SKIVE: SPARSE-KV IMPORTANCE-SCHEDULED VERIFICATION BY EXPECTED TRUE POSITIVES

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding (SD) drafts candidate tokens and verifies them in one target forward pass; sparse key–value (KV) cache selection reads only a budgeted subset of the cached context. Verifying the drafted sequence under a sparse cache is lossy in a way the standard acceptance count hides: sparse verification can accept tokens dense verification would reject. We score sparse verification by its *true positives*: the accept decisions it shares with the dense verifier. From one assumption, that a node's chance of a true positive rises linearly in the importance mass (the selector's scores) the retained blocks keep, we derive an objective for the two decisions a sparse verifier makes: *which layers re-select the cache*, and how sparse to go. **Sparse-KV Importance-scheduled Verification by Expected true positives (SKIVE)** applies it to the first, which prior work settles offline by calibrated search. Reusing one selection across all layers costs 15–27 true-positive points; the objective scores a stale selection by the mass it retains, so the best set of re-selecting layers maximizes retained mass, solved exactly by dynamic programming over a table the model builds from its own selections during decoding, and the same solve fixes the anchor count. The schedule needs no calibration data and its only model input is the layer count; on LLaMA-3.1-8B placement matters more than count: the six to eight anchors it chooses reproduce per-layer true positives, whereas six uniformly spaced layers lose 5–8 true-positive points. Against measured costs the objective becomes a throughput function Φ(sparsity, anchor count) that bounds the sparsity and says when sparse verification should, and should not, be used. End to end on LLaMA-3.1-8B with an Arctic-LSTM tree drafter at 64K, the online schedule reaches $5.5\times$ autoregressive (AR) decoding at 4.8% lower ROUGE-L, against $3.4\times$ for per-layer selection and $1.15\times$ for dense SD, and within 3% ROUGE-L of AR with the same sparse cache; it transfers to Qwen3-8B with no calibration, and a fused-kernel SGLang port goes from behind dense tree verification at batch 1 to 52% ahead at batch 8.

1 INTRODUCTION

Long-context LLM decoding pays two memory-bandwidth costs per token: every token re-reads the model weights, and every attention layer re-reads the KV cache, which grows with context. *Speculative decoding* (SD) (Leviathan et al., 2023; Chen et al., 2023) amortizes the weight read: a cheap drafter proposes tokens the target verifies in one forward pass. *Sparse KV-cache selection* (Tang et al., 2024; Zhang et al., 2023) attacks the cache read: blocks are scored against the current query and only a budgeted top-scoring subset is read. Measured on one stack (Figure 1), the two are complementary: SD's gain fades with context as the KV read comes to dominate, and sparse selection pays only once the avoided cache read exceeds the latency of selecting blocks, extra per-layer work that scores each block's key statistics against the query and takes a top-K before attention. Since each helps where the other does not, we compose them and *verify the drafted candidates under a sparse cache*.

Composing the two is not free. A sparse verifier reads a different cache from the dense one, so it can accept a candidate the dense verifier would reject and silently change the output, and the standard acceptance count cannot see it. Prior compositions (Wang et al., 2025; Lee et al., 2026a; Wang et al., 2026a) report acceptance and task accuracy, so this loss goes unmeasured. We score sparse verification by its *true positives*, the accept decisions it shares with the dense verifier, and derive

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

from one assumption, that a node's chance of a true positive rises linearly in the importance mass the retained blocks keep for it, an objective for the two decisions the composition adds (§4.1).

The first decision is which of the L layers re-select the cache, since every scored layer costs latency. Prior work fixes the set offline on calibration data, by a sensitivity sweep, layer-to-layer overlap or greedy search on language-model loss (§3), and holds it fixed per model; the dominant strategy, root reuse (Wang et al., 2025), scores once for the first draft token and reuses that selection for every tree node and layer. None chooses the set by agreement with the dense verifier, and none adapts it during decoding. At matched budgets root reuse costs 15–27 true-positive points and 8–15% ROUGE-L end to end, and we find the loss comes from reusing one selection across layers, not across tree nodes (§4.2); re-selecting at every layer recovers it but scores all L layers. Our objective scores a reused selection by the mass it retains at the reusing layer; over all layer pairs these scores form one table, on which the best set of re-selecting layers, the *anchors*, is a chain problem that dynamic programming (DP) solves exactly at every count, the count chosen where one more anchor stops recovering mass. Built from the model's own selections during decoding, six to eight anchors placed this way recover per-layer agreement at a quarter of the scoring cost (Table 3).

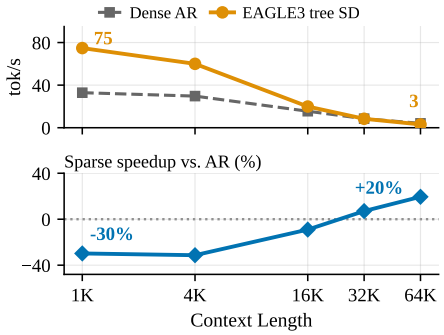


Figure 1: **Why compose.** Tree SD's gain over AR fades as the KV read grows (top); a sparse cache (s=0.9) on plain AR pays only past 16K, once the avoided read outgrows the selection overhead (bottom). LLaMA-3.1-8B + EAGLE3, HF/SDPA, RTX 6000 Ada.

The second decision is how sparse to go. Existing selectors leave the budget as a hyperparameter, and none accounts for the scoring cost the schedule adds. Against measured round costs our objective becomes a throughput function, $\Phi(s, n_a)$, correct tokens per second as a function of sparsity and scored layers, which bounds the sparsity and says when sparse verification should, and should not, be used. Because it is defined per round it does not see loss that accumulates across rounds, so the operating point is capped by a quality floor measured end to end (§4.4, §5.3).

Contributions. **(1)** We propose expected true positives, the accept decisions a sparse verifier shares with the dense one, as the quantity sparse verification should maximize; it exposes the false accepts the acceptance count hides and, against measured costs, bounds the sparsity and predicts when sparse verification pays. **(2)** We show that root reuse loses its true positives by reusing one selection across layers rather than across tree nodes, and we turn the choice of re-selecting layers into a scheduling problem that an online controller solves during decoding from the model's own selections, with no calibration data. **(3)** Our experiments show that six to eight self-chosen anchors reproduce per-layer agreement at a quarter of the scoring work: with an Arctic-LSTM tree drafter at 64K the controller reaches 5.51× AR against 3.44× for per-layer selection and 1.15× for dense SD, within 3% ROUGE-L of AR with the same sparse cache, and a fused-kernel SGLang port goes from behind dense tree verification at B=1 to 52% ahead at B=8, as the cost model predicts.

2 Background

2.1 Speculative verification and what sparsity can lose

Given a committed prefix, a *drafter* proposes a tree of N candidate tokens whose root-to-leaf paths are the possible continuations; chain drafters are the degenerate case with one path. The target verifies all N nodes in one forward pass under a *tree-causal mask*: node v attends only to the prefix and its own ancestors. Greedy verification accepts a child if and only if it equals the target's argmax at that position; the longest accepted path is committed, plus one *bonus token*, the target's argmax after it. We write accepted tokens per round, bonus included, as τ (TACC in tables). Under greedy decoding dense verification is *lossless* (bit-identical to AR in FP32), so sparsity alone explains any quality change. Sparse verification can flip an accept decision: comparing the two verifiers on the same candidate and prefix gives a *true positive* (TP) when both accept, a *false positive* (FP) when only

sparse accepts, which silently alters the output, and a *false negative* when sparse rejects a candidate dense accepts, which shortens the accepted path and, through the bonus token, can also change the output (Table 1). The standard statistic counts both, $\tau_{\text{sparse}} - 1 = \text{TP} + \text{FP}$ per round, so it overstates sparse verification; we *optimize TP*, report TP and FP together, and pair ROUGE-L against AR.

Table 1: Sparse vs. dense verification of one candidate token: TP is the optimization target, FP the harmful event.

	Dense: Accept	**Dense: Reject**
Sparse: Accept	**TP** — correct acceptance	**FP** — corrupts output
Sparse: Reject	FN — loses an accepted token	TN — correct rejection

2.2 Sparse selection, rounds, anchors, schedules

The KV cache is partitioned into M blocks of 16 tokens. A *selector* scores each block against a query: by dot product with the block's mean key (MeanPool), by min/max key envelopes (Quest (Tang et al., 2024)), or by variants (TokenSelect (Wu et al., 2025), TidalDecode (Yang et al., 2025a)). At *sparsity* s, attention reads only $K = \lfloor M(1-s) \rfloor$ blocks, shared by all KV heads; the first and last 8 blocks are always kept. Selection applies only to the verify forward: prefill and the drafter run dense. A *round* is one verification: the drafter proposes a tree, the base model runs it through all L layers in one forward, and the verifier accepts a prefix. A layer that recomputes its selection is an *anchor*; the set of anchors is the *schedule* $\mathcal{A} \subseteq \{0, \ldots, L-1\}$ with $0 \in \mathcal{A}$, and every other layer ℓ reuses the selection of its nearest preceding anchor $a(\ell) = \max\{a \in \mathcal{A} : a \leq \ell\}$; every layer keeps its own KV cache, so a reused selection shares block indices and each layer reads its own keys and values at those blocks. Its size $n_a = |\mathcal{A}|$ is the number of layers that pay for scoring in a round: per-layer selection is $n_a = L$, and root reuse is $\mathcal{A} = \{0\}$ scored with the root query. Sparsity shrinks only the KV-attention read, which is 26% of round time at 16K and 45% at 64K on our EAGLE3 setup (Appendix A.1).

3 Related Work

Tree and long-context speculative decoding. Tree-structured SD (Cai et al., 2024; Miao et al., 2024; Li et al., 2024a;b; 2025) is our substrate; SKIVE takes the tree topology from the drafter and does not change it. Long-context variants keep the target lossless by sparsifying or quantizing the *draft* cache (TriForce (Sun et al., 2024), MagicDec (Sadhukhan et al., 2024), LongSpec (Yang et al., 2025b), SparseSpec (Zhao et al., 2025), Vegas (Yue et al., 2026), VeriCache (Yao et al., 2026), QuantSpec (Tiwari et al., 2025)), a different axis from ours; SpecPV (Tan et al., 2025) verifies over a partial cache, DeFT (Yao et al., 2025) optimizes exact tree-attention kernels, Bastion (Oh et al., 2026) budgets draft tokens rather than KV blocks, and MARS (Song et al., 2026) relaxes acceptance where the dense margin is small, whereas we keep acceptance strict. Dustin (Lee et al., 2026a) sparsifies the target's cache from the drafter's attention with one global index set for all layers, at batch 8–16 with chain drafts and a fixed token budget, and reports task accuracy and attention recovery rather than agreement with the dense verifier; the layer axis it fixes globally is the one SKIVE schedules.

Sparse KV-cache selection and cross-layer reuse. Selectors differ in how blocks are scored (Quest (Tang et al., 2024), TokenSelect (Wu et al., 2025), TidalDecode (Yang et al., 2025a), H_2O (Zhang et al., 2023)); a complementary line compresses the cache itself (Xiao et al., 2025; Sun et al., 2025). All assume a single query per decode step, and the budget is typically a free hyperparameter (Twilight's (Lin et al., 2025) adaptive top-p comes closest to deriving it); Φ (§4.4) sets it from a measured TP curve and stack costs. Reusing a selection across layers is the standard way to amortize scoring. TidalDecode re-selects at a few layers found by a sensitivity sweep; Wang et al. (2025) either re-score the root query at every layer or pick anchor layers by adjacent-layer Jaccard overlap on calibration data; IndexCache (Bai et al., 2026) and SSV (Wang et al., 2026a) fix full/shared or refresh/reuse layers by greedy search over language-model loss on calibration data, SSV adding an offline profile table over context buckets. CLSA (Sun et al., 2026) trains a shared routing index; PRR (Wang et al., 2026b) reuses top-K sets across decode steps; PyramidKV, SqueezeAttention and

AdaKV (Cai et al., 2024; Wang and Gan, 2024; Feng et al., 2024) vary the budget *size* per layer, a different axis. Across this family the layer set is chosen offline against a proxy (language-model loss, layer-to-layer similarity, a sensitivity sweep) or reported only through throughput and task accuracy. None chooses it by agreement with the dense verifier; SKIVE's schedule is the exact solution of Eq. 3 over a table built online from the model's own selections, and on LLaMA-3.1-8B it finds that anchor placement outweighs anchor count. The closest neighbor, Wang et al. (2025), justifies root reuse across nodes and layers by an average Jaccard overlap > 0.8 that hides the layer-wise dip; we count false acceptances and false rejections separately, test the node and layer reuses separately, and choose the layer set from the same objective.

4 Method: One Objective for Sparse Verification

SKIVE learns which layers need a fresh block selection by periodically measuring how well earlier selections serve later layers; an *online controller* (Algorithm 1) builds that table during decoding and solves the schedule on it. One objective, expected true positives as a function of the retained cache blocks, motivates the table and also sets the sparsity to operate at.

4.1 The Objective: Expected True Positives

At each layer sparse verification retains $K = \lfloor M(1-s) \rfloor$ of the M cache blocks as one set $\mathcal{S}$ shared by all N candidate queries the round verifies, and we want the set that maximizes expected TP. Let $\sigma_\ell(q_v, b)$ be the selector's score of block b for node v's query at layer ℓ, and define the layer's *importance* and the *mass* a set retains,

$$I_\ell(b) := \sum_{v=1}^{N} \bar{w}_v \, \sigma_\ell(q_v, b), \qquad m_\ell(\mathcal{S}) := \sum_{b \in \mathcal{S}} I_\ell(b), \qquad \mathbb{E}[\mathrm{TP}_\ell](\mathcal{S}) \;\approx\; \lambda_0 + \lambda \, m_\ell(\mathcal{S}), \quad (1)$$

where $\bar{w}_v$ is the drafter's path probability, discounted by depth and normalized over the tree, a free prior that v lies on the dense-accepted path. Both the layer schedule and the operating point rest on this approximation: over the sparsity range we use, a node's sparse-acceptance probability rises linearly in the importance mass the retained set keeps, with one common slope $\lambda > 0$ (derivation and empirical probes in Appendix B.1; no probed non-linear correction helps). Neither λ_0 nor λ depends on $\mathcal{S}$: the intercept is the same for every candidate set and the positive slope preserves their order, so maximizing expected true positives means maximizing retained mass, and within a layer the maximizing size-K set is the top-K blocks by I_ℓ, one weighted sum and one top-K into which any selector plugs. The across-node weighting has a small effect: scoring the root query alone is within 0.3 TP points and within noise end to end (Table 2 and §5.2); the results below are therefore insensitive to $\bar{w}_v$.

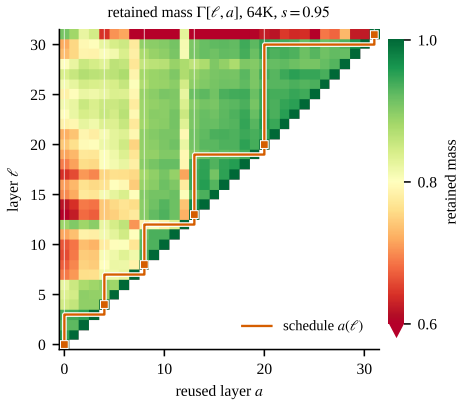
4.2 Observation: the Loss Comes from Reuse Across Layers

Root reuse bundles two shortcuts, one selection for all tree nodes and one for all layers, and Table 2 separates them: re-scoring the same root query at every layer recovers all $27/17/15$ true-positive points root reuse loses at 16K/32K/64K, while aggregating the tree's queries instead of the root adds less than half a TP point. Most of the TP loss comes from reuse across layers, and its cause is visible in the selections themselves. Figure 2a tabulates, for every pair of layers, how much of layer ℓ's importance mass layer a's selection retains. Layer 0's selection keeps 95% of layer 1's mass, 85% of layer 4's and 67% of layer 8's, and the largest drops are at a few depths (7–8, 12–13, 31). Reuse is cheap across most adjacent layers and expensive across a few: choosing the re-selecting layers is a scheduling problem, and Eq. 1 already scores a reused selection.

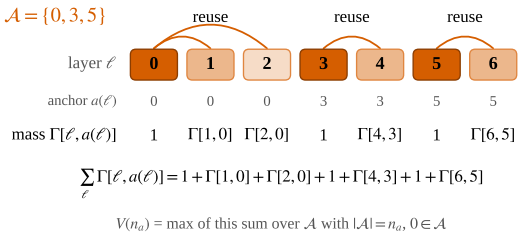
4.3 What the Objective Decides Across Layers: the Schedule

The same objective quantifies the cost of reusing a stale selection. Let $\mathcal{S}_a$ be the set layer a selects for itself; masses count unprotected blocks only, importances shifted by their minimum. Under the linearity step, the expected TP contribution of layer ℓ when it reuses layer a's selection is proportional to the *retained-mass ratio*

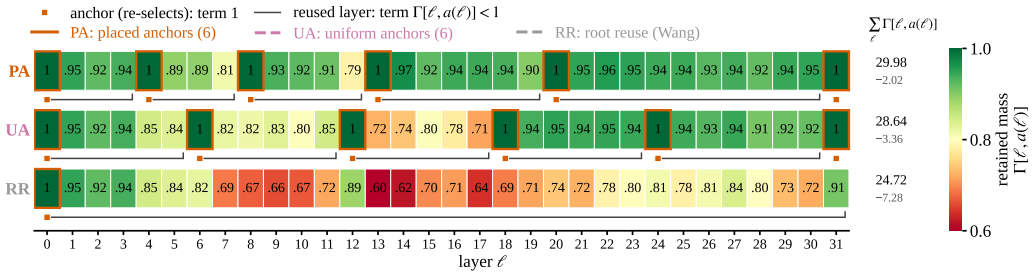
$$\Gamma[\ell, a] \;=\; \frac{m_\ell(\mathcal{S}_a)}{m_\ell(\mathcal{S}_\ell)} \;\in [0,1], \qquad a \le \ell, \quad (2)$$



(a) The retained-mass table $\Gamma[\ell, a]$ (Eq. 2), with the six anchors the solver places on it (squares) and each layer's reuse under that schedule (steps). Rows shift toward red where a reused selection goes stale; the largest drops, at layers 7–8, 12–13 and 31, are where the solver places the anchors.



(b) Eq. 3 on seven layers with anchors $\mathcal{A} = \{0, 3, 5\}$: an anchor re-selects and contributes 1; every other layer reuses the selection of its nearest preceding anchor $a(\ell)$ (arcs) and contributes $\Gamma[\ell, a(\ell)]$.



(c) The same sum on the table in (a), one cell per layer's term, for three schedules. The six placed anchors start a fresh selection where the previous one has gone stale and keep 29.98 of 32; six uniform anchors fall after the drops and keep 28.64; root reuse keeps 24.72.

Figure 2: **Which layers re-select, and what a schedule retains.** LLaMA-3.1-8B + EAGLE3 at 64K, s=0.95, 3,701 rounds. The count is held at six here; the controller chooses its own.

and $1 - \Gamma[\ell, a]$ is the *staleness* of that reuse. The best n_a-anchor schedule maximizes the mass the round retains, and we write that best value as

$$V(n_a) \;=\; \max_{|\mathcal{A}|=n_a,\; 0\in\mathcal{A}} \; \sum_{\ell=0}^{L-1} \Gamma[\ell, a(\ell)], \tag{3}$$

with $\mathcal{A}^\star(n_a)$ the schedule that attains it (Figure 2b, 2c). Eq. 3 weights every layer equally, a choice tested in Appendix C.2. Each layer's term depends only on its nearest preceding anchor, so Eq. 3 is a chain: sweeping layers in order while tracking which anchor is currently in use and how many have been placed solves it exactly in $O(L^2 n_a)$ time, about 0.3 ms at L=32, returning the best of all $\binom{L-1}{n_a-1}$ schedules. One pass returns the optimum $V(n_a)$ for every count at once, and the controller takes the count from that curve, adding anchors while the next one still recovers at least ϵ of one layer's mass,

$$n_a^\star \;=\; \min\{\, n \geq 3 : V(n{+}1) - V(n) < \epsilon \,\}, \qquad \epsilon = 0.15. \tag{4}$$

On the arXiv calibration documents (§5.1) it gives six anchors at 16K, seven at 32K and eight at 64K on LLaMA-3.1-8B (end to end at 64K, seven on most documents) and ten, eight and nine of 36 layers on Qwen3-8B. Measured TP supports these counts: at 16K and 32K it falls by about a TP point once the count drops below six, and at 64K it is flat from four anchors upward, so the extra anchors the rule keeps at 64K cost no quality; at these contexts any miscount is toward more anchors, not fewer.

Algorithm 1 Online layer schedule. One pass over a document's rounds; $\bar{\Gamma}$ persists across documents, so a fresh model starts from an empty table.

Require: layers L, sparsity s, full-round period R, average rate η, count threshold ϵ; persistent table $\bar{\Gamma}$
$\mathcal{A} \leftarrow \{0, \dots, L-1\}$ {first round scores every layer}
for round $t = 0, 1, 2, \dots$ of this document **do**
  verify: $\ell \in \mathcal{A}$ selects $\mathcal{S}_\ell \leftarrow \text{top-}K(I_\ell)$; $\ell \notin \mathcal{A}$ reuses $\mathcal{S}_{a(\ell)}$ {Eq. 1}
  if $\mathcal{A} = \{0, \dots, L-1\}$ **(a full-scoring round:** every $\mathcal{S}_\ell$ **is known) then**
    $\Gamma[\ell, a] \leftarrow m_\ell(\mathcal{S}_a)/m_\ell(\mathcal{S}_\ell)$ for $a \le \ell$ {Eq. 2, one matrix product}
    $\bar{\Gamma} \leftarrow \eta\,\bar{\Gamma} + (1-\eta)\,\Gamma$ {$\bar{\Gamma} \leftarrow \Gamma$ if empty}
    $V \leftarrow \text{DP}(\bar{\Gamma})$ {best retained mass at every anchor count}
    $n_a \leftarrow \min\{n \ge 3 : V(n{+}1) - V(n) < \epsilon\}$ {Eq. 4}
    $\mathcal{A}^\star \leftarrow \arg\max$ schedule at n_a {Eq. 3, by backtracking the same DP}
  end if
  $\mathcal{A} \leftarrow \{0, \dots, L-1\}$ if $(t{+}1) \bmod R = 0$ else $\mathcal{A}^\star$
end for
DP: sweep $\ell = 0 \dots L{-}1$ carrying $B[a, c]$, the best $\sum_{j \le \ell} \bar{\Gamma}[j, a(j)]$ with a the anchor in use and c anchors placed; layer ℓ either inherits ($B[a, c]$ += $\bar{\Gamma}[\ell, a]$) or anchors ($B[\ell, c{+}1] \leftarrow \max_a B[a, c] + 1$).

Anchor placement accounts for the later layers that reuse each selection: re-selecting within the round once a layer's own selection has gone stale is 1.0–1.3 TP points worse at equal cost (Appendix C.1).

Learning the table online. Building the table requires every layer's own selection, the cost the schedule avoids, so it is measured periodically; Γ varies slowly with the model and the document. The online controller (Algorithm 1), at a fixed sparsity, scores every layer on the first round after prefill (a *full-scoring round*: every layer re-selects, attention stays sparse, and no dense pass is involved) and every $R = 32$ rounds after, folds each table into a running average $\bar{\Gamma} \leftarrow 0.9\,\bar{\Gamma} + 0.1\,\Gamma$, re-solves Eqs. 3–4, and keeps $\bar{\Gamma}$ across documents. It needs only the layer count and what it decodes, and costs $n_a + (L - n_a)/R$ scored layers per round (6.8 at n_a=6, L=32) plus about 0.3 ms for the table and the solve. The running average and the periodic full-scoring round are both needed: a greedy rule that re-scores the stalest layers each round oscillates, because staleness is measured against the anchors currently in use (Appendix C.2).

Where the anchors land. Holding the count at six to isolate placement, the schedule solved on LLaMA-3.1-8B's calibration table is at the drops of Figure 2a: $\{0, 5, 8, 14, 20, 31\}$ at 16K, $\{0, 5, 8, 13, 18, 31\}$ at 32K and $\{0, 4, 8, 13, 20, 31\}$ at 64K; layers 0, 8 and 31 are anchors at every context and the others shift by one or two layers as the context grows, so the profile is set by the model and adjusted by the context. Six anchors placed by Eq. 3 reproduce per-layer TP (Table 9), while six uniformly spaced anchors lose 4.8 TP points at 64K (Table 9); online, the controller held at six converges to these anchors within about ten documents (Appendix C.2).

4.4 What the Objective Decides Against Cost: the Operating Point

Dense SD measures throughput as accepted tokens per round time, τ / T_{round}. We replace the numerator with true positives alone (§2.1), normalized by the dense verifier's own candidate count into the rate $\text{TP}(s) = \text{TP}_{\text{count}}(s)/(\tau_{\text{dense}} - 1) \in [0, 1]$, so TP=1 means sparse accepts every candidate dense would; $\text{FP}(s)$ counts the tokens sparse accepts beyond the prefix both accept, on the same denominator; bonus tokens are not candidates and are excluded. With sparsity s as the within-layer budget and n_a as the across-layer budget, round time decomposes as

$$T_{\text{round}}(s, n_a) = \underbrace{T_{\text{draft}} + T_{\text{fixed},0}}_{\beta_0} + c\,n_a + \underbrace{T_{\text{kv}}}_{\alpha}(1 - s), \qquad \Phi(s, n_a) = \frac{\text{TP}(s, n_a)}{\beta_0 + c\,n_a + \alpha(1 - s)}, \tag{5}$$

where c is the cost of scoring and selecting blocks at one layer (0.41/0.78/1.78 ms at 16K/32K/64K with EAGLE3, $\beta_0 = 107/172/234$ ms; 0.50/0.82/1.59 ms and $\beta_0 \approx 40$ ms with the Arctic LSTM drafter; α in Appendix A.1), so round time is linear in scored layers per round. Φ is proportional to correctly accepted tokens per second. Sparse verification pays only when the saved read αs exceeds the scoring cost $c\,n_a$ it adds; the test is $\Phi(s^\star) > \Phi(0)$, and it fails when α is small relative to β_0 (on the H200 it fails at batch size 1 and passes as the per-round KV read grows with batch size; §6).

Table 2: **What the objective counts.** TP and FP rates at the EAGLE3 operating points (LLaMA-3.1-8B + EAGLE3, arXiv calibration, dual-run protocol, n=30; MeanPool) by schedule; the root-query row isolates the node axis (§5.1).

	16K (s=0.85)		32K (s=0.95)		64K (s=0.97)	
Schedule	TP	FP	TP	FP	TP	FP
Every layer (32)	0.966	0.035	0.955	0.045	0.974	0.023
Online anchors	0.957	0.040	0.953	0.044	0.972	0.029
Root query, every layer	0.963	0.039	0.952	0.045	0.973	0.023
Root reuse (Wang)	0.693	0.194	0.783	0.186	0.821	0.178

Where Φ peaks. Setting $\partial\Phi/\partial s = 0$ gives $\mathrm{TP}'(s^*)/\mathrm{TP}(s^*) = -\alpha/(\beta_0 + c\,n_a + \alpha(1 - s^*))$: relative TP decay balances relative time saved, so a flatter TP curve and a larger α relative to the fixed cost both raise s^*. The table fixes the count (Eq. 4). Given the count, Φ needs two offline inputs that the schedule never uses: $\mathrm{TP}(s, n_a)$, measured on a small arXiv calibration set, and (β_0, c, α), fitted from timed rounds. With EAGLE3 it peaks at $s^* = 0.90, 0.95, 0.97$ at 16K/32K/64K; the tables use 0.85 at 16K, within 1% of $\Phi(s^*)$. As a check, searching Φ offline over count and sparsity together also gives six anchors, so the count rule agrees with the cost-aware optimum, and the online controller, paying for the count it chooses, is within 2% of it (Figure 3); with the Arctic LSTM drafter $s^* = 0.95, 0.97, 0.99$ (Appendix B.4). Above $s{\approx}0.93$ the curve is flat at 16K and 32K (the Table 3 points are within 3–5$\%$ of $\Phi(s^*)$) and tilts at 64K, where s=0.99 is 12% higher and measures 13% faster. Φ does not see quality: per-round TP misses loss that accumulates across rounds (§5.3), and at s=0.99 at 64K both schedules lose 6–7$\%$ ROUGE-L against AR, though not significantly at n=50, so the headline table uses the quality-constrained point on that plateau.

5 Experiments

5.1 Setup

Stacks. LLaMA-3.1-8B-Instruct on one RTX 6000 Ada (48 GB), HuggingFace (HF) SDPA verification (Appendix A.3), with two drafters: EAGLE3 (Li et al., 2025), whose TACC collapses with context, for the agreement measurements, and Snowflake's Arctic LSTM Speculator extended to a $[6, 2, 2, 1]$ tree, whose TACC stays flat at 2.8–2.9 from 4K to 64K (Appendix A.4), for the throughput study, where a stable drafter keeps the verification saving visible. We replicate on Qwen3-8B+EAGLE3 (Appendix D.2). **Data and metrics.** $\mathrm{TP}(s)$ and the *calibration table* (§4.3) come from arXiv summarization; quality and throughput from GovReport (Huang et al., 2021) and arXiv. We report the paired relative change in ROUGE-L against gold summaries (ΔRL, vs. AR), TACC, decode throughput after prefill, and the TP/FP rates of §4.4. $\mathrm{TP}(s)$ is measured by a *dual-run protocol*: dense and sparse verifiers run on the same densely written prefix each round and the candidates both accept are counted (EAGLE3 drafter, n=30 arXiv documents), so accumulated sparse-write error is excluded (§5.3). Seed 42. **Methods.** Baselines: dense AR, dense SD, AR with a sparse cache at the same keep rate and no drafting (*AR + sparse cache*), and root reuse (Wang et al., 2025), which scores the root query at layer 0 and reuses the set everywhere (*Root reuse (Wang)*). All sparse methods share one keep rate per context, isolating the schedule. SKIVE uses MeanPool and the online controller of §4.3, which chooses count and placement from its own table (*Online anchors*). Two ablations name which layers re-select instead: *Every layer (32)* scores all layers (n_a=L, the objective without a schedule) and *Uniform anchors (6)* spaces six anchors evenly; *Placed anchors (6)* are the six Eq. 3 places on the calibration table before the run, the controller's offline reference. The node axis is isolated by *Root query, every layer*, the variant of Wang et al. (2025) that re-scores the root query at every layer with no node aggregation, in Table 2.

5.2 True Positives by Schedule

$\mathrm{TP}(s)$ is *schedule-dependent* (Figure 3a): per-layer selection holds $\mathrm{TP} \geq 0.955$ at the operating points of Table 2, while root reuse is at 0.69–0.82, with FP at 0.18–0.19 against 0.02–0.05. The online controller reproduces per-layer TP and FP to within one TP point at every operating point with a quarter of the scoring work, six placed anchors with a fifth (Appendix C.1).

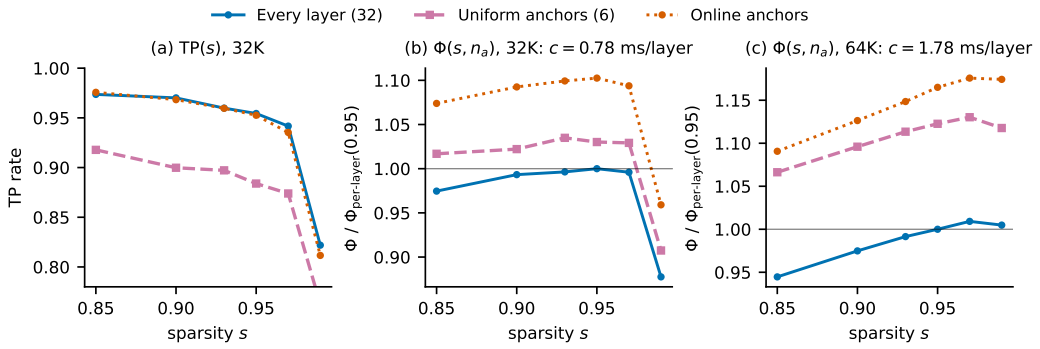


Figure 3: **Schedule sweep** (LLaMA-3.1-8B + EAGLE3, arXiv calibration, n=30; Φ with the EAGLE3 round times of §4.4). (a) $\mathrm{TP}(s)$ at 32K: the online controller tracks per-layer selection; six uniformly spaced anchors are lower. (b),(c) $\Phi(s, n_a)$ relative to per-layer at s=0.95 at 32K and 64K.

Table 3: **LLaMA-3.1-8B-Instruct + Arctic LSTM tree drafter on GovReport and arXiv.** Decode speedup over AR and paired ΔRL vs. AR (paired t-test: * $p < 0.05$, † $p < 0.01$; unmarked cells are not significant); n=50 paired, 128 tokens, one RTX 6000 Ada; AR measured in the same run at each context and dataset. Each context uses the sparsest grid point at which the online controller's paired ΔRL against AR on these GovReport documents is within 5% and not significant (s=0.80 at 4K, 0.85 at 16K, 0.95 at 32K and 64K), and the arXiv rows are measured at the same points; the online controller scores 7.1–8.5 layers per round. Dense SD's ΔRL is FP16 reduction-order noise. The aggressive points and dense-refresh rows are in Appendices B.4 and E.2.

	4K		16K		32K		64K	
Method	×AR	ΔRL	×AR	ΔRL	×AR	ΔRL	×AR	ΔRL
GovReport								
AR (tok/s)	29.8		22.5		15.2		8.9	
Dense SD	1.75	+0.0%	1.21	+0.4%	1.20	+0.2%	1.15	+0.2%
AR + sparse cache	0.70	−1.8%	0.96	−2.1%	1.34	−6.7%	1.56	−2.1%
Root reuse (Wang)	2.06	−7.9%†	2.71	−11.9%†	4.10	−13.7%†	6.80	−11.0%†
Every layer (32)	1.53	−5.3%†	2.07	−3.2%	2.87	−5.5%	3.44	−1.7%
Online anchors	1.88	−4.7%	2.55	−2.4%	3.74	−4.7%	5.51	−4.8%
arXiv								
AR (tok/s)	30.1		22.3		15.1		8.8	
Dense SD	1.59	−0.2%	1.04	+0.5%	1.00	−0.5%	0.95	−0.1%
AR + sparse cache	0.70	−1.5%	0.95	+0.2%	1.34	−3.3%	1.59	−0.6%
Root reuse (Wang)	1.90	−14.3%†	2.47	−13.1%†	3.70	−13.6%†	5.85	−15.4%†
Every layer (32)	1.38	−5.9%*	1.83	−1.3%	2.46	−0.5%	2.89	−1.6%
Online anchors	1.67	−1.9%	2.19	0.0%	3.22	−1.3%	4.59	−2.6%

The node axis. Table 2 separates the two reuses of root reuse. Re-scoring the root query at every layer with no node aggregation recovers 27/17/15 TP points at 16K/32K/64K; aggregating the tree nodes' queries adds 0.1–0.3 in aggregate, concentrated at tree depths 2–3 where it adds 0.6–1.4 TP points ($p < 0.01$, paired; Appendix B.2), and end to end the two are within noise in ROUGE-L at all four contexts (−4.0/ − 4.7/ − 5.2/ − 3.0% against −5.3/ − 3.2/ − 5.5/ − 1.7% vs. AR, GovReport runs of Table 3). Under a fixed schedule the four selectors are within a few percent of one another (Appendix B.3); we use the cheapest, MeanPool, everywhere else.

5.3 End-to-End Results

With the drafter stable (Table 3), dense SD reaches only 1.15× AR at 64K because verification is bound by the KV read; sparsity sets how much of it is read and the schedule how often selections are recomputed. Per-layer selection reaches 3.44× AR at a non-significant ROUGE-L change but

scores all 32 layers. The online controller, choosing its own count with no calibration data, reaches $5.51\times$ at a ROUGE-L not significantly different from per-layer's (paired $\Delta = -0.005 \pm 0.009$, two standard errors) and at the same acceptance (TACC 2.87 against 2.88), so its $1.6\times$ advantage comes entirely from scoring 8.5 layers per round instead of 32. Root reuse is fastest ($6.80\times$) but loses 8–15% ROUGE-L everywhere ($p < 0.01$). AR + sparse cache loses 2–7% ROUGE-L on GovReport and up to 3% on arXiv at the same keep rate; the controller's 0–5% is inside that band except at 4K and 64K on GovReport, where verification adds about 3 percentage points to the cache's own loss (aggressive points: Appendix B.4).

Other drafters, models and tasks. With EAGLE3 the online controller gives $1.79/1.65/1.38\times$ AR at 16K/32K/64K at $-0.8/-3.5/-5.8\%$ ROUGE-L, beating dense SD everywhere; the smaller 64K gain is a drafter limit, as Φ predicts (Appendix B.4). On Qwen3-8B the online controller, with no calibration, picks ten, eight and nine anchors and trails per-layer by 0.6–1.2 TP points; there uniform anchors do as well, since its staleness is spread evenly across depth (Appendix D.2). On retrieval, multi-hop question answering (QA) and code the cost is larger (Appendix D.3): needle-retrieval pass rates fall to 92–100% against AR's 100%, token-level F_1 (token-F1) on the LongBench multi-hop sets (HotpotQA, 2WikiMQA, MuSiQue) drops by up to 0.03 absolute, and edit similarity on LongBench code completion (LCC) by up to 0.05; the largest losses are at 8K, where the KV read is small and sparsity adds little over dense SD.

Where the residual loss comes from. Much of the residual is the sparse cache's own rather than verification's (Table 3), and it is dataset-dependent, about 5% on GovReport against 1–3% on arXiv. Per-round agreement does not fully predict it: at 64K per-round TP is still 0.956 at s=0.99 while ROUGE-L falls 6–9% for both schedules and AR + sparse cache, because sparsity also touches the K/V entries each accepted token writes, and that error accumulates across rounds. In a 32K probe, a densely written cache avoids 93–96% of confident flips (Appendix E). On the online schedule a periodic dense refresh of those entries gives no significant gain at 64K and costs 14–43% of the speed, so no headline configuration uses it (Appendix E.2).

6 Production-Stack Validation: SGLang

We ported SKIVE into SGLang's EAGLE3 pipeline (stock drafter, s=0.9) as fused scoring and top-K at the anchor layers and a bespoke sparse tree-attention kernel in place of FlashInfer's verify attention, under a static schedule (the six-anchor schedule of §4.3 at 64K) rather than the controller itself. On an H200 (AR 116.5 tok/s at 64K) we vary batch size at 64K, where the cost model changes sign. Selection cost and the KV read it avoids both scale with the number of *sequences* while weight reads do not, so the avoidable share of the verify read rises from 36% at B=1 to 82% at B=8, and Eq. 5 predicts a loss at B=1 turning into a gain as B grows. Measured, SKIVE trails dense tree verification by 19% at B=1 and leads by 14%, 38% and 52% at B=2, 4 and 8 (each 90% interval excludes parity); against AR it is 0.78–$1.21\times$, because on this card AR is fast next to the port's fixed per-round cost; on the RTX 6000 Ada of our HF runs the same port reaches $2.06\times$ the CUDA-graph AR at 32K and B=4 (Table 15). Per-layer selection never crosses parity (0.82–1.00 against dense tree), so at $B \geq 2$ six placed anchors add 26–52 percentage points over per-layer selection. Quality is measured single-stream (ROUGE-L within $\pm 7\%$ of AR, n=20; table, figure and caveats in Appendix F).

7 Conclusion

One expected-true-positive objective decides which layers re-select the cache: the anchor schedule that maximizes the importance mass reused selections retain, solved exactly over a table the model builds from its own selections; the same solve fixes the count. On LLaMA-3.1-8B where the anchors sit matters more than how many, and the schedule needs no calibration data: with the Arctic LSTM drafter it reaches $5.51\times$ AR at 64K against $3.44\times$ for per-layer selection, at a ROUGE-L not significantly different from it. Much of the remaining loss is the sparse cache's, not the verifier's, and it is dataset-dependent. *Limitations.* SKIVE is lossy, and on retrieval, multi-hop QA and code completion the cost exceeds summarization's (Appendix D.3), and on Qwen3-8B uniform placement already suffices (§5.3). Speedups are single-stream HF numbers except §6, on two 8B-class model families at $B \leq 8$.

Reproducibility statement

Appendix A describes the implementation, the round-time parameters, the EAGLE3 RoPE configuration fix, the attention backends and the LSTM tree-drafter extension; Appendix D.2 the Qwen3-8B replication; and Appendix F the SGLang port's configuration. The online controller (Algorithm 1), the patched verifier, the drafter code and the scripts that generate every table and figure are provided as supplementary material, together with the retained-mass tables and per-round logs the figures are drawn from.

AI use statement

We used Claude (Anthropic) to help draft and edit text, to write experiment and figure scripts, to run and monitor experiments, and to check the manuscript's numbers against result files and the bibliography against arXiv records, all under our direction. The research questions, the method and the experimental design are ours, and no data were generated or labeled with AI. We reviewed and tested the AI-assisted code and checked every number, statistical test and conclusion ourselves, and we take responsibility for the paper.

References

T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *ICML*, 2024.

C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper. Accelerating large language model decoding with speculative sampling. *arXiv:2302.01318*, 2023.

L. Huang, S. Cao, N. Parulian, H. Ji, and L. Wang. Efficient attentions for long document summarization. In *NAACL*, 2021.

Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023.

C. Lin, J. Tang, S. Yang, H. Wang, T. Tang, B. Tian, I. Stoica, S. Han, and M. Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning. In *NeurIPS*, 2025.

Y. Li, F. Wei, C. Zhang, and H. Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *ICML*, 2024.

Y. Li, F. Wei, C. Zhang, and H. Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *EMNLP*, 2024.

Y. Li, F. Wei, C. Zhang, and H. Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. *arXiv:2503.01840*, 2025.

X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R. Y. Y. Wong, A. Zhu, L. Yang, X. Shi, C. Shi, Z. Chen, D. Arfeen, R. Abhyankar, and Z. Jia. SpecInfer: Accelerating generative large language model serving with tree-based speculative inference and verification. In *ASPLOS*, 2024.

J. Yao, K. Chen, K. Zhang, J. You, B. Yuan, Z. Wang, and T. Lin. DeFT: Decoding with flash tree-attention for efficient tree-structured LLM inference. In *ICLR*, 2025.

J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *ICML*, 2024.

J. Wang, J. Tan, Y. Hu, J. Qin, Y. Sun, Y. Xie, X. Cai, J. Li, and M. Zhang. Accelerate speculative decoding with sparse computation in verification. *arXiv:2512.21911*, 2025.

Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. Wang, and B. Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *NeurIPS*, 2023.

W. Wu, Z. Pan, C. Wang, L. Chen, Y. Bai, T. Wang, K. Fu, Z. Wang, and H. Xiong. TokenSelect: Efficient long-context inference and length extrapolation for LLMs via dynamic token-level KV cache selection. In *EMNLP*, 2025.

L. Yang, Z. Zhang, Z. Chen, Z. Li, and Z. Jia. TidalDecode: Fast and accurate LLM decoding with position persistent sparse attention. In *ICLR*, 2025.

G. Xiao, J. Tang, J. Zuo, J. Guo, S. Yang, H. Tang, Y. Fu, and S. Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *ICLR*, 2025.

H. Sun, L.-W. Chang, W. Bao, S. Zheng, N. Zheng, X. Liu, H. Dong, Y. Chi, and B. Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *ICML*, 2025.

Z. Tan, X. Zhang, C. Hu, J. Peng, and K. Xia. SpecPV: Improving self-speculative decoding for long-context generation via partial verification. *arXiv:2512.02337*, 2025.

Y. Zhao, J. Tang, K. Zhu, Z. Ye, C.-C. Chang, C. Lin, J. Park, G. Xiao, M. S. Abdelfattah, M. Gao, B. Kasikci, S. Han, and I. Stoica. Accelerating large-scale reasoning model inference with sparse self-speculative decoding. *arXiv:2512.01278*, 2025.

P. Yang, C. Du, F. Zhang, H. Wang, T. Pang, C. Du, and B. An. LongSpec: Long-context lossless speculative decoding with efficient drafting and verification. In *ACL*, 2025.

H. Sun, Z. Chen, X. Yang, Y. Tian, and B. Chen. TriForce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv:2404.11912*, 2024.

R. Sadhukhan, J. Chen, Z. Chen, V. Tiwari, R. Lai, J. Shi, I. E.-H. Yen, A. May, T. Chen, and B. Chen. MagicDec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. *arXiv:2408.11049*, 2024.

Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, Y. Dong, J. Tang, and J. Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *ACL*, 2024.

R. Tiwari, H. Xi, A. Tomar, C. Hooper, S. Kim, M. Horton, M. Najibi, M. W. Mahoney, K. Keutzer, and A. Gholami. QuantSpec: Self-speculative decoding with hierarchical quantized KV cache. In *ICML*, 2025.

W. Lee, J.-J. Chen, X. Lin, P.-S. Wang, C.-C. Chang, C.-C. Yang, N.-C. Huang, G. L. Zhang, and K.-C. Wu. Dustin: Draft-augmented sparse verification for efficient long-context generation with speculative decoding. In *ICML*, 2026.

M. Lee, A. Ramachandran, S. Kundu, and T. Krishna. Polestar: Drift-aware cache calibration and token commitment for efficient inference of diffusion LLMs. *arXiv:2607.14107*, 2026.

S. Oh, N. Cao, Y. Kim, H. Jung, H. Ahmad, S. Bae, and S.-Y. Yun. Bastion: Budget-aware speculative decoding with tree-structured block diffusion drafting. *arXiv:2605.29727*, 2026.

J. Song, X. Wang, H. Wang, X. Lei, B. Shi, S. Han, E. Yang, X.-W. Chang, and L. Ai. MARS: Unleashing the power of speculative decoding via margin-aware verification. *arXiv:2601.15498*, 2026.

Z. Wang, Z. Zhong, N. Shen, Y. Zhou, R. Gu, and S. Zhong. SSV: Sparse speculative verification for efficient LLM inference. *arXiv:2605.19893*, 2026.

J. Yao, S. Shen, K. Du, S. Feng, D. Seo, R. Zhang, Y. Huang, Y. Liu, S. Lu, and J. Jiang. VeriCache: Turning lossy KV cache into lossless LLM inference. *arXiv:2605.17613*, 2026.

Y. Yue, Y. Xue, and J. Huang. Vegas: Self-speculative decoding with verification-guided sparse attention. In *ICML*, 2026.

Y. Bai, Q. Dong, T. Jiang, X. Lv, Z. Du, A. Zeng, J. Tang, and J. Li. IndexCache: Accelerating sparse attention via cross-layer index reuse. *arXiv:2603.12201*, 2026.

Y. Sun, Y. Zhang, L. Dong, J. Wang, and F. Wei. You only index once: Cross-layer sparse attention with shared routing. *arXiv:2606.06467*, 2026.

T. Wang, G. Rattihalli, A. Dhakal, J. Li, Z. Ren, D. Milojicic, and L. Shangguan. Predict, reuse, and repair: Accelerating dynamic sparse attention for long-context LLM decoding. *arXiv:2606.30389*, 2026.

Z. Cai, Y. Zhang, B. Gao, Y. Liu, T. Liu, K. Lu, W. Xiong, Y. Dong, B. Chang, J. Hu, and W. Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv:2406.02069*, 2024.

Z. Wang and S. Gan. SqueezeAttention: 2D management of KV-cache in LLM inference via layer-wise optimal budget. *arXiv:2404.04793*, 2024.

Y. Feng, J. Lv, Y. Cao, X. Xie, and S. K. Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. *arXiv:2407.11550*, 2024.

A Implementation and Reproducibility

The appendix follows the questions of the main text. Appendix A gives the implementation, the convention behind every $\times$AR ratio, the round-time parameters of Eq. 5, and the two drafters; Appendix B verifies the objective's assumption and reports the node axis, the selector sweep and the Φ-chosen operating points; Appendix C is the layer schedule (count, placement, the online controller); Appendix D is the end-to-end evidence on other drafters, models and tasks (EAGLE3, Qwen3-8B, retrieval, QA and code), with the schedule sweeps on both models; Appendix E locates the residual quality loss and tests a periodic dense refresh against it; Appendix F is the SGLang port.

Keep rates and populations. Keep rates are fixed per table rather than per cell: the Arctic LSTM headline table steps s=0.80/0.85/0.95/0.95 from 4K to 64K by the rule in its caption, and the EAGLE3 tables step keep $15/5/3\%$ with context, so cells compare down a column and not across drafters. GovReport has over 500 documents that pass the length filter (at least $0.8\times$ the context) at 32K but fewer than 60 at 64K, so every 64K cell draws its n=50 from nearly the whole population, which caps the paired statistics there (Appendix E.2).

A.1 Ratios and Round-Time Parameters

Absolute tok/s is specific to each implementation (its generation length and decode loop), so the quantity to compare across Tables 3, 10 and 15 is the ratio to the AR baseline measured in the same implementation on the same GPU. The SGLang figures follow the same convention: they are against SGLang's own AR baseline, not against either HF implementation.

Table 4 gives the measured round-time parameters behind the saveable share of §2.2 and the α of Eq. 5 on that stack; the EAGLE3 end-to-end table itself is in Appendix D.1.

Table 4: **Round-time parameters.** Measured on EAGLE3, HF+SDPA, LLaMA-3.1-8B, RTX 6000 Ada. T_{fixed} (sparsity-independent verify cost) and T_{kv} (dense KV read) are a linear fit of verify time against keep rate $(1-s)$ over 11 sparsity levels (190 timed rounds per level at 16K/32K, 95 at 64K; $R^2 = 0.92$–0.93); T_{draft} is inferred from measured throughput and acceptance, not timed directly (extrapolated at 64K). $\alpha = T_{\text{kv}}$ is the α of Eq. 5; $\beta = T_{\text{draft}} + T_{\text{fixed}}$ was timed with every layer scored, so it contains $32c$, whereas the β_0 and c of Eq. 5 are fitted from end-to-end round times. Saveable $= T_{\text{kv}}/T_{\text{dense}}$.

Context	T_{draft} (ms)	T_{fixed} (ms)	β (ms)	T_{kv} (ms)	T_{dense} (ms)	Saveable
16K	58	42	100	35	135	26%
32K	93	50	143	94	237	40%
64K	160	77	237	196	433	45%

A.2 RoPE Configuration Bug in EAGLE3

The public EAGLE3 checkpoint for LLaMA-3.1-8B (`yuhuili/EAGLE3-LLaMA3.1-Instruct-8B`) stores stale RoPE configuration values in its `config.json`: `max_position_embeddings=2048` and `rope_theta=10000`. The base LLaMA-3.1-8B model uses 131,072 and 500,000 respectively.

The shipped configuration therefore runs the draft head with `rope_theta` $= 10{,}000$ while the base model, whose hidden states the head consumes, uses $500{,}000$, and beyond the 2K positions the head was trained on its positional encodings diverge from the base model's: from 4K context onward almost every draft token is rejected.

Table 5 measures both configurations under one protocol (LLaMA-3.1-8B + EAGLE3, dense SD, arXiv, n=10 documents of at least $0.8\times$ the context, 200 generated tokens, the Table 10 tree).

The fix is to override the draft model's RoPE configuration with the base model's values (`max_position_embeddings=131072`, `rope_theta=500000`). The override is a trade: below 4K the shipped values accept 0.1–0.6 more tokens per round, because the head was trained at θ=10^4, and the override gives that up to hold 3.1–3.2 from 4K to 16K where the shipped configuration falls to 1.3–1.6.

Table 5: **The RoPE bug.** LLaMA-3.1-8B + EAGLE3 acceptance (TACC, bonus token included) with the shipped RoPE configuration and with the base model's values, one protocol for both columns (n=10, arXiv, dense SD, 200 tokens). ROUGE-L is identical between the columns in every row (dense SD is lossless); only acceptance changes. Prompts at 2K are 1.2–2K tokens, so generation stays inside the trained positions.

Context	TACC (shipped config)	TACC (override)
512	2.42	2.28
1024	3.76	3.19
2048	3.84	3.28
4096	1.63	3.10
8192	1.37	3.15
16384	1.27	3.21

Any reproduction using the public EAGLE3 checkpoint at contexts beyond 2K will observe almost no speedup from speculative decoding unless the configuration is corrected. All EAGLE3 baselines and SKIVE measurements in this paper use the corrected configuration; a flag in the supplementary code restores the shipped values for this diagnostic only. We do not know whether prior work used the public checkpoint as shipped or applied an equivalent fix.

A.3 Attention Backend Discussion

Tree verification is not a standard single-query causal-attention call: it uses a tree-causal mask over N candidate nodes and, in SKIVE, a sparse KV block set chosen for that verification event. For this mechanism study we therefore verify on the PyTorch scaled-dot-product-attention (SDPA) / eager path, which exposes the mask and the block schedule directly and lets every schedule share one execution path, keeping the comparison on the selector and the schedule rather than on backend-specific kernel support. Dense prefill uses FlashAttention, to fit 64K contexts. For verification, production systems should fuse SKIVE's block choices into native tree/sparse-attention kernels; DeFT (Yao et al., 2025) shows the complementary direction, scheduling exact tree attention once the kernel owns the tree. SKIVE decides which KV blocks such a sparse verifier reads and at which layers it re-selects.

A.4 Arctic LSTM as a Tree Drafter

The Arctic LSTM Speculator (`Snowflake/Arctic-LSTM-Speculator-Llama-3.1-8B-Instruct`) is a single-step recurrent cell trained with n_{predict}=3: three trained $(\text{proj}, \text{head})$ index pairs producing predictions at depths 0, 1, 2 (token positions t+1, t+2, t+3). It takes $(h_t, \text{token}_{t+1})$ and emits the next state plus vocab logits.

To produce a tree, we run the cell in batched form along depth. At every depth, parent states pass through one cell call, the per-parent vocab logits are top-k branched, and parent states are duplicated k times (`repeat_interleave`) so each child enters the next depth carrying its parent's state. A depth-D tree therefore costs D batched cell calls; for the schedule $[6, 2, 2, 1]$ this expands to 6+12+24+24=66 nodes in four calls.

For depths beyond Arctic's n_{predict}=3, we reuse the $(\text{proj}{=}1, \text{head}{=}2)$ pair the speculator already applies at depth 2. The projection appears at both trained continuation depths so its weights are in distribution; the head was trained to emit t+3 logits and is the only piece used out of distribution. With one rollout depth, the $[6, 2, 2, 1]$ tree accepts 2.81/2.83/2.87/2.87 tokens per round (bonus included) at 4K/16K/32K/64K under the online controller in the runs of Table 3. We use $[6, 2, 2, 1]$ throughout; nearby shapes such as $[6, 3, 2, 1]$ and $[8, 2, 2, 1]$ are within measurement noise. Verify q_len stays under 128, which is the threshold at which the patched `LlamaAttention` short-circuits to a prefill path that ignores the tree mask.

Inline bonus token

Each speculative-decoding round emits a bonus token (the verifier's argmax at the deepest accepted node). Its vocab id is known when verify finishes, but its KV-cache slots are unpopulated. We

populate them by prepending the bonus as the first query of the next round's verify forward; q_len becomes 67 and no separate bonus forward is needed. The drafter then receives the captured leaf hidden h_p rather than the post-bonus hidden h_{p+1}. The cell at depth 0 combines this hidden with the bonus embedding before the gates, so subsequent depths run the standard recurrence. Feeding the drafter the pre-bonus hidden costs roughly 10–17% TACC; removing the separate bonus forward, which is memory-bound by the model's $\sim$15 GB weight read at long context, more than pays for it. Dense SD, root reuse and SKIVE in Table 3 all run with the same inline bonus.

B THE OBJECTIVE: ASSUMPTION AND SWEEPS

B.1 THE OBJECTIVE'S ASSUMPTION, AND ITS VERIFICATION

From expected TP to a top-K rule. A node contributes one true positive only when dense would have accepted it and sparse verification, given $\mathcal{S}$, also accepts, so $\mathbb{E}[\text{TP}](\mathcal{S}) = \sum_v \pi_v \Pr(v \text{ accepted sparse} \mid \mathcal{S})$ with $\pi_v = \bar{w}_v \propto p_v/(1+d_v)$, the depth-discounted path prior of Eq. 1 normalized over the tree. The one assumption is that the second factor is locally linear in the retained score mass,

$$\Pr(v \text{ accepted sparse} \mid \mathcal{S}) \;\approx\; \lambda_0 \;+\; \lambda\, m_v(\mathcal{S}), \qquad m_v(\mathcal{S}) \;:=\; \sum_{b\in\mathcal{S}} \sigma(q_v, b), \tag{6}$$

with one intercept and one slope shared across nodes in the operating regime. Substituting and collecting terms by block gives Eq. 1: the sum splits into a term independent of $\mathcal{S}$ and $\lambda \sum_{b\in\mathcal{S}} I_\ell(b) = \lambda\, m_\ell(\mathcal{S})$, so for $\lambda > 0$ the maximizing size-K set is the top-K blocks by I_ℓ, and at a reused layer the same step scores a reused selection by the mass it retains (Eq. 2). The rest of this subsection tests Eq. 6.

Does acceptance follow retained mass? The test is that the per-node sparse-accept probability is locally linear in the retained score mass $m_v(\mathcal{S}) = \sum_{b\in\mathcal{S}} \sigma(q_v, b)$ (Eq. 6). If that dependence were meaningfully non-linear, the drafter path prior p_v would need a data-driven correction factor $\rho(s_v)$:

$$\bar{w}_v \;\propto\; \underbrace{p_v}_{\text{path prior}} \cdot \underbrace{\rho(s_v)}_{\text{non-linear correction}}, \tag{7}$$

and we ask whether any $\rho(\cdot)$ beats the uncorrected baseline $\rho(s_v) \equiv 1$, the form Eq. 1 selects (*Linear* in Table 6). The three probe families are concentration, a budget threshold near the cutoff $\sigma_{(K)}$, and score-row entropy. Two control rows: NO PRIOR drops p_v, so any lift there must come from ρ, and Root-only (the root query re-scored at every layer, the *Root query, every layer* row of Table 2) is a per-layer-scoring lower bound.

Table 6: **Non-linear corrections to the per-node weight** (LLaMA-3.1-8B + EAGLE3). $\Phi^\star$ ($\times 10^3$) for each probe family $\rho(s_v)$, and its change against the root query re-scored at every layer ($\Delta\Phi_{\text{root}}$) and against the linear form Eq. 1 selects ($\Delta\Phi_{\text{lin}}$).

Variant	$\rho(s_v)$ probes …	$\Phi^\star \times 10^3$	$\Delta\Phi_{\text{root}}$	$\Delta\Phi_{\text{lin}}$
Root-only	—	9.237	—	−0.72%
Linear (Eq. 1)	1 *(no non-linear correction)*	**9.304**	**+0.72%**	—
Peakedness	concentration: $(\max - \text{med})/\sigma$	9.297	+0.64%	−0.07%
Budget-aware	threshold: $(s_{(K)} - s_{(K+1)})/\sigma$	9.300	+0.68%	−0.04%
Negentropy	entropy: $\log M - H(\text{softmax}\, s_v)$	9.304	+0.72%	0.00%
No prior	ρ alone, no p_v	9.239	+0.02%	−0.69%

$\Phi^\star$ in units of 10^{-3}/ms; "Root-only" uses only s_0 (per-layer scoring, no node aggregation).

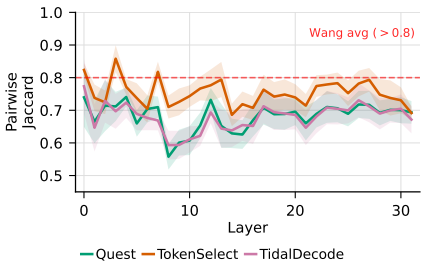
Protocol: arXiv calibration, MeanPool selector, 16K context, n=30, sparsity grid $\{0.0, 0.25, 0.50, 0.75, 0.85, 0.88, 0.90, 0.93, 0.94,\ 0.95, 0.96, 0.97, 0.99\}$; $s^\star = \arg\max_s \Phi(s) = \text{TP}(s)/(\beta + \alpha(1-s))$ from Table 4.

Reading the table. The path prior is the only node signal that helps, and it helps by 0.7%: NO PRIOR is indistinguishable from Root-only ($\Delta\Phi = +0.02\%$; $+0.25$ TP points at $s^\star$), so the score row alone cannot order nodes whatever non-linear shape we give ρ; in Eq. 1 the path prior p_v supplies that per-node ordering. No non-linear correction beats the linear baseline; the lift is $\leq 0.07\%$ across the three probes (peakedness -0.07%, budget-aware -0.04%, negentropy 0.00%), inside run-to-run noise at n=30. Negentropy ties because its normalization saturates when score rows have similar entropy, leaving p_v alone, as the linear model predicts. The tested corrections give no measurable improvement at the operating points of Table 10, so the undiscounted path prior is the weight Eq. 6 implies; the depth discount $1/(1+d_v)$ the implementation adds changes nothing measurable. Root-only also bounds what aggregating all N nodes adds *in aggregate*: 0.7% of Φ over root-query scoring at every layer, so the large gap to root reuse (Figure 3a) comes from re-scoring every layer (the depth split is in Appendix B.2).

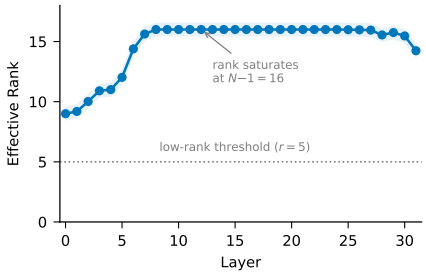
Cross-context and cross-selector linearity. At 32K and 64K the gap to Root-only shrinks as TP saturates above 0.97 and the linear form stays within noise of the best correction at every cell; a selector $\times$ likelihood cross-product at 16K (MeanPool, Quest, TokenSelect, TidalDecode) gives the same answer, peakedness lifting $\Phi^\star$ by $\leq 0.1\%$ at every selector.

B.2 THE NODE AXIS

Two diagnostics motivate aggregating the tree nodes' queries, although the loss itself comes from the layer axis (§5.2). The nodes' top-K block sets diverge most in the middle layers (Figure 4a), and they share no low-dimensional structure that a few basis queries could stand in for (Figure 4b), so scoring cannot be reduced to a handful of directions. The middle layers are also where the retained-mass table of Figure 2a drops. The shared top-K set is nevertheless insensitive to which node scores it (Appendix B.1): the queries differ, but they largely agree on the blocks that matter, so re-scoring the layer matters more than re-scoring the node.



(a) Per-layer Jaccard overlap of the tree nodes' top-K block sets (N=17 queries, 7–15K context): Quest and TidalDecode dip to 0.56–0.65 at layers 8–15. The dashed average is the one reported by Wang et al. (2025).



(b) Effective rank of the deviation matrix of the N=17 tree queries (maximum $N-1$=16) per layer. From layer 8 the rank is near its maximum, so the queries are not collinear.

Figure 4: **Node-axis diagnostics** (LLaMA-3.1-8B + EAGLE3).

Where node aggregation acts. Table 2 (main text) makes node aggregation look nearly free of effect: Root-only trails Eq. 1 by 0.1–0.3 TP points in aggregate. The aggregate hides where the difference is. Splitting the same n=30 events by tree depth (Table 7) shows that Eq. 1 gains nothing at the root — Root-only *is* the root's own selection, so the two agree there by construction — and gains 0.8 TP points at depth 2 and 1.2–1.4 at depth 3 at 16K and 32K ($p < 0.01$, paired), and 0.6 at depth 2 at 64K; there only 2% of nodes reach depth 3 and the gain is not significant. Shallow nodes dominate the node count, so the aggregate is small; deep nodes decide whether an accepted path extends, so the aggregate understates what aggregation buys. The two levers are independent: Eq. 1's gain over Root-only is the same size whether layers are re-scored or reused ($+0.2$–0.7 vs. $+0.2$–0.6 TP points at s=0.85–0.95, 16K–32K, n=30), with the same depth gradient under both schedules, so node aggregation and per-layer re-scoring add rather than interact.

Table 7: **Where node aggregation acts** (LLaMA-3.1-8B + EAGLE3, dual-run TP protocol). Gain of Eq. 1 over the root query alone (both re-scored at every layer) in dense–sparse verify-argmax agreement at depth-d tree nodes, TP points, at the Table 2 operating points (arXiv, n=30 documents, paired t-test: † $p < 0.01$). The last row is the fraction of rounds whose dense-accepted path reaches depth d (16K/32K/64K), i.e. how often that depth decides accepted length. ^{a}Only 2% of 64K nodes are at depth 3.

	depth 0	depth 1	depth 2	depth 3
16K, s=0.85	0.0	+0.2	+0.8†	+1.4†
32K, s=0.95	−0.2	+0.7†	+0.8†	+1.2†
64K, s=0.97	0.0	+0.3†	+0.6†	−1.0^a
rounds reaching d	.83/.75/.51	.64/.50/.12	.38/.21/.01	.19/.07/.00

B.3 Selector Sweep

Figure 5 sweeps the selector under a fixed every-layer schedule at 16K and 32K and adds a Φ-penalty heatmap: for each selector, the drop in Φ as a percentage of its own maximum when the sparsity is mis-chosen. The four selectors are within a few percent of one another in Φ at every sparsity, and the penalty for mis-choosing s has the same shape for all of them, so the selector is not where the operating point is decided.

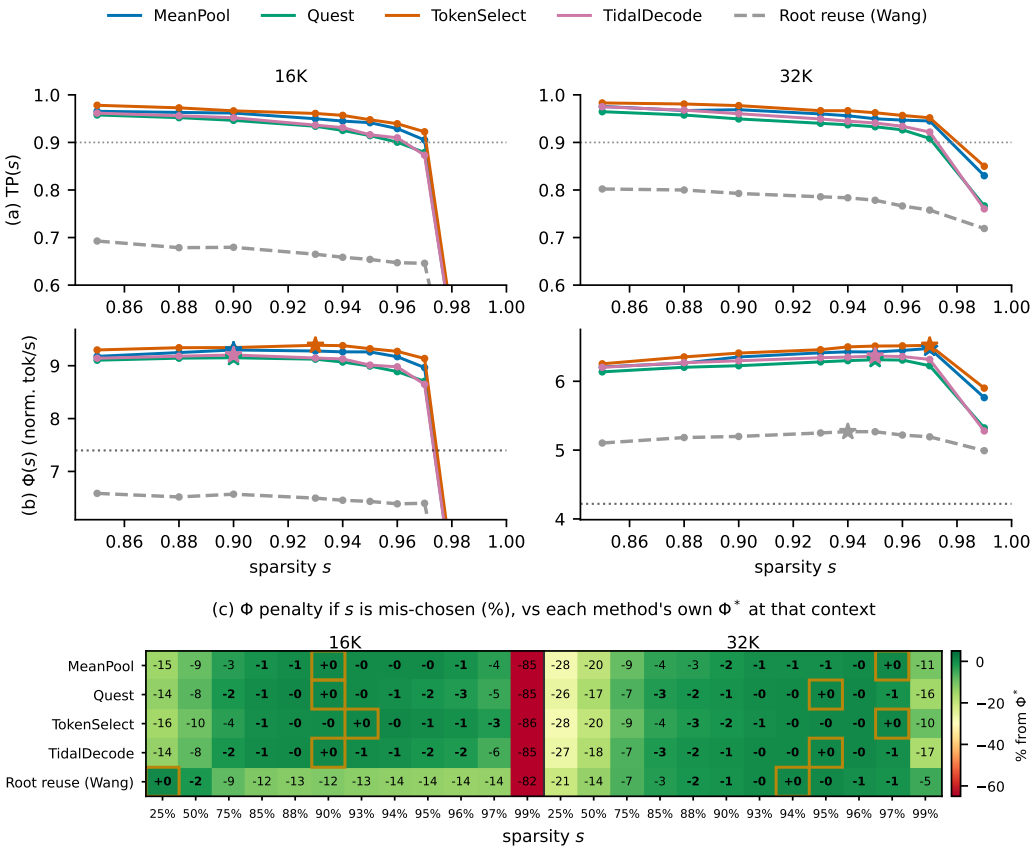


Figure 5: **Selector comparison** (LLaMA-3.1-8B-Instruct + EAGLE3, dual-run TP protocol on arXiv, n=30, every-layer schedule; Φ with the Table 4 costs). Columns: 16K, 32K. (a) TP(s) and (b) Φ(s), cropped to $s \geq 0.85$ where the curves separate; (c) Φ penalty over the full dial, selectors down the rows, context blocks across. Gold: each selector's s^* in its block; ⋆ in (b): the same point when inside the window (root reuse at 16K peaks at s^*=0.25, off the left edge).

B.4 The Operating Point: Φ-Chosen Points and the Drafter Ceiling

Under the Arctic LSTM setup's round times ($\beta_0 \approx 40$ ms, $c = 0.50/0.82/1.59$ ms at 16K/32K/64K) Φ selects s=0.95 at 16K, 0.97 at 32K and 0.99 at 64K. Past the rule of Table 3, Table 8 collects the sparser 4K and 16K points (s=0.85, 0.95), at which the controller's loss is significant, and the Φ-chosen 32K and 64K points; arXiv was not measured at these points.

Table 8: **The aggressive operating points** (LLaMA-3.1-8B + Arctic LSTM, GovReport): the points past the rule of Table 3, the sparser 4K and 16K points (s=0.85, 0.95) and the Φ-chosen points at 32K and 64K (0.97, 0.99), same protocol, AR measured in the same run at each context. At 32K and 64K the online controller loses 6.4 and 6.8% ROUGE-L against 4.7 and 4.8% at the headline point; neither schedule row is significant at n=50 (p=0.10–0.14), while AR + sparse cache and root reuse lose significantly. Per-round TP is still 0.956 at 64K, so per-round Φ would not register such a loss (§5.3). The online controller reaches 6.24× AR at 64K here against 5.51× at the headline point.

	4K, s=0.85		16K, s=0.95		32K, s=0.97		64K, s=0.99	
Method	×AR	ΔRL	×AR	ΔRL	×AR	ΔRL	×AR	ΔRL
AR (tok/s)	29.9		22.9		14.1		8.0	
Dense SD	1.90	+0.0%	1.30	+0.4%	1.16	+0.2%	1.15	+0.2%
AR + sparse cache	0.69	−4.4%*	0.93	−6.7%†	1.23	−8.3%*	1.77	−8.5%*
Root reuse (Wang)	2.04	−9.0%†	2.78	−13.1%†	3.74	−13.4%†	7.80	−12.0%†
Every layer (32)	1.52	−6.7%†	2.05	−6.4%†	2.67	−5.7%	3.88	−6.4%
Online anchors	1.87	−5.6%*	2.53	−5.4%*	3.43	−6.4%	6.24	−6.8%

The drafter ceiling. Sparse verification saves KV-read time only, so what it can win depends on how much of a round is the verify forward. With EAGLE3 at 64K that share is small twice over: 160 of each 433 ms round is drafting (Table 4), and the drafter's acceptance falls to 1.75 tokens per round, so each round commits little. Figure 6a shows the collapse, from 3.64 at 16K, and the Arctic LSTM tree holding 2.77–2.87 from 4K to 64K instead.

That difference sets how visible the schedule's effect is on each stack (Figure 6b). On EAGLE3 at 64K, moving from every-layer selection to the online controller saves about 23 layers of scoring per round, some 41 ms of a 433 ms round, and the measured speedup moves 1.25 → 1.38× AR: a 10% gain, which is what a drafter-bound round allows. On the LSTM the same change moves

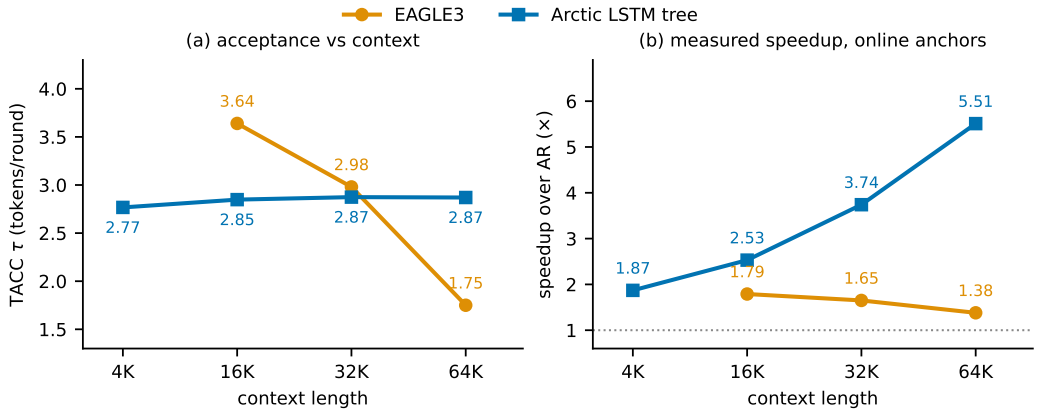


Figure 6: **Drafter ceiling** (LLaMA-3.1-8B with EAGLE3 and with the Arctic LSTM tree drafter; GovReport, n=50, RTX 6000 Ada). (a) Bonus-inclusive acceptance against context under the online controller: EAGLE3 falls 3.64 → 1.75 from 16K to 64K while the Arctic LSTM tree holds 2.77–2.87 from 4K to 64K. (b) Measured speedup over AR under the online controller; the LSTM cells are from Table 3 at 32K and 64K and from Table 8 at 4K and 16K, the EAGLE3 cells from Table 10. The schedule's gain is larger on the drafter whose acceptance does not decay.

$3.44 \rightarrow 5.51\times$, a 60% gain. The saving per round is similar on both drafters; the LSTM round is shorter, so the same saving is a larger fraction of it. This is the other half of Φ's prediction (§4.4): when α is small relative to β_0, sparse verification is not expected to pay, and on EAGLE3 at 64K it barely does.

C The Layer Schedule

C.1 Count and Placement

All cells: dual-run TP protocol, arXiv, n=30 documents, EAGLE3 drafter, paired against per-layer selection on the same events.

Anchor count and placement. Table 9 sweeps the anchor count of the DP schedule on the calibration table and compares placed anchors against uniformly spaced ones, the online controller, and a published placement. Six placed anchors match per-layer TP at every context, five lose 1 TP point at 16K/32K and 0.4 at 64K; uniform placement at the same count loses 5–8 TP points, and the every-fourth-layer schedule (8 layers) loses 1.7–3. The online controller, choosing six to eight anchors, is within 0.8 TP points of per-layer at every context. TidalDecode's published re-selection layers for this model, $\{0, 2, 13\}$, found by a sensitivity sweep for single-query decoding (Yang et al., 2025a), transplanted as anchors lose $3.2/2.7/0.8$ TP points at 16K/32K/64K: within a TP point of the DP's own three-anchor schedule and well above uniform placement at twice the count. A denser hand schedule (every fourth layer up to layer 8, then every second; 14 anchors) is dominated by the online controller on both axes ($4.86\times$ at -5.8% against $5.51\times$ at -4.8% at 64K, the protocol of Table 3).

Table 9: **Anchor count and placement** (LLaMA-3.1-8B + EAGLE3, dual-run TP protocol, arXiv, n=30). TP at s=0.95 by schedule; Δ = paired TP points vs. Every layer (32); paired t-test, * $p < 0.05$, † $p < 0.01$. The online and TidalDecode rows are a later run of the same protocol on the same documents and event stream, not paired in-run with the per-layer row, so no significance marks are given for them. TidalDecode keeps layers 0–1 dense in the original method; here they are sparse on layer 0's selection.

Schedule (anchors)	16K	32K	64K
Every layer (32)	0.938	0.954	0.978
Placed anchors (8)	0.937 (-0.1)	0.954 (0.0)	0.977 (-0.1)
Placed anchors (6)	0.937 (-0.1)	0.952 (-0.3)	0.977 (-0.1)
Placed anchors (5)	0.927 (-1.1^*)	0.942 ($-1.2^\dagger$)	0.974 (-0.4)
Placed anchors (4)	0.918 ($-2.0^\dagger$)	0.943 ($-1.2^\dagger$)	0.975 (-0.3)
Placed anchors (3)	0.914 ($-2.4^\dagger$)	0.936 ($-1.8^\dagger$)	0.970 (-0.8^*)
Uniform anchors (6)	0.862 ($-7.5^\dagger$)	0.895 ($-5.9^\dagger$)	0.930 ($-4.8^\dagger$)
Every 4th layer (8)	0.908 ($-3.0^\dagger$)	0.927 ($-2.7^\dagger$)	0.961 ($-1.7^\dagger$)
Online anchors (6–8)	0.930 (-0.8)	0.953 (-0.1)	0.978 (0.0)
TidalDecode layers $\{0, 2, 13\}$ (3)	0.906 (-3.2)	0.927 (-2.7)	0.970 (-0.8)
TidalDecode layers $\{0, 13\}$ (2)	0.903 (-3.5)	0.926 (-2.8)	0.967 (-1.1)

The count rule. Eq. 4 is evaluated by the controller on its own table. We measure it in a run separate from the online row of Table 9 (dual-run TP protocol, n=30, s=0.95), paired in-run against per-layer selection and against the controller held at a fixed count. On LLaMA it chooses six anchors at 16K (-0.5 TP points; fixed six -0.3, fixed eight -0.4), seven at 32K (-0.7; fixed six -0.3, eight -0.6) and eight at 64K (-0.2; fixed six $+0.0$, eight -0.2), none of the differences significant. On Qwen3-8B it settles on ten, eight and nine anchors at 16K, 32K and 64K (10.5–12.5 scored layers per round) and trails per-layer selection by 0.6–1.2 TP points paired in its own run; in a separate 32K run paired the same way it is at -0.7 TP points (not significant) where a fixed six is -1.3 ($p < 0.01$) and a fixed eight -0.9: the rule raises the count on a model whose staleness is spread more evenly across depth. End to end at 64K the self-chosen controller (seven anchors on most documents) reaches $5.51\times$ AR at -4.8% ROUGE-L, within noise of the controller held at eight ($5.46\times$, -3.7%; paired $\Delta = -0.002 \pm 0.004$, two standard errors).

Exact solve versus greedy search. A greedy search on the same table (drop the anchor whose removal costs least, or add the one that gains most) reaches the same set as the dynamic program at 64K and differs by one layer at 16K/32K. Where they differ the exact set is better: at 16K, s=0.95, six exact anchors are -0.4 TP points vs. per-layer and the greedy sets -0.9/-1.1; eight exact $+0.1$ vs. greedy -0.5; at 32K, -0.5 vs. -0.7/ $-$ 0.4 and -0.3 vs. -0.4. The exact solve costs 0.3 ms and is never worse, so we use it throughout.

Reacting within a round. A rule that measures staleness at each reused layer on the actual round and re-selects where it exceeds a threshold needs no lag and no table, but is worse at equal cost: at about eight re-selections per round it loses 1.3 (16K) and 1.0 (32K) TP points where eight fixed anchors lose 0.05; it needs 12–13 re-selections per round to match eight placed anchors, and adding it on top of the offline anchors gains nothing. Re-selecting where staleness has already appeared is one layer too late; the schedule places anchors for the layers above them.

C.2 The Online Controller

Online convergence and persistence. Restarting the controller on every document, its six-anchor schedule is 0.9 TP points under the calibration-table schedule at 16K and 0.0 at 32K. Keeping the table across documents (running average with η=0.9, a full-scoring round every 32 rounds) closes the 16K gap after about ten documents: over the first ten it trails that schedule by 1.5 TP points, over the next twenty by 0.2, and settles on the same anchors, $\{0, 5, 8, 14, 20, 31\}$. Extra full-scoring rounds early in each document (a warm-up) add scored layers without changing ROUGE-L at 64K within noise. Replayed on recorded tables, a rule that re-scores the stalest layers each round oscillates (mean staleness $1 - \Gamma$ over layers of 0.09 against 0.048 for the fixed schedule), because staleness is measured relative to the current anchors; the moving-average table with an exact re-solve is stable.

A late-layer weight in the schedule sum. Weighting late layers helps Qwen3-8B ($+0.9$ TP at nine anchors) but costs LLaMA-3.1-8B 1.1 TP points, so Eq. 3 has no such term. The rest of the Qwen3 evidence is in Appendix D.2.

D Generalization: the Other Drafter, Model and Tasks

The headline configuration is LLaMA-3.1-8B with the Arctic LSTM tree drafter (Table 3). This appendix fills the rest of the drafter-by-model grid: LLaMA-3.1-8B with EAGLE3 (Table 10, Figure 7), Qwen3-8B with EAGLE3 (Table 11, Figure 8), and the headline configuration on other tasks (Table 12). We have no LSTM tree drafter for Qwen3-8B, so that cell of the grid is empty.

D.1 LLaMA-3.1-8B with EAGLE3

EAGLE3 is the drafter behind the calibration table and every TP measurement in the paper; on the base model and datasets of Table 3, at its own operating points, Table 10 gives its end-to-end results. The online controller beats dense SD at every context; on GovReport it is 7–10 ROUGE-L percentage points above root reuse at a 1–6% speed cost. The ranking on GovReport is consistent on arXiv: the online controller reaches $1.74/1.63/1.42\times$ over AR, within $0.1\times$ of GovReport, at ROUGE-L within noise of AR at every context and 10–15 ROUGE-L percentage points above root reuse. TACC is lower on scientific prose, but the quality ordering dense SD $\geq$ SKIVE $\gg$ root reuse holds; the 64K narrowing reflects the EAGLE3 drafter ceiling rather than the schedule.

Schedule sweep. The 32K sweep of Figure 3 is extended to all three contexts in Figure 7, which adds the TP grid of every schedule, context and sparsity. Every layer and the online controller are indistinguishable in TP at every context until s passes the operating point, where they fall together; uniform placement is 3–10 TP points below them (5–8 at s=0.95) and root reuse 15–30. The penalty for mis-choosing s is asymmetric: under-shooting to s=0.85 costs at most 3% of Φ at 32K and 7% at 64K, whereas over-shooting to 0.99 costs 85% at 16K and 12–13% at 32K, where TP drops sharply; at 64K the drop has not begun and 0.99 is within 1% of the optimum.

64K. Every schedule's s^* lies at 0.97–0.99 at 64K, where the saveable share reaches 45% (Table 4). There the online controller reaches $1.18\times$ the per-layer Φ at s=0.95, uniform placement $1.13\times$ and

Table 10: **LLaMA-3.1-8B-Instruct + EAGLE3 on GovReport and arXiv.** End to end (n=50, 200 tokens, greedy, one RTX 6000 Ada). Keep $= 1-s$ is $15/5/3\%$ at 16K/32K/64K, the arXiv-calibrated Φ operating points. AR and AR + sparse cache are separate measurements on the same GPU; the sparse rows of a context share one run, and the online and every-layer rows are separate runs on the same documents (paired t-test: * $p < 0.05$, † $p < 0.01$; unmarked cells are not significant). Uniform anchors (six, evenly spaced) and every-layer selection bracket the controller: the same count with worse placement, and no schedule at all. Dense SD is lossless in exact arithmetic; its residual is FP16 reduction order at $q_{\text{len}} \approx N$ against AR's $q_{\text{len}}=1$. TACC of the SD rows is 3.0–3.8 at 16K, 2.4–3.1 at 32K and 1.6–1.8 at 64K. Ratios are against the AR of the same implementation (Appendix A.1).

	16K		32K		64K	
Method	×AR	ΔRL	×AR	ΔRL	×AR	ΔRL
GovReport						
AR (tok/s)		17.4		9.7		4.8
Dense SD	1.39	+0.8%	1.11	+0.4%	0.79	−0.7%
AR + sparse cache	0.95	−1.3%	1.21	−4.2%	1.29	−2.8%
Root reuse (Wang)	1.84	−10.1%†	1.67	−13.5%†	1.46	−12.6%†
Uniform anchors (6)	1.84	−2.9%	1.67	−8.7%†	1.46	−9.5%†
Every layer (32)	1.70	+0.2%	1.53	−4.3%†	1.25	−5.3%*
Online anchors	1.79	−0.8%	1.65	−3.5%	1.38	−5.8%†
arXiv						
AR (tok/s)		16.9		8.9		4.5
Dense SD	1.25	+0.3%	0.99	−1.2%	0.73	−0.4%
AR + sparse cache	0.95	+0.5%	1.27	+0.8%	1.36	−1.5%
Root reuse (Wang)	1.77	−12.8%†	1.63	−11.5%†	1.42	−13.2%†
Uniform anchors (6)	1.81	+1.1%	1.74	−4.7%	1.51	−6.7%†
Every layer (32)	1.64	+2.1%	1.55	+1.9%	1.31	−0.4%
Online anchors	1.74	+2.2%	1.63	−1.5%	1.42	−0.6%

root reuse 1.05×; root reuse holds TP ≈ 0.82 at its own s^* where the controller holds 0.97, so the 13-percentage-point gap in relative Φ survives at the context where root reuse is least bad.

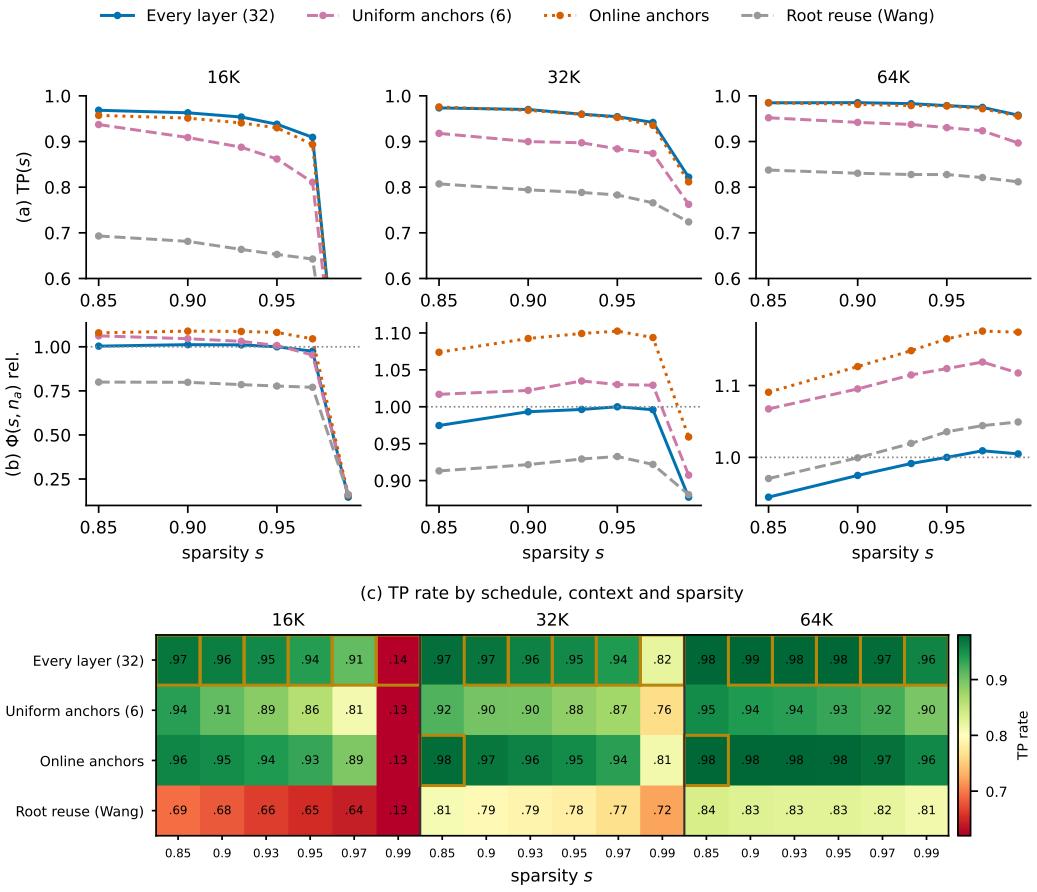


Figure 7: **Schedule sweep, all contexts** (LLaMA-3.1-8B-Instruct + EAGLE3, dual-run TP protocol on arXiv, n=30, MeanPool; Φ with the EAGLE3 round times of §4.4, as in Figure 3; six-point sparsity grid). Arms: *Every layer (32)* re-selects at all 32 layers; *Uniform anchors (6)* are six evenly spaced layers, the same count with no table; *Online anchors* is the controller choosing count and placement from its own table; *Root reuse (Wang)* scores layer 0 once. Columns: 16K, 32K, 64K. (a) TP(s) by schedule. (b) $\Phi(s, n_a)$ relative to every-layer at s=0.95. (c) TP for every schedule, context and sparsity on one scale, gold marking the best schedule in each column; root reuse is priced at one scored layer.

D.2 QWEN3-8B WITH EAGLE3

Qwen3-8B (`Qwen/Qwen3-8B` with the public EAGLE3 drafter `Tengyunw/qwen3_8b_eagle3`) is our second model family and also a diagnostic: its EAGLE3 drafter keeps acceptance high at 64K (dense-SD TACC 3.25) but is expensive on this stack, so it isolates the draft-cost-bound regime that the β_0 term of Eq. 5 predicts, in which verification-side savings are capped by T_{draft}. Same EAGLE3/HF/SDPA stack, same RTX 6000 Ada, same arXiv calibration protocol; the attention path in `modeling_qwen3_kv.py` was extended to mirror LLaMA's prefill-SDPA / verify-eager split (so the SKIVE hook applies unchanged) and the schedule was re-derived against Qwen3's 36 decoder layers (vs. 32 for LLaMA).

Diagnostics. Qwen3 follows the same roofline as Table 4 ($R^2 \geq 0.87$; the 16K fit is the loosest, as α is smallest there relative to β_0), with verify-only saveable fractions $42\%/66\%/73.5\%$ at 16K/32K/64K. On the same 13-point arXiv calibration sweep, per-layer peaks at $s^* = 0.93/0.97/0.99$ (Figure 8), so the $s = 0.85/0.95/0.97$ operating points shared by Tables 10 and 11 are below the Qwen optimum at 16K and 32K and on its plateau at 64K. The four selectors peak in nearly the

same band, with Φ spread under 5% across MeanPool, Quest, TokenSelect, and TidalDecode; as on LLaMA, the selector does not decide the operating point (Appendix B.3).

Table 11: **Qwen3-8B (36 layers) + EAGLE3 on GovReport and arXiv**, same protocol as Table 10: n=50, 200 tokens, greedy, keep rates $15/5/3\%$, one RTX 6000 Ada; the sparse rows of a context share one run, AR and AR + sparse cache are separate measurements on the same GPU, and the online row (the controller choosing its own count, ten, eight and nine anchors at 16K, 32K and 64K on this model) and the every-layer row (all 36 layers re-select) are separate runs against the same AR. Paired t-test: * $p < 0.05$, † $p < 0.01$. Verifier fidelity is reported as TP in Figure 8. Unlike LLaMA, placement does not consistently pay on Qwen3 on GovReport, whose staleness is spread more evenly across depth (Figure 8); on arXiv root reuse costs 20–23 ROUGE-L percentage points, three times its GovReport cost, while uniform anchors, every-layer selection and the controller stay within 2–7 points.

	16K		32K		64K	
Method	×AR	ΔRL	×AR	ΔRL	×AR	ΔRL
GovReport						
AR (tok/s)		15.4		8.4		4.2
Dense SD	1.36	-0.1%	1.24	-0.1%	1.05	$+0.2\%$
AR + sparse cache	0.87	-5.2%	1.24	-4.4%	1.31	$+0.8\%$
Root reuse (Wang)	1.88	$-8.3\%^{\dagger}$	1.90	$-9.9\%^{\dagger}$	1.74	-6.1%
Uniform anchors (6)	1.88	-3.9%	1.90	$-5.1\%^{*}$	1.71	-0.5%
Every layer (36)	1.75	$-4.8\%^{\dagger}$	1.80	-4.3%	1.60	$+1.5\%$
Online anchors	1.85	$-5.3\%^{\dagger}$	1.90	$-6.0\%^{*}$	1.69	-0.8%
arXiv						
AR (tok/s)		15.6		8.6		4.2
Dense SD	1.21	-0.3%	1.08	$+0.2\%$	0.98	-0.5%
AR + sparse cache	0.92	-1.0%	1.21	-0.3%	1.31	-6.2%
Root reuse (Wang)	1.75	$-23.2\%^{\dagger}$	1.79	$-19.8\%^{\dagger}$	1.67	$-20.8\%^{\dagger}$
Uniform anchors (6)	1.67	-3.3%	1.73	$-5.7\%^{\dagger}$	1.62	$-7.2\%^{\dagger}$
Every layer (36)	1.58	-2.1%	1.60	$-3.3\%^{*}$	1.50	$-4.6\%^{*}$
Online anchors	1.68	-2.7%	1.74	-3.5%	1.60	$-7.0\%^{\dagger}$

Figure 8 is the schedule sweep on this model, at the anchors the online controller settles on for it.

What replicates. The same pattern appears on Qwen3: SKIVE and root reuse are equally fast, but SKIVE keeps far more of the dense verifier's accepts. At 16K the online controller reaches $1.85\times$ over AR vs. LLaMA's $1.79\times$ (Table 10); at 32K it reaches $1.90\times$ vs. LLaMA's $1.65\times$. SKIVE's TP advantage over root reuse holds at every context: 22/21/14 TP points at s=0.95 for per-layer selection and 21/20/14 for the online controller on the arXiv calibration set of Figure 8. The downstream quality gap over root reuse is narrower on GovReport (3.0/3.9 ROUGE-L percentage points at 16K/32K against LLaMA's 9.3/10.0) and wider on arXiv, where root reuse collapses hardest (20.5/16.3 ROUGE-L percentage points against LLaMA's 15.0/10.0).

Qwen3 SD operating point. Qwen3 shifts the optimum sparser because its per-layer TP curve decays gently, still holding TP ≥ 0.88 at s=0.95 at every context (0.92 at 32K), and because its EAGLE3 drafter does not collapse at 64K: dense-SD TACC is 3.25 on GovReport (2.73 on arXiv) versus LLaMA's 1.83. We report TP in Figure 8: TACC characterizes the drafter's bonus-inclusive committed-token count, while TP measures how many dense non-bonus accepts survive sparse verification (§2.1).

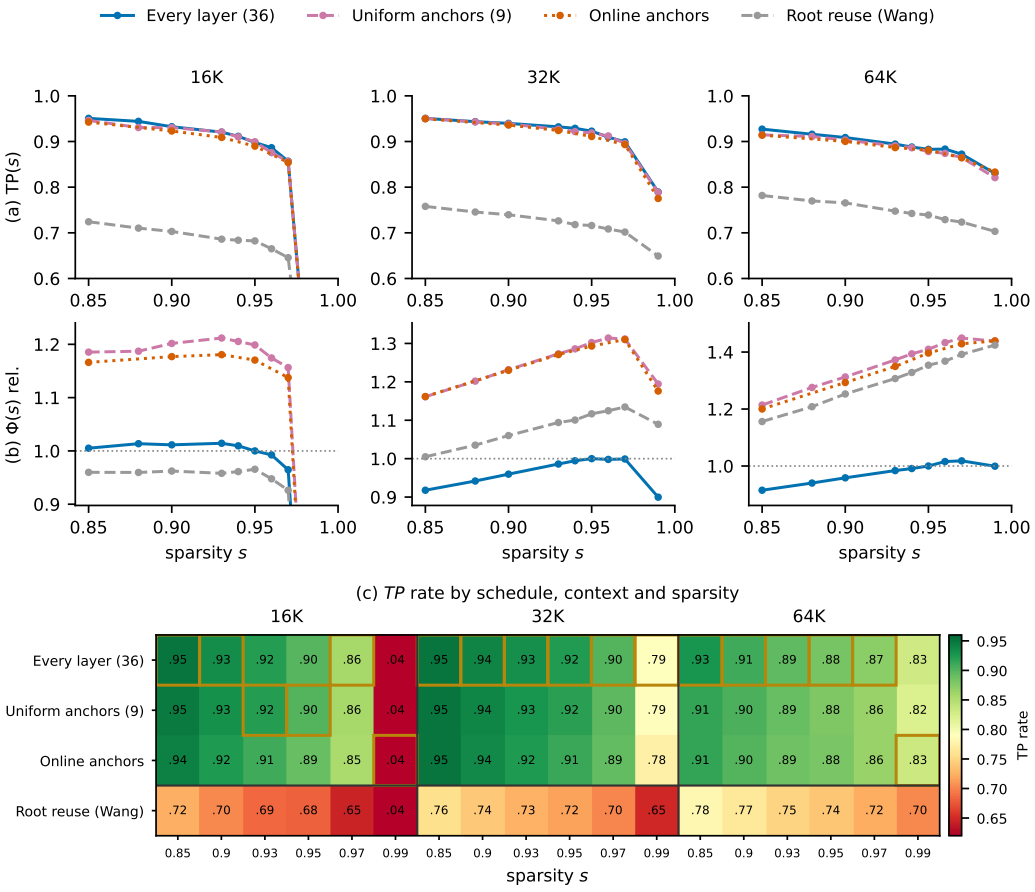


Figure 8: **Schedule sweep, all contexts, on Qwen3-8B** (Qwen3-8B + EAGLE3, dual-run TP protocol on arXiv, n=30, MeanPool, no calibration; 13-point sparsity grid, the online arm on six of those points in its own run of the same protocol). Arms as in Figure 7, at nine anchors, close to the ten, eight and nine the online controller chooses at 16K/32K/64K on this model. Columns: 16K, 32K, 64K. (a) $\mathrm{TP}(s)$ by schedule, cropped to $s \geq 0.85$; the s=0.99 collapse runs off the axis and is shown in (c). (b) $\Phi(s)$ relative to every-layer at s=0.95, priced with Qwen3-8B's round-time parameters and the per-layer scoring cost of the LLaMA fit. (c) TP for every schedule, context and sparsity on one scale, gold marking the best schedule in each column. Root reuse collapses on Qwen3 as it does on LLaMA, by 14–22 TP points at s=0.95. Uniform placement does not: at s=0.95 nine uniform anchors are within 0.5 TP points of every-layer at every context (within 1.4 over the grid), and the online controller (ten, eight and nine anchors) trails the every-layer curve drawn here by 0.2–1.2 (0.6–1.2 paired within its own run); Qwen3's staleness is spread across depth, so evenly spaced anchors already cover it.

D.3 Other Tasks: Retrieval, Multi-hop QA and Code

Tables 3 and 10 measure summarization. This subsection asks whether SKIVE's quality advantage over root reuse holds on other tasks by re-running the same comparison on five additional probes.

Setup. We evaluate four LongBench tasks (Bai et al., 2024) — HotpotQA, 2WikiMQA, and MuSiQue (multi-hop QA, scored with token-F1) and LCC (code completion, scored as edit similarity on the first generated line) — plus needle-in-a-haystack retrieval (NIAH; a fixed synthetic needle injected into arXiv prose at depths $\{10\%, 50\%, 90\%\}$, scored as binary key-phrase recovery). The stack matches Table 10 (EAGLE3/HF/SDPA); these are quality-only runs, so cells were split across A6000 and RTX 6000 Ada GPUs. SKIVE (LSTM) uses s=0.95 at every context, the Table 3 point at 32K and 64K; SKIVE (EAGLE3) uses the Table 10 operating points at 16K–64K (s=0.85/0.95/0.97)

and s=0.90 at 8K, where no calibration sweep exists, so the LSTM variant runs sparser at 8K–16K and its losses are correspondingly larger. We report absolute score changes with significance rather than relative percentages, because several AR baselines are near 0.1 and relative changes exaggerate small absolute differences. We exclude samples below $0.8\times$ the target context and cap generation at 32 tokens for QA and 64 for LCC/NIAH. These short-answer probes are prefill-dominated, so we report quality only — Table 10 and §5.3 remain the end-to-end speed measurements.

Table 12: **Cross-task quality** on LLaMA-3.1-8B-Instruct. AR is the absolute score; the three sparse columns are absolute changes vs. AR (token-F1 for QA, edit similarity for LCC). Paired t-test vs. AR: * $p < 0.05$, † $p < 0.01$; unmarked cells are within noise. The 2WikiMQA 16K row has n=19, the whole population above the length filter. NIAH cells are pass rates; AR passes 100% at every context. The SKIVE (LSTM) column uses the online controller of §4.3; the SKIVE (EAGLE3) column uses a fixed six-anchor schedule. SKIVE (EAGLE3) is at the same keep rate as root reuse; SKIVE (LSTM) is at s=0.95 throughout.

Task	Ctx	n	AR	Root reuse Δ	**SKIVE Δ**	
					EAGLE3	**LSTM**
LongBench: token-F1 (QA) and edit similarity (LCC), absolute change						
HotpotQA	8K	100	0.125	-0.025^*	$\mathbf{+0.002}$	$\mathbf{-0.011}$
	16K	100	0.186	$-0.037^\dagger$	$\mathbf{-0.004}$	$\mathbf{-0.022}$
2WikiMQA	8K	98	0.137	$-0.036^\dagger$	$\mathbf{-0.010}$	$\mathbf{-0.015}$
	16K	19	0.114	-0.051^*	$\mathbf{+0.008}$	$\mathbf{+0.007}$
MuSiQue	8K	100	0.096	-0.043^*	$\mathbf{-0.026}^*$	$\mathbf{-0.027}^*$
	16K	100	0.124	-0.036^*	$\mathbf{-0.007}$	$\mathbf{-0.022}$
LCC	8K	36	0.524	-0.065^*	$\mathbf{+0.002}$	$\mathbf{-0.047}$
NIAH retrieval (% pass; AR = 100% at every context)						
NIAH	16K	60	—	25	**98**	**93**
	32K	60	—	0	**95**	**92**
	64K	60	—	0	**100**	**100**

Findings. On multi-hop QA the absolute cost of sparse verification is larger than on summarization. Against AR, root reuse's loss is significant on all seven LongBench cells; the EAGLE3 and LSTM variants of SKIVE are each significant on one, the same cell (MuSiQue 8K, -0.026 and -0.027). The largest absolute losses (-0.03 F1 on MuSiQue at 8K) sit below the regime Φ recommends (saveable share 26% at 16K, Table 4), and they arise despite high per-decision fidelity: on 2WikiMQA at 16K per-layer selection keeps TP=0.976 at s=0.95 (dual run, n=22 under that probe's own length filter), but a three-token answer is decided by one wrong token, so short-answer F1 amplifies rare flips that a 200-token summary absorbs. Needle retrieval gives the clearest contrast: root reuse collapses to 0% pass at 32K and 64K, while both SKIVE variants stay above 90% at every context and both match AR at 64K. SKIVE is above root reuse on every cell; its own loss on retrieval and short-answer QA is up to 0.03 F1 and 8 percentage points of needle pass rate.

E Where the Residual Loss Comes From

Section 5.3 attributes the residual quality loss of sparse verification to the K/V entries that accepted tokens write under sparse reads. This appendix gives the measurement behind that attribution (Appendix E.1) and, as a supplementary analysis, tests the repair it suggests, a periodic dense refresh of the written entries (Appendix E.2). No headline configuration uses the refresh.

E.1 Read and Write Channels

Sparse verification can change a decision through two channels. The *read* channel is the current step: its attention misses a block that mattered. The *write* channel is the cache: the K/V entries of each accepted token are computed from hidden states that passed through sparse attention, and every later step reads them. Prefill is dense, so only generated positions carry write error, and the number of such entries grows with every accepted token.

Figure 9 separates the two in teacher-forced decoding, which feeds the dense output back one token at a time so that every configuration is scored on the same positions (tree verification writes accepted entries the same way). A *confident flip* is a position where sparse and dense verification pick different tokens although the dense margin (top-1 minus top-2 logit) is at least 1; this sets aside near-ties (counting them, dense writes remove 48% of all flips). Writing the cache densely and keeping only the reads sparse gives 3 confident flips against 71 with both channels sparse at s=0.90, and 8 against 115 at s=0.95; no document has more flips this way (paired over documents, $p < 10^{-5}$). Keeping the writes sparse and making the flipped step's reads dense leaves 47 and 65. In this probe, then, most confident flips involve the sparsely written cache, and a step's sparse reads alone rarely cause one. The two partial counts do not add up to the total: many flips need both channels at once.

Per-round TP sees only the read channel: the dual-run protocol (§5.1) evaluates each sparse verification on the cache that dense verification wrote.

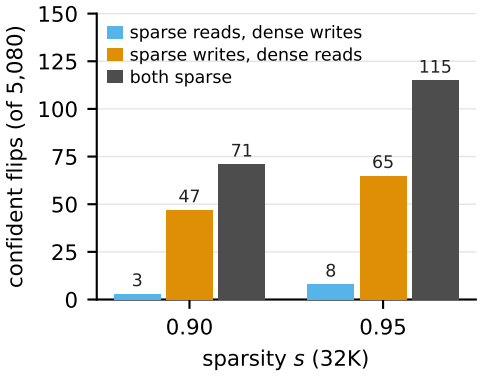


Figure 9: **Read and write channels** (LLaMA-3.1-8B, GovReport, 32K; teacher-forced single-token decoding with every layer selecting, 40 documents $\times$ 127 positions = 5,080 per bar). Confident flips (dense margin $\geq$ 1) with only the reads sparse (the cache written densely), with the cache written sparsely and the flipped step's reads dense, and with both sparse. The first two bars do not add up to the third; many flips need both channels.

E.2 Periodic Dense Refresh

If most of the loss enters through the cache, rewriting the cache should recover it. Every W committed tokens, a dense refresh re-feeds the last W tokens through the base model with dense attention and overwrites their K/V entries, while verification reads stay sparse. The 32K runs re-feed the window one token at a time and the 64K runs in one forward pass; either way the cost is roughly one dense pass over the window every W tokens. (W is unrelated to the period R at which the online controller re-scores every layer, Algorithm 1.) Periodic dense re-verification has precedent (Yang et al., 2025a; Tan et al., 2025; Lee et al., 2026b); we use it here to test the attribution above. The two settings below differ in drafter, context and output length and are not compared with each other.

What it recovers (EAGLE3, 32K). In the teacher-forced probe at s=0.95, a refresh every 32/16/8 tokens reduces confident flips from 115 to 56/38/27. End to end (Table 13; EAGLE3 with *Every layer (32)*, n=40 paired documents, 200 tokens, gap measured against dense verification), a refresh every 16 tokens leaves no significant gap at $s \leq 0.97$ and a significant one at $s \geq 0.98$. It recovers 0.002–0.008 ROUGE-L (1–4% of dense) at every s; the gap that remains, and grows with s, comes from the sparse reads and from the entries written since the last refresh.

On the online schedule it does not pay (Arctic LSTM, 64K). Table 14 adds refresh to the headline configuration (*Online anchors*, s=0.95, GovReport, n=50 paired, 128 tokens). Without refresh the online controller's loss is not significant (-4.8%, p=0.20). Periods from 8 to 64 tokens raise ΔRL by 1.0 to 2.9 percentage points, gains of the size seen at 32K but none significant here (paired within the run, $p \geq 0.11$), and cost 14–43% of the speed (5.40$\times$ down to 4.63–3.07$\times$). In the same setting, refresh every 16 tokens gives *Every layer (32)* +2.7 and +2.6 percentage points at 16K and 32K (significant at 32K, p=0.035) and none at 4K or 64K, at 9–21% of its speed. Nor does refresh close root reuse's gap: with a refresh every 16 tokens it stays significantly below AR at each of s=0.85, 0.90 and 0.95 (-8.3 to -9.6%) and is slower than the online schedule without refresh. We therefore leave refresh out of every headline configuration. The online schedule was tested with refresh at 64K only, with limited power (only 57 GovReport documents pass the 64K length filter of at least 0.8$\times$ the context; 90% equivalence bounds $\pm$7–11%) and over 128 generated tokens; longer outputs accumulate more write error.

Table 13: **Dense refresh at 32K** (LLaMA-3.1-8B + EAGLE3, Every layer (32), GovReport, n=40 paired documents, 200 tokens). Absolute ROUGE-L gap to dense verification (lossless, so AR up to FP16 noise; ROUGE-L 0.195) without refresh and with a refresh every 16 tokens, and the ROUGE-L the refresh recovers. Paired t-test: * $p < 0.05$, † $p < 0.01$; unmarked cells are within noise. Retained tokens are nominal, $(1 - s) \times 32{,}768$; shorter documents retain proportionally fewer.

s	Retained tokens	Gap, no refresh	Gap, refresh every 16	Recovered
0.90	3,277	0.0051	0.0011	0.0041
0.95	1,638	0.0078*	0.0017	0.0061*
0.97	983	0.0141†	0.0066	0.0075*
0.98	655	0.0167†	0.0147†	0.0020
0.99	328	0.0293†	0.0247†	0.0046

Table 14: **Dense refresh at 64K** (LLaMA-3.1-8B + Arctic LSTM, Table 3 protocol; GovReport, n=50 paired, 128 tokens): speedup over AR, paired ΔROUGE-L(gold) vs. AR, paired t (* $p < 0.05$), and the equivalence bound within which the cell passes a 90% equivalence test (two one-sided tests, TOST) against AR. W is the dense-refresh period in tokens (not the controller's R). The online rows are the controller of §4.3 choosing its own count, all from one run. Rows come from three runs on RTX 6000 Ada GPUs, so speed ratios across runs carry about 5% variance: the online row without refresh reads 5.40× here and 5.51× in Table 3, with identical output.

Method	s	Refresh W	×AR	ΔRL	t	TOST
Every layer (32)	0.95	none	3.46	−1.7%	−0.88	±5.0%
		16	2.74	−1.6%	−1.00	±4.2%
Online anchors	0.95	**none**	**5.40**	**−4.8%**	−1.31	±10.8%
		64	4.63	−3.0%	−0.87	±8.7%
		32	4.38	−1.9%	−0.57	±7.4%
		16	3.95	−3.3%	−0.94	±9.2%
		8	3.07	−3.8%	−1.26	±8.9%
Root reuse (Wang)	0.95	16	4.33	−9.6%*	−2.44	±16.2%
	0.90	16	3.85	−8.3%*	−2.12	±14.9%
	0.85	16	3.41	−9.0%*	−2.46	±15.1%

F Production-Stack Details

Cost model and batch crossover. This appendix expands §6. Block scoring and top-K selection run in one fused GPU kernel per layer, and a bespoke sparse tree-attention kernel replaces FlashInfer's verify attention; prefill and the drafter stay on FlashInfer. Per-block key statistics are cached and updated only for blocks that change, so 82–85% of scoring calls read no prefix keys. On an H200 the AR baseline reaches 116.5 tok/s at 64K single-stream and 313.8 tok/s aggregate at B=8. Selection costs a few milliseconds per round; that cost and the KV read it avoids both scale with the number of *sequences* scored. Weight reads do not: a verify forward streams the model's 15 GB once per round however many sequences share it, while each 64K sequence contributes 8.6 GB of KV. The avoidable share of the verify read, $B \cdot \mathrm{KV}/(B \cdot \mathrm{KV} + \mathrm{weights})$, therefore rises from 36% at B=1 to 82% at B=8, so Eq. 5 predicts sparsity is a losing trade at B=1 on this hardware and a winning one as B grows.

The crossover is where the cost model predicts it (Table 15, Figure 10): with the fixed schedule at 64K SKIVE trails dense tree by 19% at B=1 and leads by 14%, 38% and 52% at B=2, 4 and 8 (each 90% interval excludes parity), the saved read overtaking the selection cost between B=1 and B=2, as the avoidable share crosses one half. At 32K each sequence carries half the KV (4.3 GB), so the share reaches one half only near B=3.5, and the crossover moves with it: SKIVE trails dense tree by 37% and 7% at B=1 and 2, is level at B=4 (1.06 ± 0.10) and leads by 32% at B=8. A second effect the context sweep could not show: on this hardware dense tree verification gets worse as the batch grows ($0.96 \to 0.79\times$ AR from B=1 to B=8 at 64K, $1.16 \to 0.85\times$ at 32K), since speculation's extra verify compute is poor value when memory is abundant. With a sparse cache the trend reverses: SKIVE/AR rises from 0.78 at B=1 to 1.21 at B=8.

Table 15: **Batch size on SGLang** (LLaMA-3.1-8B-Instruct with SGLang's EAGLE3 drafter, GovReport, greedy decoding, s=0.9), on an H200 and on the RTX 6000 Ada used for every HF measurement; a dash marks batch sizes a 48 GB card cannot hold. SKIVE uses the six-anchor schedule of §4.3. AR is aggregate decode throughput in tok/s, Acc. len the mean number of tokens accepted per round, and the remaining groups are ratios; ×AR is against eager AR on the H200 and against AR with CUDA graphs, SGLang's default serving mode, on the RTX 6000 Ada. Intervals are 90% confidence intervals over four batches of documents; a SKIVE ratio is **bold** when its whole interval lies above 1. Quality is measured single-stream (Table 16).

		32K				64K			
	B	1	2	4	8	1	2	4	8
H200									
AR (tok/s)	eager	134.2	227.3	354.0	493.4	116.5	175.3	245.4	313.8
Acc. len	Tree	2.50	2.39	2.57	2.49	2.17	2.22	2.25	2.25
	SKIVE	2.41	2.44	2.49	2.48	2.39	2.22	2.27	2.29
×AR	Tree	1.16	1.00	1.01	0.85	0.96	0.94	0.87	0.79
	SKIVE	0.73	0.93	**1.08**	1.10	0.78	**1.07**	**1.20**	**1.21**
	90% interval	±.04	±.08	±.02	±.17	±.09	±.05	±.04	±.03
SKIVE/Tree	mean	0.63	0.93	1.06	**1.32**	0.81	**1.14**	**1.38**	**1.52**
	90% interval	±.07	±.03	±.10	±.15	±.06	±.08	±.14	±.07
RTX 6000 Ada									
AR (tok/s)	CUDA graphs	44.7	72.8	106.0	—	36.5	52.4	—	—
	eager	43.9	70.8	103.6	—	35.0	51.0	—	—
Acc. len	Tree	2.57	2.53	2.53	—	2.10	2.22	—	—
	SKIVE	2.46	2.51	2.53	—	2.37	2.23	—	—
×AR	Tree	1.73	1.60	1.54	—	1.45	1.52	—	—
	SKIVE	**1.50**	**1.79**	**2.06**	—	**1.63**	**1.73**	—	—
	90% interval	±.07	±.12	±.09	—	±.16	±.07	—	—
SKIVE/Tree	mean	0.87	**1.11**	**1.37**	—	**1.13**	**1.14**	—	—
	90% interval	±.09	±.05	±.32	—	±.10	±.10	—	—

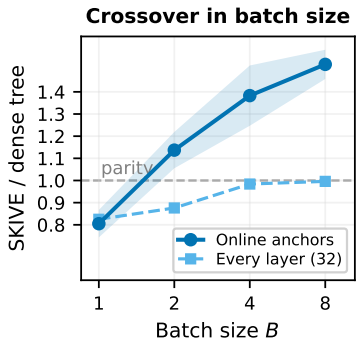


Figure 10: **The production-stack crossover is in batch size**: at fixed 64K the SKIVE/dense-tree ratio rises from 0.81 at B=1 to 1.52 at B=8 with the fixed schedule (solid; band = 90% confidence interval), crossing parity between B=1 and B=2; per-layer selection (dashed) never crosses parity (1.00 at B=8).

Caveats. Quality is not measured at B>1: the batched runs record throughput and accept lengths only, so the fidelity anchor remains the single-stream measurement below (ROUGE-L within ±7% of AR at n=20, Table 16). On the H200 all three arms run with CUDA graphs disabled, since the bespoke sparse kernels are eager-only; this costs the AR arm most, and the B=4 margin over AR is of the same order as that overhead, so the AR comparison is indicative and the dense-tree comparison, where both arms pay the same penalty, is the controlled one. Finally, SKIVE's B=1 cells ramp across their four batches as its kernels autotune and the block-statistics cache fills, while AR's throughput is flat, so their means understate the steady state (at 64K, 81 → 104 tok/s, 78 → 100 in the every-layer run, and 0.88× AR in the last batch); at 64K and $B \geq 2$ the four batches agree within 6%, and at 32K and B=8 the first batch is slow for dense tree and SKIVE alike (289 and 436 tok/s, against 454–594 in the other three).

Configuration. SGLang 0.5.10.post1 with the FlashInfer attention backend, eager mode (CUDA graphs disabled), and the stock EAGLE3 drafter (12 draft tokens per round, 4 draft steps, top-k 8). SKIVE: MeanPool selector, s=0.9 at every batch size, so that batch size is the only variable (a less aggressive keep rate than the HF headline's 5%), protected prefix and suffix kept as whole blocks, node weights from the drafter's path log-probabilities. In a microbenchmark at 32K before the block-statistics cache was added, fused scoring and top-K cost 5.3 ms per round and the sparse tree attention 2.4 ms; the cache then removed most scoring reads. We did not re-time the per-round overhead on the H200, so the "a few milliseconds" above is carried over from these pieces rather than measured in place. Within each context and GPU, the AR, dense-tree and SKIVE rows of Table 15 were measured back to back on the same card. Each cell runs B concurrent requests on four successive sets of documents; we report the mean aggregate decode throughput and, for ratios, the mean of the four per-set ratios with a 90% confidence interval. GovReport has 25 documents longer than 64K tokens, so at 64K and B=8 a few of the 32 requests repeat a document. Bit-identity of the batched path was verified on 4,864 per-layer block-selection calls at 8K, B=2 (byte-equal outputs), and at the kernel level for B=3–8 across 42 shapes; the end-to-end B=8 cell itself is not bit-identity-checked.

The same port on the RTX 6000 Ada. The RTX 6000 Ada block of Table 15 repeats the H200 protocol on the card used for every HF measurement, at the batch sizes a 48 GB card holds (32K up to B=4, 64K up to B=2). Here AR is slow relative to the port's fixed per-round cost, so speculation itself pays: dense tree reaches 1.45–1.73× AR and SKIVE 1.50–2.06×, with 2.06× at 32K and B=4, where every batch is at or above 2.00×. The reference is AR with CUDA graphs, SGLang's default serving mode; on this card graphs make AR only 2–4% faster than the eager mode the sparse kernels require. At 64K SKIVE already beats dense tree at B=1 (1.13×), where the H200 needs B=2. In the 32K B=4 cell the dense tree's first batch is slow (132 tok/s against 168–179 afterwards), which widens that cell's SKIVE/tree interval.

Single-stream quality. Table 16 is the fidelity anchor §6 refers to: the same port measured one request at a time on an RTX 6000 Ada, where ROUGE-L against the AR output can be paired per document. SKIVE's ROUGE-L is within ±7% of AR in every cell and no difference is significant, but at n=20 the test only excludes differences larger than 8–17%, so we do not claim quality parity. The single-stream throughput ratio is sensitive to the drafter and the draft-token count in a way the batch trend is not: with the Arctic-LSTM drafter at 32 draft tokens the 64K GovReport cell gives SKIVE/tree 0.89 rather than 1.04, and at 12 draft tokens it returns to parity (0.99). §6 therefore reports the batch axis, where the mechanism is a memory-traffic ratio independent of the draft-token count.

Table 16: **Single-stream SGLang port** (RTX 6000 Ada, LLaMA-3.1-8B + EAGLE3, greedy, s=0.9, n=20 documents per cell, 12 draft tokens). AR in tok/s; Tree and SKIVE as ×AR. ΔRL is SKIVE's ROUGE-L change vs. AR (paired t: $+0.8/-0.7/+0.1/-1.8/+0.9$); at n=20 the 90% equivalence bounds are ±8–17%.

Task	Ctx	AR	Tree	SKIVE	SKIVE/Tree	ΔRL
GovReport	16K	49.8	1.86	1.69	0.91	+2.3%
	32K	44.2	1.76	1.69	0.96	−5.0%
	64K	36.2	1.60	1.67	1.04	+0.6%
arXiv	32K	44.4	1.71	1.57	0.91	−6.3%
	64K	36.6	1.54	1.57	1.02	+5.3%

G Broader Impact

SKIVE accelerates long-context inference (up to 5.51× AR, Table 3) without modifying or retraining the target model. It is lossy: it composes lossy block selectors with speculative decoding and scores the loss by true positives against the dense verifier (§4.4); the deviation against AR is in Tables 3, 10 and 12, larger on short-answer QA and retrieval than on summarization. As with all inference-efficiency work, cheaper inference lowers the barrier to any deployment, which model-level safety measures address better than keeping inference expensive.