathcal{C}_k$, keep $s(r) = 1$, recompute cumulative scores $s(v)$, and cache the next-token logits $\ell_{\mathrm{m}}(v)$
    Select the grafting frontier using the updated scores: $\mathcal{F}_k \leftarrow \mathrm{Top}_W(\mathcal{C}_k, s)$
    Initialize the newly generated set: $\mathcal{N}_{k+1} \leftarrow \emptyset$
    for each node $v \in \mathcal{F}_k$ **do**
      Reuse the cached middle drafter logits $\ell_{\mathrm{m}}(v)$
      Select child tokens: $\mathcal{Y}_v \leftarrow \mathrm{Top}_m(\ell_{\mathrm{m}}(v))$
      for each token $y \in \mathcal{Y}_v$ **do**
        Create a child node $u = (v, y)$ under parent v
        Store the edge score $p(u) \leftarrow q_{\mathrm{m}}(y \mid \pi(v))$
        Compute the cumulative path score $s(u) \leftarrow s(v) \cdot p(u)$
        Attach u under v without overwriting any existing child or branch
        Add u to $\mathcal{N}_{k+1}$
      end for
    end for
    Mark all nodes in the middle drafter candidate pool as visited: $\mathcal{P}_{k+1} \leftarrow \mathcal{P}_k \cup \mathcal{C}_k$
  end if
  Set $\mathcal{T}_{k+1} \leftarrow \mathcal{T}_k$
end for
Apply final global pruning: $\mathcal{T}_{\mathrm{verify}} \leftarrow \mathrm{GlobalPrune}(\mathcal{T}_{D+1}, B_{\mathrm{ver}})$
Verify $\mathcal{T}_{\mathrm{verify}}$ with p_{tar} using tree attention and the standard rejection-sampling rule
return Target-verified output tokens

Algorithm 3 Global score-based pruning with ancestor closure

Require: Draft tree $\mathcal{T}$ with root r, node scores $s(v)$, verification budget B_{ver}
Ensure: Pruned verification tree $\mathcal{T}_{\text{verify}}$, tree mask, position ids, and retrieval paths

Let $M \leftarrow B_{\text{ver}} + 1$, where the extra node is the root
if $|V(\mathcal{T})| \leq M$ **then**
    return $\mathcal{T}$ and its verification buffers
end if
Compute the depth $d(v)$ of every node $v \in V(\mathcal{T})$
Rank all non-root nodes by decreasing score:

$$v_1, v_2, \ldots \leftarrow \text{Sort}\big(V(\mathcal{T}) \setminus \{r\}, (-s(v), d(v), \text{id}(v))\big)$$

Initialize the kept node set: $\mathcal{K} \leftarrow \{r\}$
for each ranked node v_i **do**
    Trace the ancestor chain from v_i upward until reaching a node already in $\mathcal{K}$:

$$\mathcal{A}(v_i) \leftarrow \{v_i, \text{parent}(v_i), \ldots\} \setminus \mathcal{K}$$

    if $|\mathcal{K}| + |\mathcal{A}(v_i)| \leq M$ **then**
        Add the whole chain: $\mathcal{K} \leftarrow \mathcal{K} \cup \mathcal{A}(v_i)$
    end if
    if $|\mathcal{K}| = M$ **then**
        break
    end if
end for
Construct the induced subtree $\mathcal{T}_{\text{verify}} \leftarrow \mathcal{T}[\mathcal{K}]$
Re-index the kept nodes so that parents always appear before children
Build the tree-attention mask:

$$\text{mask}(u, v) = 1 \quad \text{iff} \quad v = r \text{ or } v \text{ is an ancestor of } u \text{ or } v = u$$

Set each node's position id to its depth in $\mathcal{T}_{\text{verify}}$
Extract all root-to-leaf paths in $\mathcal{T}_{\text{verify}}$ as retrieval paths and pad shorter paths with -1
return $\mathcal{T}_{\text{verify}}$, tree mask, position ids, and retrieval paths

F Qualitative Examples of Non-destructive Grafting

To complement the quantitative ablation in Table 2, we provide two traced examples showing why non-destructive grafting is needed. The trace collection follows the online evaluation setting in Section 5. We use the Qwen3-32B/Qwen3-0.6B target–middle pair on CNN/DailyMail, with the same tree-construction budget and decoding configuration as in Section 5. To isolate the grafting mechanism, we use the fixed scheduling pattern $[1, 0, 1, 0, 1]$, where 1 denotes calling the middle drafter and 0 denotes using the small drafter. Thus, the middle drafter is called at drafting steps 1, 3, and 5.

Each example focuses on one middle drafter step. Under expanded grafting-position selection, the middle drafter may select a historical node, i.e., a node that already exists in the shared tree before the current middle drafter step. This selected historical node may already have children and a subtree. Non-destructive grafting attaches the newly generated middle drafter children under this node without removing the existing children. The examples below show cases where the existing subtree contains a branch later accepted by the target model. If overwriting were used instead, that accepted branch would be discarded.

We use the following notation:

[H]: node already present before the current middle drafter step,

[M]: node newly attached by the current middle drafter step.

In the text trees, * marks the selected grafting position, and + marks nodes on the later target-accepted path. Omitted branches are denoted by ....

Case 1: The selected grafting position is the historical node `.`. Before the current middle drafter step, this node already contains the historical branch `L -> umber -> Liquid -> ators`. The target later accepts this branch, forming the continuation `. Lumber Liquidators`.

```
[H] <root>
\-- [H] . * +
    |-- [H] L +
    |   \-- [H] umber +
    |       \-- [H] Liquid +
    |           \-- [H] ators +
    |-- [M] The
    |-- [M] They
    |-- [M] CBS
    \-- ...
```

Non-destructive grafting preserves the historical branch while attaching new middle drafter children such as `The`, `They`, and `CBS`. With overwriting, the accepted continuation `. Lumber Liquidators` would be removed.

Case 2: The selected grafting position is the historical node `that`. Before the current middle drafter step, this node already contains the historical branch `P -> elle -> gr -> ini`. The target later accepts this branch, forming the continuation `mentions that Pellegrini`.

```
[H] <root>
\-- [H] mentions +
    \-- [H] that * +
        |-- [H] P +
        |   \-- [H] elle +
        |       \-- [H] gr +
        |           \-- [H] ini +
        |-- [M] Martinez
        |-- [M] Hart
        |-- [M] the
        \-- ...
```

Again, non-destructive grafting keeps the existing historical subtree while adding new middle drafter children under the selected historical node. With overwriting, the target-accepted continuation `mentions that Pellegrini` would be discarded.

G Additional Experimental Results

G.1 Supplementary Study on the Small Drafter Choice

In the main experiments, we use a training-free n-gram drafter as the small drafter. This is because TreeGraft relies on the small drafter as a genuinely low-cost expansion source. If the small drafter is replaced by a neural model with nontrivial forwarding cost, the default expansion steps may substantially reduce the overall speedup.

To examine this effect, we conduct a fixed-schedule preliminary study. We remove the learned scheduler and call the middle drafter at drafting steps 1, 3, and 5, i.e., using the schedule $[1, 0, 1, 0, 1]$. For each middle–target pair, we compare two small drafter choices: the smallest pretrained neural model from the same family, and the training-free n-gram drafter. All other settings, including the benchmark, tree budget, decoding hyperparameters, and middle–target pair, are kept unchanged. All speedups are measured relative to autoregressive decoding with the target model.

Table 9 compares the two small drafter choices under the same middle–target pairs. Replacing the neural small drafter with the n-gram drafter consistently improves speedup in all three configurations. On average, the speedup increases from $0.865\times$ to $1.180\times$, corresponding to a relative gain of 36.4%. In the two Qwen settings, using Qwen3-0.6B as the small drafter even leads to speedups below $1\times$, meaning that this configuration becomes slower than target-only autoregressive decoding.

These results support our main design choice. In the shared-tree setting, using two neural drafters does not necessarily improve efficiency, because the forward cost of the neural small drafter can offset

Table 9: Effect of the small drafter choice under a fixed middle drafter schedule. The middle drafter is called at steps 1, 3, and 5. Values are speedups over autoregressive target decoding.

Small / Middle / Target	Speedup
LLaMA 3.2-1B-Instruct / LLaMA 3.1-8B-Instruct / LLaMA 3.3-70B-Instruct	1.116×
n-gram / LLaMA 3.1-8B-Instruct / LLaMA 3.3-70B-Instruct	1.299×
Qwen3-0.6B / Qwen3-4B / Qwen3-32B	0.735×
n-gram / Qwen3-4B / Qwen3-32B	1.108×
Qwen3-0.6B / Qwen3-8B / Qwen3-32B	0.743×
n-gram / Qwen3-8B / Qwen3-32B	1.134×
Average with neural small drafter	0.865×
Average with n-gram small drafter	1.180×

the benefit of speculative decoding. The n-gram drafter provides a cheaper small drafter endpoint, allowing TreeGraft to reserve neural forwarding cost for the selectively invoked middle drafter.

G.2 Stability of Online Evaluation

To check whether the main online results are sensitive to wall-clock measurement noise, we repeat the online evaluation of the scheduler used in the main experiments. The two runs use the same scheduler parameters, identified tree signals, benchmarks, evaluation inputs, and decoding hyperparameters. The only difference is that the wall-clock measurements are collected from two independent online evaluation runs. We denote the results used in Table 1 as Run 1, and the repeated evaluation as Run 2.

We evaluate two representative target–middle pairs: Qwen3-32B/Qwen3-0.6B and LLaMA 3.3-70B-Instruct/LLaMA 3.2-1B-Instruct. The former represents a Qwen setting where the benefit depends more on selective middle drafter calls, while the latter represents a LLaMA setting where the middle drafter is substantially smaller than the target model. Table 10 reports the average speedup over six benchmarks.

Table 10: Stability of repeated online evaluation. Run 1 denotes the result used in the main table, and Run 2 denotes an independent repeated evaluation with the same scheduler and decoding settings. Values are average speedups over six benchmarks.

Target–middle pair	Run 1	Run 2	Abs. diff.
Qwen3-32B / Qwen3-0.6B	1.470×	1.473×	0.003×
LLaMA 3.3-70B-Instruct / LLaMA 3.2-1B-Instruct	2.297×	2.289×	0.008×
Average	1.884×	1.881×	0.003×

The repeated evaluation produces nearly identical results. Across the two representative target–middle pairs, the average speedup changes from $1.884\times$ to $1.881\times$, with an absolute difference of only $0.003\times$. This suggests that the main online results are stable under repeated wall-clock evaluation, and that the conclusions in Section 5 do not depend on a single timing run.

G.3 Complete Main Results and Endpoint Synergy

Table 11 gives the complete per-task speedups underlying the compact main-text table.

Table 11: Online evaluation results. Values are speedups over autoregressive target decoding. TG denotes TreeGraft, while S and M denote the single-drafter endpoints All Small and All Mid, respectively. The small drafter used by TG and S is a training-free n-gram drafter. Blue rows denote held-out target–middle pairs not used in offline fitting or scheduler distillation; green columns denote the held-out MT-Bench task; orange cells denote their intersection. For LLaMA pairs, 70B, 8B, 3B, and 1B refer to LLaMA 3.3-70B, LLaMA 3.1-8B, LLaMA 3.2-3B, and LLaMA 3.2-1B, respectively.

Target–Middle	Alpaca			GSM8K			HumanEval			NQ			CNN/DM			MT-Bench			Average			Improvement	
	TG	S	M	TG	S	M	TG	S	M	TG	S	M	TG	S	M	TG	S	M	TG	S	M	TG vs S	TG vs M
Qwen3-32B/1.7B	**1.43**	1.07	1.23	**1.72**	1.46	1.47	**1.49**	1.19	1.27	**1.39**	1.06	1.15	**1.42**	1.22	1.23	**1.51**	1.26	1.21	**1.49**	1.21	1.26	23.1%	18.3%
Qwen3-32B/0.6B	**1.38**	1.07	1.09	**1.79**	1.46	1.48	**1.46**	1.19	1.24	**1.34**	1.06	1.07	**1.39**	1.22	1.09	**1.46**	1.26	1.18	**1.47**	1.21	1.19	21.5%	23.5%
Qwen3-32B/8B	**1.32**	1.07	0.99	**1.60**	1.46	1.24	**1.35**	1.19	1.03	**1.33**	1.06	0.98	**1.34**	1.22	1.04	**1.43**	1.26	1.08	**1.40**	1.21	1.06	15.7%	32.1%
LLaMA-70B/1B	**2.24**	1.19	2.21	2.48	1.57	**2.52**	2.52	1.82	**2.58**	**2.19**	1.16	2.14	2.09	1.46	**2.14**	**2.26**	1.44	2.21	**2.30**	1.44	**2.30**	59.7%	0.0%
LLaMA-70B/3B	**1.87**	1.19	**1.87**	2.05	1.57	**2.08**	2.10	1.82	**2.13**	**1.89**	1.16	1.84	1.85	1.46	**1.87**	**1.91**	1.44	**1.91**	**1.95**	1.44	**1.95**	35.4%	0.0%
LLaMA-8B/1B	**1.28**	1.14	1.27	**1.59**	1.50	1.37	**1.87**	1.75	1.45	**1.33**	1.15	1.21	**1.48**	1.44	1.25	**1.47**	1.40	1.26	**1.50**	1.40	1.30	7.1%	15.4%
Qwen3-32B/4B	**1.33**	1.07	1.10	**1.61**	1.46	1.30	**1.31**	1.19	1.15	**1.31**	1.06	1.08	**1.34**	1.22	1.13	**1.41**	1.26	1.15	**1.39**	1.21	1.15	14.9%	20.9%
Qwen3-8B/0.6B	**1.14**	1.10	0.80	**1.50**	1.45	0.97	**1.24**	1.23	0.89	**1.19**	1.16	0.80	**1.24**	1.19	0.80	**1.29**	1.28	0.84	**1.27**	1.24	0.85	2.4%	49.4%
LLaMA-70B/8B	1.77	1.19	**1.78**	1.93	1.57	**1.97**	1.93	1.82	**2.03**	1.73	1.16	**1.75**	1.77	1.46	**1.82**	**1.83**	1.44	1.81	1.83	1.44	**1.86**	27.1%	-1.6%
LLaMA-8B/3B	**1.17**	1.14	0.91	**1.53**	1.50	0.97	1.74	**1.75**	1.02	**1.19**	1.15	0.89	**1.46**	1.44	0.92	**1.42**	1.40	0.94	**1.42**	1.40	0.94	1.4%	51.1%
Average	**1.49**	1.12	1.33	**1.78**	1.50	1.54	**1.70**	1.50	1.48	**1.49**	1.12	1.29	**1.54**	1.33	1.33	**1.60**	1.34	1.36	**1.60**	1.32	1.39	21.2%	15.1%

G.4 Scheduler Transfer Across Temperature, Width, Models, and Hardware

We test no-refit scheduler transfer across four deployment changes: sampling temperature ($T = 0 \rightarrow 1$), candidate-tree width ($W = 10 \rightarrow 5$), model family, and hardware (A100→H20). For temperature transfer, we directly apply the scheduler fitted at $T = 0$ to $T = 1$ across all 60 main-table settings. As shown in Table 12, TreeGraft reaches 1.55×, outperforming All-Small (1.34×) and All-Mid (1.01×). As a control, we recollect call/skip trajectories and refit the scheduler at $T = 1$, obtaining a comparable speedup of 1.56×. Thus, the no-refit result shows that the scheduler fitted at $T = 0$ can transfer to non-greedy sampling at $T = 1$.

Table 12: Average speedup under sampling decoding with $T = 1$.

Method	Setting	Average speedup	Improvement over better fixed endpoint
All-Small ($T = 1$)	—	1.34×	—
All-Mid ($T = 1$)	—	1.01×	—
TreeGraft ($T = 1$)	Refit	**1.56**×	+16.4%
TreeGraft ($T = 1$)	No refit	**1.55**×	+15.7%

We directly apply the scheduler fitted at $W = 10$ to $W = 5$ across all 60 main-table settings, with no refitting. As shown in Table 13, at $W = 5$, TreeGraft reaches 1.50× versus 1.32× and 1.24× for the endpoints, a 13.6% improvement over the better fixed endpoint. This shows that the scheduler can transfer to some extent under a change in candidate-tree width.

Table 13: Average speedup in no-refit scheduler transfer under a change in candidate-tree width.

Tree width	TreeGraft	All-Small	All-Mid	Improvement over better fixed endpoint
$W = 10$	**1.60**×	1.32×	1.39×	+15.1%
$W = 5$	**1.50**×	1.32×	1.24×	+13.6%

Table 1 already provides evidence of scheduler transfer to four held-out model pairs from seen families and to MT-Bench, which was excluded from fitting. We further test both cross-family directions (30 settings each) and A100→H20 transfer (all 60 settings). For each cross-family direction, the scheduler is fitted using the source-family fitting pairs specified in Appendix B.1 and then evaluated on the other family without refitting. For A100→H20, the scheduler fitted on A100 is directly evaluated on H20 without refitting. As shown in Table 14, TreeGraft improves over the better fixed endpoint by 14.6%, 6.5%, and 32.1%, respectively. These results indicate that the scheduler can transfer to some extent across model pairs, model families, workload, and hardware without refitting.

Table 14: Average speedup in transfer evaluation across model families and hardware.

Deployment change	TreeGraft	All-Small	All-Mid	Improvement over better fixed endpoint
Qwen fitting → LLaMA testing	**1.65**×	1.44×	1.32×	+14.6%
LLaMA fitting → Qwen testing	**1.32**×	1.24×	0.95×	+6.5%
A100 → H20	**1.85**×	1.40×	1.15×	+32.1%

G.5 Scheduler Feature Ablations

We separately remove runtime context, scheduling history, and tree signals and refit the scheduler for each ablation. The comparison covers all 60 settings in the main table. As shown in Table 15, relative to the corresponding ablated variants, the full scheduler improves average speedup by 9.6%, 10.3%, and 7.4%, respectively. The largest observed contributions come from scheduling history and runtime context, while tree signals also contribute to performance.

Table 15: Ablation of scheduler features.

Scheduler	Average speedup	Full-scheduler speedup improvement
Without runtime-context features	1.46×	+9.6%
Without scheduling-history features	1.45×	+10.3%
Without tree-signal features	1.49×	+7.4%
Full learned scheduler	**1.60**×	–

G.6 Comparison with Heuristic Schedulers

We evaluate four heuristic schedulers over all 60 main-table settings: early-call, entropy/confidence, candidate-score-gap, and latency-normalized. For each heuristic, we report the single global parameter from a predefined grid that maximizes average speedup. This provides an optimistic upper bound on each heuristic's performance within the predefined grid; the selected value appears in the method name.

Table 16: Learned scheduler vs. heuristic baselines.

Method	Average speedup	Learned-scheduler speedup improvement
Always-Early (call in the first 3 drafting steps)	1.48×	+8.1%
Entropy/Confidence ($\tau = 0.7$)	1.25×	+28.0%
Candidate-Score-Gap ($\tau = 0.7$)	1.46×	+9.6%
Latency-Normalized ($\tau = 1.0$)	1.50×	+6.7%
Learned scheduler	**1.60**×	–

As Table 16 shows, the strongest heuristic, Latency-Normalized, achieves 1.50× but remains below the learned scheduler's 1.60×. This gap is consistent with the fact that each heuristic captures only one aspect of the scheduling state, whereas the learned scheduler combines runtime context, tree signals, and observed scheduling history to estimate the middle drafter's marginal benefit and latency cost.

H Existing Assets and Licenses

We use publicly available pretrained checkpoints and benchmarks, including LLaMA models, Qwen3 models, GSM8K, Alpaca, NQ, HumanEval, CNN/DM, and MT-Bench. We cite the original sources for all models and datasets and use them only for research evaluation. The LLaMA checkpoints are used under the Meta LLaMA license and acceptable-use policy, Qwen3 checkpoints are released under the Apache License 2.0, GSM8K and HumanEval are released under the MIT License, Alpaca is released for research use under CC BY-NC 4.0, and Natural Questions is released under Apache License 2.0. For CNN/DailyMail and MT-Bench, we follow the terms and licenses specified by their

official dataset/model-card or repository distributions. For datasets whose redistributed versions have additional terms, we follow the terms provided by the corresponding dataset hosts.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main scope of TreeGraft as a training-free multi-drafter framework for shared-tree speculative decoding, and the reported claims are supported by the method design in Sections 3–4 and the empirical results in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a separate Limitations section discussing the training-free scope of TreeGraft and the use of an n-gram drafter as the low-cost small drafter. These limitations clarify that TreeGraft is orthogonal to training-based draft-source improvement and that the lookup-based small drafter is one practical instantiation rather than an exhaustive choice. (See Appendix A)

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results, theorems, or proofs. The mathematical expressions are used to define the algorithmic components and scheduling objective.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides the model pairs, datasets, baselines, metrics, tree-construction hyperparameters, offline fitting procedure, scheduler training details, and online evaluation protocol in Section 5 and Appendices B–D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide a code repository at `https://github.com/fjm9933/TreeGraft`. The experiments use public benchmarks and publicly available pretrained checkpoints. Together with the algorithmic details, hyperparameters, tree-construction settings, scheduler descriptions, and evaluation protocols provided in the paper and supplemental material, the released code supports reproduction of the main experimental results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 specifies the evaluated model pairs, datasets, baselines, metrics, and decoding hyperparameters. Appendices B and D further describe fitting data collection, online states, scheduler training, evaluation subsets, and runtime settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Appendix G.2 reports repeated-run stability statistics for the timing-based online speedup results under the same scheduler, inputs, and decoding hyperparameters. We use this as variability information.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix D reports the hardware and runtime configuration used for online evaluation, including the use of two NVIDIA A100 GPUs, batch size 1. Appendix C also reports representative per-call latency measurements for target and drafter models.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses public language-model checkpoints and public text benchmarks, and does not involve human-subject experiments, private data, or sensitive personal information. We are not aware of any deviation from the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The work may reduce the computational cost and latency of LLM inference, which can improve accessibility and energy efficiency. At the same time, faster inference could also lower the cost of generating harmful or misleading text, so responsible deployment should follow the safeguards of the underlying language models.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new pretrained language model, image generator, scraped dataset, or other high-risk asset. TreeGraft is an inference acceleration framework applied to existing models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the original sources for all pretrained models and benchmarks used in the experiments and includes an existing-assets statement in Appendix H. We use these assets for research evaluation and follow their corresponding licenses and terms of use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not introduce or release a new dataset, pretrained model, or benchmark asset. The contribution is an algorithmic framework and evaluation protocol for inference acceleration.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human-subject research, so IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development, experimental design, and scientific conclusions of this work do not rely on LLMs as an important, original, or non-standard research component. Any use of LLM-based tools, if any, was limited to writing, editing, or formatting assistance and did not affect the methodology, experiments, or results.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.