RIVET: RAGGED INFERENCE WITH VARIABLE-GRANULARITY EXECUTION FOR TOKEN-TREE VERIFICATION IN SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

As agentic applications use larger language models and longer interactions, autoregressive decoding incurs increasingly costly serial target-model passes. Tree-based speculative decoding reduces these passes by verifying multiple candidate continuations at once. This verification requires tree attention: each candidate sees the shared history and its own ancestor path, while the number of candidates and the tree topology vary across rounds. Conventional attention kernels use fixed query tiles, KV partitions, and general masks, leading to padded query work, inefficient KV scheduling, and repeated visibility checks even over the fully visible prefix. We present RIVET, a variable-granularity execution scheme for token-tree verification. It maps tree visibility to specialized regions, processing regular blocks with dense or causal attention while restricting ancestor checks to irregular blocks. It selects query tiles for the physical query shape and sizes KV partitions to balance parallelism against GPU resource and merge costs. Online-softmax merging preserves exact tree-attention semantics across regions and partitions. In nine kernel configurations, RIVET reduces attention latency by 51.6% on average relative to FlashInfer. Across 24 model–workload configurations, it improves output throughput by 7.6–50.5% and reduces median time per output token (TPOT) by 3.0–34.2% relative to the strongest baseline selected separately for each configuration and metric.

1 INTRODUCTION

The growing use of agentic applications such as Codex is driving language models toward larger scales and longer, multi-step interactions. In these workloads, autoregressive decoding dominates inference cost because each new token requires a full model forward pass. Speculative decoding uses a lightweight draft model to propose multiple candidates for joint verification by the target model, replacing serial target-model steps with a parallel verification pass (Leviathan et al., 2023; Chen et al., 2023). Existing methods improve drafting with multi-token prediction (MTP) heads (Gloeckle et al., 2024) or block diffusion (Chen et al., 2026), while EAGLE and Medusa extend candidate generation from a chain to a tree (Li et al., 2024a; Cai et al., 2024). In each round, the target model performs *tree attention*, where each query attends only to the shared history and its ancestor path. As tree attention dominates attention work during verification, its efficiency is critical to translating speculative decoding's gains into lower runtime.

Two lines of work have advanced attention efficiency. For regular shapes, FlashAttention uses IO-aware tiling and online softmax to avoid reading and writing the quadratic attention matrix to GPU memory (Dao et al., 2022; Dao, 2024). Flash-Decoding partitions the KV dimension and merges partial results exactly, exposing more parallelism for long-context decoding (Dao et al., 2023). FlashInfer brings customizable templates, heterogeneous KV-cache support, and dynamic request scheduling together in an inference attention engine (Ye et al., 2025). For tree verification, DeFT explicitly accounts for memory traffic in tree attention; its KV-guided grouping and partitioning of flattened tree KV improve prefix reuse and load balance (Yao et al., 2025).

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

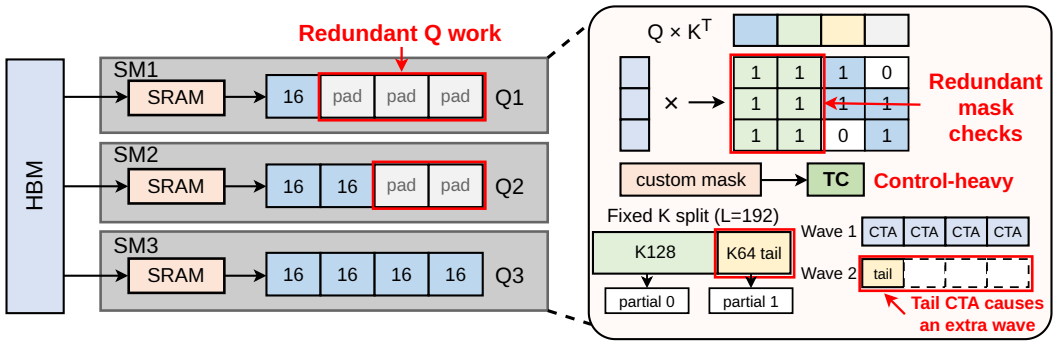


Figure 1: Execution mismatches in tree-attention verification: padded Q rows, a KV tail that can extend the CTA schedule by one wave, and redundant mask checks over the always-visible prefix.

Although existing attention kernels achieve high GPU efficiency through tiling and kernel optimizations for fixed QKV shapes, they do not map naturally to irregular tree verification. Each verification round operates on a candidate tree whose node count and topology vary with the draft, requiring a dynamically constructed attention mask. Existing implementations largely expose this structure through uniform physical query tiles (Q tiles), general masks, and fixed KV partitioning, leaving request-specific shapes and visibility structure unexploited.

As shown in Figure 1, tree verification exposes three sources of execution inefficiency. Along the Q dimension, varying the effective rows within a fixed physical Q tile has little effect on Cooperative Thread Array (CTA) latency, whereas larger query workloads require more physical rows and increase both work and latency. Along the K dimension, the best KV split changes with context length. Finer splits reduce the KV work assigned to each CTA, but also create more partial attention states, merge work, and CTAs that compete for scheduling waves. For masks, a general custom-mask path performs per-score visibility checks across the full $Q \times K$ domain, including uniformly visible prefix blocks. As KV length grows, redundant mask accesses and control checks increase, competing with general matrix–matrix multiplication (GEMM) for memory bandwidth and execution resources. These observations reveal that tree-verification efficiency depends on tightly coupled execution choices, motivating a unified execution strategy that adapts to the structure of each verification step.

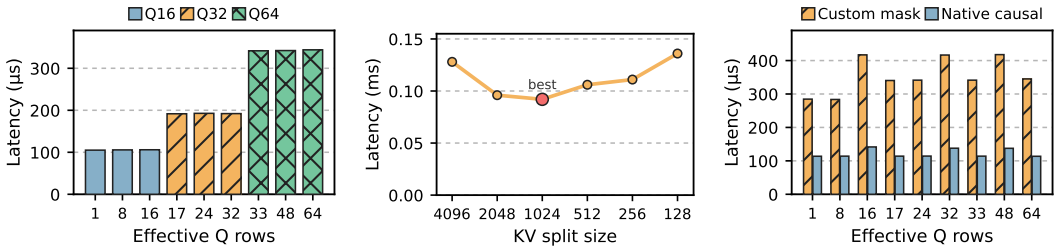
Motivated by these observations, we introduce RIVET, an adaptive execution scheme for token-tree verification. RIVET maps ancestor visibility to specialized attention regions, processing the shared history through native dense attention and restricting visibility checks to irregular blocks. It selects physical query tiles to reduce padding while accounting for repeated KV traversal, and adjusts KV partitions to balance GPU parallelism against scheduling and merge costs. These mechanisms adapt execution to changing verification workloads while preserving exact tree attention.

Our contributions are threefold. First, we identify query padding, KV-split imbalance, and redundant mask checks by analyzing how token-tree structure interacts with GPU execution (Sections 2 and 3). Second, we design RIVET with structure-aware masking, adaptive Q tiles, and resource-aware KV splitting, and implement it in FlashInfer with SGLang integration (Section 4). Third, we evaluate RIVET through kernel, end-to-end, and ablation experiments (Section 5). RIVET reduces attention latency by 51.6% on average across nine kernel configurations. Across 24 end-to-end configurations, it improves throughput by 7.6–50.5% and reduces median TPOT by 3.0–34.2% relative to the strongest baseline for each configuration and metric.

2 Background

2.1 Token Trees in Speculative Decoding

Tree-structured candidates. Tree-based speculative decoding verifies multiple candidate continuations in a single target-model pass by restricting each query to the shared history and its permitted ancestor path. SpecInfer derives a topology-aware causal mask from the candidate tree, while Medusa and EAGLE use analogous ancestor-aware masks for candidates produced by their decoding



(a) Q-template staircase. (b) KV split-size sensitivity. (c) Custom-mask overhead.

Figure 2: Warm-cache attention latency. (a) Effective Q rows vary across Q16, Q32, and Q64 templates with other execution settings fixed. (b) KV split size varies at $B = Q = 1$ and $KV = 131{,}072$. (c) Effective Q rows vary for custom-mask and native-causal paths at $B = 1$ and $KV = 131{,}072$. Panels (b) and (c) use $H_q = H_{kv} = 1$, head dimension 128, and page size 1.

heads (Miao et al., 2024; Cai et al., 2024; Li et al., 2024a). Dynamic draft-tree policies prune and expand candidates according to their estimated acceptance probabilities (Li et al., 2024b); consequently, both the node count and mask topology can vary across verification rounds.

Tree attention. Let P_i be the shared historical prefix of request i, and let $\mathrm{Anc}_i(v)$ contain candidate node v and every candidate ancestor from the tree root to v. The keys visible to query v are

$$\mathrm{Visible}_i(v) = P_i \,\dot\cup\, \mathrm{Anc}_i(v), \tag{1}$$

where $\dot\cup$ denotes a disjoint union. Tree attention retains the standard attention computation,

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\mathsf{T}}{\sqrt{d_k}}\right) V,$$

but for each candidate query v, softmax and the weighted sum range only over the keys in $\mathrm{Visible}_i(v)$. The tree mask restricts each candidate token to its visible nodes; it changes the visible key set without changing the attention formula or positional semantics. Nodes may be reordered in memory as long as their original token positions are preserved for RoPE (Su et al., 2024), ALiBi (Press et al., 2021), or any other position-dependent computation. SpecInfer expresses visibility with a topology-aware causal mask, while Medusa uses a tree mask for the candidates generated by its decoding heads (Miao et al., 2024; Cai et al., 2024). DeFT instead makes KV traffic an explicit design concern through prefix-aware grouping and flattened tree-KV partitions (Yao et al., 2025).

2.2 GPU Execution Model

A GPU kernel launches a grid of cooperative thread arrays (CTAs). All threads of one CTA execute on the same streaming multiprocessor (SM), and a CTA becomes resident only when the SM has enough thread, register, and shared-memory capacity. A kernel template therefore determines more than its arithmetic shape: its resource footprint bounds the number of concurrent CTAs and the number of scheduling waves required by the grid.

Within a CTA, tiled attention loads successive KV blocks into on-chip storage, computes their contributions for a fixed physical Q tile, and updates an online-softmax state (Milakov & Gimelshein, 2018; Dao et al., 2022). KV and region partitions preserve tree-attention semantics through online-softmax state merging (Dao et al., 2022), up to floating-point reassociation (Appendix D). Each extra partition, however, writes another partial state and increases merge work, synchronization, and workspace traffic.

3 Motivation

3.1 Redundant Q Work from Variable-Length Requests

A speculative batch usually contains requests with different numbers of draft tokens. To process every request in the batch with a single attention-kernel launch, existing implementations often choose

a common physical Q tile based on the longest draft-token sequence in the batch or a configured maximum, padding shorter requests to that shape. If one request is assigned the Q64 template, a request with 16 effective rows can still execute with a 64-row physical Q tile. However, the padded rows consume Q loads, registers, and matrix multiply–accumulate (MMA) instructions, so the CTA cost follows the physical template rather than the effective row count.

To quantify this effect, we measure attention latency as the number of valid query rows increases, which selects different query tile configurations at runtime. Figure 2a reveals a clear staircase. Within each template, latency changes by less than 1% across the row counts shown. When the effective row count increases from 16 to 17, execution moves from Q16 to Q32 and the median latency rises from 106.0 to $191.7\,\mu$s, a $1.81\times$ increase. The next transition, from 32 to 33 rows, moves from Q32 to Q64 and raises latency from 192.1 to $341.6\,\mu$s, a $1.78\times$ increase. Therefore, the cost is governed by the selected template rather than by valid rows alone. This observation motivates runtime selection among Q templates.

3.2 KV Partitioning and Scheduling Imbalance

Even when each request has been assigned its optimal Q tile, partitioning along the K dimension can introduce another source of waste. The executor distributes each KV interval across CTAs, and a fixed `split_size` creates one partial state per interval. When the KV length is not divisible by that size, the final CTA contains less useful work but still occupies a resident slot and contributes a partial result. For a length of 192, a split size of 128 produces intervals of 128 and 64 tokens. Using K64 balances the intervals, but it raises the CTA count from two to three and adds another partial write and merge input.

The batch introduces a second quantization effect. If a resource class admits S resident CTAs across the available SMs, increasing the grid from S to $S + 1$ places work in another scheduling wave. The extra CTA may contain only a short tail, yet the request cannot finish before its partial result is merged. Thus, more splits reduce per-CTA work but increase partial-result and scheduling overhead, which can raise end-to-end latency once the grid exceeds resident capacity.

Figure 2b measures this trade-off by varying the KV split size and recording the resulting attention latency. Under the configuration, latency is minimized at a split size of 1024 tokens, while both coarser and finer splits are slower. Coarser splits leave longer CTAs; finer splits add partial-result and scheduling work. The best split is thus a function of KV length and the available parallelism rather than a fixed constant.

3.3 Overhead of General Mask Execution

General custom-mask execution introduces another source of overhead in tree attention. Although tree visibility is sparse and irregular in the logical $Q \times K$ domain, a general implementation still evaluates every score position. The shared historical prefix therefore incurs the same mask-related memory I/O and control processing as the genuinely irregular candidate suffix, although every prefix key is visible to every query.

Figure 2c compares the custom-mask and native-causal paths under the same single-request configuration. Custom masking is already 2.49–$2.50\times$ slower for one to eight effective Q rows, and the gap reaches $3.04\times$ at 64 rows. Thus, the overhead is not confined to a particular Q tile; it accumulates with the per-position visibility work performed by the general mask path.

These results shift the optimization target from bitmap compactness to the scope of mask evaluation. The uniformly visible prefix should use the native dense path, while only the candidate suffix should retain ancestor-dependent processing.

4 Design

4.1 Overview

RIVET follows a single principle: compile known structure before kernel execution and bound runtime adaptation with a small set of compile-time templates and discrete choices. As shown in

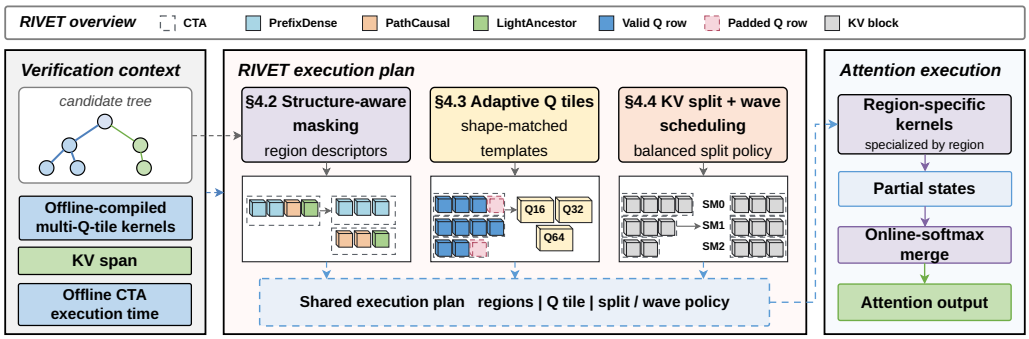


Figure 3: Overview of RIVET. The planner compiles known visibility structure into region descriptors, selects a bounded Q template, and schedules KV splits and execution waves. Region-specific kernels produce partial states that are combined by an exact online-softmax merge.

Figure 3, execution proceeds through three components. **Structure-aware masking** (Section 4.2) encodes each verification tree as region descriptors, routing regular regions through specialized paths and restricting runtime visibility checks to irregular regions. **Adaptive Q tiles** (Section 4.3) select from a small set of statically compiled templates according to the number of valid query rows, avoiding Q-direction padding. **KV-aware split and wave scheduling** (Section 4.4) chooses the KV split size by balancing per-CTA attention work, partial-result merging, and wave-level utilization. These execution choices preserve tree-attention semantics in exact arithmetic (Appendix E).

4.2 Structure-aware masks from heavy paths and block regions

Contiguous query layout from heavy paths. Following heavy-path decomposition (Sleator & Tarjan, 1983), the topology compiler chooses, for every internal node, the child with the largest subtree as its heavy child. Heavy edges form vertex-disjoint maximal paths. A DFS that visits the heavy child first assigns consecutive indices to every heavy path, which lets a Q tile align with nodes that share most of their ancestor set. Reordering changes only the execution layout: all position-dependent operations continue to use the original token positions.

Three disjoint region types. For a heavy path $H_j = [h_j, \ldots, t_j]$, the compiler covers Equation 1 with three region types:

$$\begin{aligned} &\textsc{PrefixDense}(Q_i, P_i), \\ &\textsc{PathCausal}(H_j, H_j), \\ &\textsc{LightAncestor}(H_j, U_j), \end{aligned}$$

where Q_i contains all candidate queries of request i and U_j contains the strict ancestors above the head of H_j. The first region uses native dense attention for the shared history. The second uses a causal boundary within one heavy path. The third is dense because every node on H_j sees the same strict ancestors above its path head. Candidate spans explicitly exclude P_i, and U_j excludes nodes already assigned to H_j; the compiler removes any root overlap while generating descriptors. Disjoint regions are required because an online-softmax merge cannot cancel a duplicated key.

Block classification. At block granularity, the compiler classifies each $(Q\text{-block}, K\text{-block})$ as *Dense*, *Causal*, *Zero*, or *Mixed*. Dense blocks have a common visible key set and use the native dense path; Causal blocks lie on one heavy path and use its causal boundary; Zero blocks contain no visible pair and are skipped; Mixed blocks contain both visible and invisible pairs and are subdivided or checked with ancestor intervals. Thus, only Mixed blocks require runtime visibility tests. With DFS entry and exit times, node u is an ancestor of v exactly when $\mathrm{tin}(u) \le \mathrm{tin}(v) < \mathrm{tout}(u)$, so this check uses two interval comparisons instead of a general $Q \times K$ bitmap.

Heavy-path reordering simplifies the representation of visibility; it does not change visibility itself. Adjacent nodes from different paths are not assumed to attend to each other. Every emitted block is checked against the ancestor relation before it is classified.

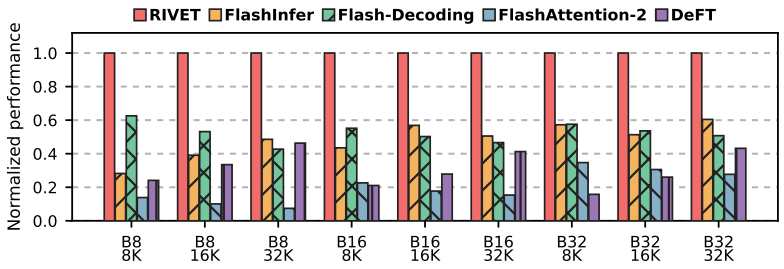


Figure 4: Inverse attention latency normalized to RIVET for each batch-size/KV-length configuration (higher is better).

4.3 Adaptive Q tiles

The effective candidate count varies continuously, but the physical query work changes at template boundaries. Generating a kernel for every runtime length would sacrifice static register allocation, shared-memory layout, and MMA mapping. RIVET compiles three Q-tile variants with 16, 32, and 64 query rows, denoted as Q16, Q32, and Q64, respectively. Their corresponding warp mappings are 1×4, 2×2, and 4×1 along the Q and K axes, respectively. At runtime, RIVET selects among these variants according to the request shape.

For grouped-query attention (Ainslie et al., 2023), template selection uses the packed query-row count $m_i = q_i(H_q/H_{kv})$, rather than the candidate-node count q_i. RIVET covers m_i with one or more Q-tile variants. A smaller tile reduces padding and resource usage, but using multiple tiles repeats the traversal of the same KV span and creates additional CTAs and partial results. RIVET therefore selects a template combination only when the saved Q work outweighs the repeated KV access and merge overhead. For $m_i > 64$, it uses multiple bounded Q tiles instead of a resource-intensive larger kernel.

4.4 KV-aware split scheduling

Once Q templates and regions are fixed, the remaining problem is to distribute each KV span across CTAs. CTAs from different requests, Q templates, and region types have different execution times, so equal KV partitions do not produce balanced workloads. A finer split increases KV parallelism but also creates more partial states and merge work; excessive splitting can introduce an additional wave.

For a candidate schedule π, CTAs with the same kernel variant and resource footprint form a class c. Let $W_c(\pi)$ denote its resident waves and $\hat{t}_a$ the profiled execution time of CTA a. We rank schedules by

$$\begin{aligned} \widehat{T}(\pi) = T_{\text{launch}}(\pi) + \sum_{c} \sum_{w \in W_c(\pi)} \max_{a \in w} \hat{t}_a \\ + T_{\text{merge}}(\pi), \\ \text{subject to } B_{\text{workspace}}(\pi) \le B_{\max}. \end{aligned} \tag{2}$$

The execution profile is indexed by Q template, KV-span class, and region type. Workspace is treated as a capacity constraint because partial-state traffic is already included in the execution and merge costs.

The scheduler evaluates a bounded set of legal splits, rejects those exceeding the workspace limit, and selects a finer split only when its wave-time reduction outweighs the added launch and merge costs. Appendix B gives the detailed procedure.

5 Evaluation

We evaluate RIVET at the kernel and inference levels. We first describe the experimental setup, then answer four questions: (1) how much do adaptive Q tiles and KV-aware splitting improve attention latency, (2) whether the improvement carries over to end-to-end speculative inference, (3) how much

each design component contributes, and (4) what overhead the additional structure and scheduling introduce.

5.1 Experimental Setup

Implementation. We implement RIVET on top of FlashInfer (Ye et al., 2025) (v0.6.14) and integrate it into SGLang (Zheng et al., 2024) (v0.5.16). The core prototype adds approximately 5.3K lines of code relative to the unmodified releases: 2.9K lines of Python, CUDA, and C++ in FlashInfer; and 2.5K lines of Python in SGLang. The implementation preserves FlashInfer's paged-KV interface, FA2 loaders and MMA path (Dao, 2024), and the online-softmax output contract used by tiled exact-attention kernels (Dao et al., 2022), so the proposed scheduler can coexist with the original native and custom-mask paths.

Testbed and metrics. Experiments run on a server with 8× NVIDIA A800-SXM4-80GB GPUs, each with 108 SMs and a 40 MiB L2 cache. The server has two Intel Xeon Gold 5320 CPUs and 512 GiB of host memory, and runs CUDA 12.9 with PyTorch 2.11.0. For kernel performance, we report median attention latency over 40 repetitions after 20 warm-up runs; for end-to-end performance, we report output throughput and median time per output token (TPOT).

Models and workloads. We evaluate Qwen3-8B and Llama 3.1-8B on long-context requests from LongBench (Bai et al., 2024) and SCBench (SharedContextBench) (Li et al., 2025a), and replay the public tool-agent request trace released with Mooncake (Qin et al., 2025; Mooncake Authors, 2025). All systems use identical prompts, target and draft checkpoints, and speculative-decoding parameters.

Baselines. We compare against four attention implementations: (1) *FlashInfer* (Ye et al., 2025) (v0.6.14), a query-centric engine that supports paged KV caches, customizable attention, and dynamic load balancing; (2) *Flash-Decoding* (Dao et al., 2023), a KV-centric design that partitions long KV sequences across CTAs and exactly merges partial softmax states; (3) *FlashAttention-2* (Dao, 2024) (v2.7.3), a query-centric, IO-aware implementation that fuses tiled attention with online softmax; and (4) *DeFT* (Yao et al., 2025), a tree-aware design that groups queries sharing KV segments and applies topology-aware reduction.

5.2 Kernel Performance

We vary the batch size from 8 to 32 and the KV length from 8K to 32K. Figure 4 reports inverse attention latency normalized to RIVET, which is fastest in all nine configurations.

Compared with query-centric kernels. RIVET reduces latency over FlashInfer and FlashAttention-2 by 51.6% and 80.0% on average, with maximum reductions of 71.8% and 92.6%. The gap generally widens with longer KV sequences because query-centric execution exposes insufficient parallelism for long-context verification and cannot adapt its query work to the varying candidate count.

Compared with KV- and tree-centric kernels. RIVET reduces latency over Flash-Decoding and DeFT by 47.5% and 69.0% on average, respectively. Flash-Decoding improves long-context parallelism by partitioning KV work, while DeFT exploits tree-level KV sharing; however, neither jointly adapts query tiling and KV partitioning to the runtime verification shape. The results show that coordinating both dimensions is necessary to avoid padding, load imbalance, and excessive merging.

5.3 End-to-End Inference Latency

We next evaluate end-to-end speculative decoding on the two models and three workloads. Figure 5 reports output throughput and TPOT across four batch-size and KV-length combinations.

Overall trends. RIVET achieves the highest throughput and the lowest TPOT in all 24 model–workload configurations. Compared with the strongest baseline in each configuration, selected separately for each metric, it improves throughput by 7.6–50.5% and reduces TPOT by 3.0–34.2%. All ranges report the minimum and maximum per-configuration percentage changes. Against FlashInfer,

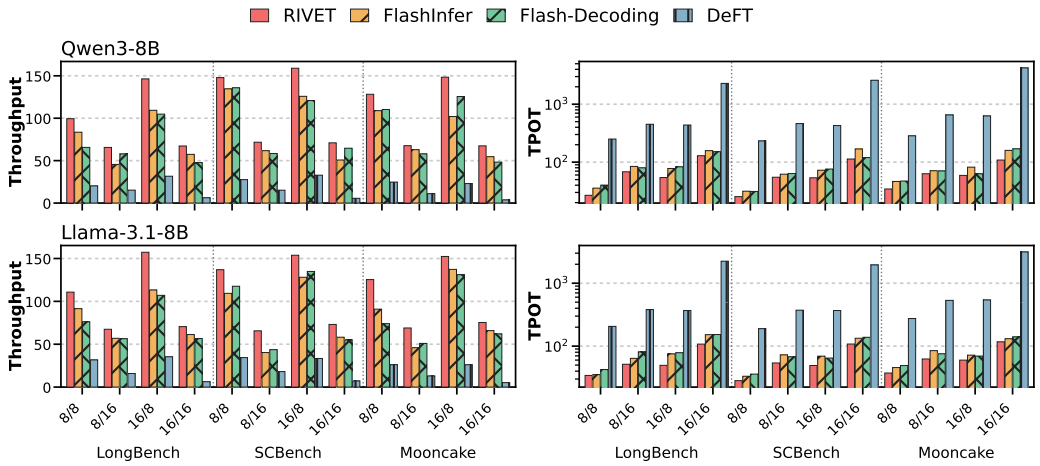
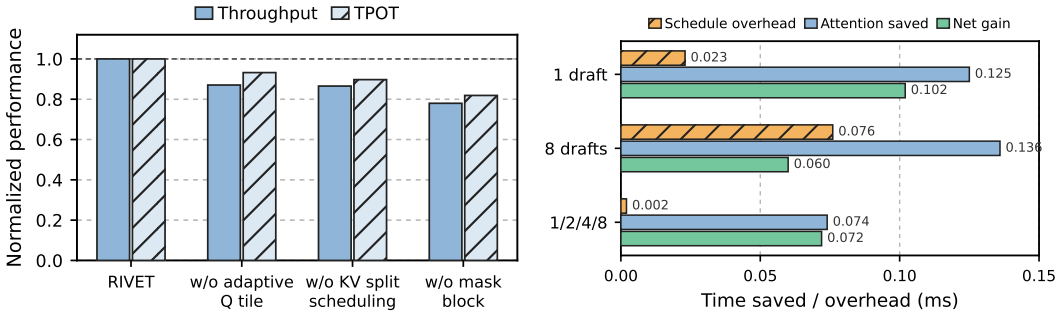


Figure 5: End-to-end throughput (tok/s, left) and median TPOT (ms, right, logarithmic scale). Each tick denotes batch size (requests)/KV length (K tokens), grouped by LongBench, SCBench, and Mooncake.



(a) Component ablation.

(b) Runtime scheduling overhead and attention savings.

Figure 6: Evaluation of the execution decisions and runtime scheduling cost. In (a), both measures are normalized to RIVET, so higher is better; TPOT is normalized by its reciprocal. In (b), values are measured in milliseconds and net gain is attention saved minus scheduling overhead.

throughput improvements range from 7.6% to 62.9% and TPOT reductions from 3.0% to 34.2%; against Flash-Decoding, the corresponding ranges are 8.8–69.5% and 5.6–37.0%. Thus, the gains appear in both aggregate output throughput and median per-token latency.

Scaling with batch size and KV length. The advantage remains positive across the four evaluated scales. Over the six model–workload pairs at each scale, throughput gains relative to the strongest baseline span 8.8–37.9% at $B8$/8K, 7.6–50.5% at $B8$/16K, 11.0–38.9% at $B16$/8K, and 9.8–25.6% at $B16$/16K. The corresponding TPOT reductions span 3.0–25.9%, 10.5–20.3%, 6.1–34.2%, and 5.6–31.9%, respectively. The gains do not grow monotonically with either batch size or KV length, indicating that execution benefits depend on the workload as well as the scale.

5.4 Ablation Study

We ablate adaptive Q tiles, KV-split scheduling, and compiled mask blocks on SCBench with batch size 8 and a 16K KV cache. Figure 6a reports throughput and inverse TPOT normalized to RIVET.

Disabling adaptive Q tiles and KV-split scheduling reduces throughput by 13.0% and 13.5%, and increases TPOT by 7.3% and 11.4%, respectively, confirming the benefits of reducing Q padding and balancing KV parallelism against merge cost. Reverting to generic custom-mask execution causes the largest degradation, reducing throughput by 22.0% and increasing TPOT by 22.1%, because regular

regions again incur mask metadata and predicate overhead. Thus, all three components contribute complementary gains. Appendix C isolates the effect of tree topology.

5.5 Overhead Analysis

The runtime scheduling path prepares region descriptors, selects a compiled Q template, and evaluates bounded legal KV splits. At $B16$/16K, its overhead ranges from 0.002 to 0.076 ms across one-draft, eight-draft, and mixed $1/2/4/8$ schedules, while the resulting execution saves 0.074–0.136 ms of attention time. Even the most expensive eight-draft case retains a 0.060 ms net saving; the one-draft and mixed schedules save 0.102 and 0.072 ms, respectively. Thus, the discrete scheduling overhead is amortized in every tested case.

6 Related Work

Speculative decoding and tree verification. Speculative sampling verifies drafted tokens in fewer target-model passes while preserving the target distribution (Leviathan et al., 2023; Chen et al., 2023). Multi-token prediction and block diffusion offer alternative draft mechanisms (Gloeckle et al., 2024; Chen et al., 2026). Medusa, EAGLE, and SpecInfer verify tree-structured candidates with ancestor-dependent masks (Cai et al., 2024; Li et al., 2024a; Miao et al., 2024). EAGLE-2 adapts draft-tree expansion to estimated acceptance probabilities, whereas EAGLE-3 changes how draft predictions are trained (Li et al., 2024b; 2025b). Sequoia optimizes the candidate tree with a hardware-aware objective, and SpecExec uses large draft trees to exploit parallel target-model execution under weight offloading (Chen et al., 2024; Svirschevski et al., 2024). These approaches construct or organize candidate trees; RIVET schedules attention for the resulting, potentially ragged tree.

IO-aware and long-context attention. FlashAttention uses tiling and online normalization to reduce attention memory traffic (Milakov & Gimelshein, 2018; Dao et al., 2022). FlashAttention-2 improves work partitioning, while FlashAttention-3 exploits asynchronous execution on Hopper GPUs (Dao, 2024; Shah et al., 2024). For decode attention, Flash-Decoding splits long KV spans and merges partial softmax states (Dao et al., 2023); FlashInfer supplies paged-KV kernels and configurable inference schedules (Ye et al., 2025). These techniques address data movement and GPU occupancy, but do not by themselves specialize visibility and tiling for a changing candidate tree.

Tree-aware attention kernels. DeFT groups queries that share KV segments, flattens tree KV accesses, and applies topology-aware reduction to improve reuse and load balance (Yao et al., 2025). RIVET instead compiles ancestor visibility into regular and mixed regions, chooses Q tiles from packed physical rows, and sizes KV partitions using resource and merge costs. Tree topology thus informs both the mask path and CTA scheduling.

7 Conclusion

Tree-based speculative decoding requires attention execution to adapt to both candidate topology and GPU resource constraints. We presented RIVET, which maps tree visibility to specialized regions, selects physical query tiles, and schedules KV partitions while preserving exact tree attention. Across nine kernel configurations, RIVET reduces attention latency by 51.6% on average relative to FlashInfer. Across 24 model–workload configurations, it improves output throughput by 7.6–50.5% and reduces median TPOT by 3.0–34.2% relative to the strongest baseline selected separately for each configuration and metric.

AI use statement

We used OpenAI Codex to assist with interpreting and checking reported benchmark results, editing and structuring manuscript text, revising scientific figures, and editing plotting scripts. Numerical claims were checked against the recorded measurements. The authors are responsible for reviewing all AI-assisted material and take responsibility for the final content of this paper.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023. doi: 10.18653/v1/2023.emnlp-main.298.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024. doi: 10.18653/v1/2024.acl-long.172.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 5209–5235, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*, 2024. URL https://arxiv.org/abs/2402.12374.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-Decoding for long-context inference. Stanford Center for Research on Foundation Models, 2023. URL https://crfm.stanford.edu/2023/10/12/flashdecoding.html. Accessed 2026-09-17.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 15706–15734, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 19274–19286, 2023.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV cache-centric analysis of long-context methods. In *International Conference on Learning Representations*, 2025a. URL https://aka.ms/SCBench.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 28935–28948, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, 2024b. doi: 10.18653/v1/2024.emnlp-main.422.

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

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024. doi: 10.1145/3620666.3651335.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018. URL https://arxiv.org/abs/1805.02867.

Mooncake Authors. Mooncake FAST'25 trace release. GitHub repository, 2025. URL https://github.com/kvcache-ai/Mooncake/tree/main/FAST25-release. Tool-agent request trace release.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021. URL https://arxiv.org/abs/2108.12409.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170. USENIX Association, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems*, volume 37, pp. 68658–68685, 2024. doi: 10.52202/079017-2193.

Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. doi: 10.1016/0022-0000(83)90006-5.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063.

Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. SpecExec: Massively parallel speculative decoding for interactive LLM inference on consumer devices. In *Advances in Neural Information Processing Systems*, volume 37, pp. 16342–16368, 2024. doi: 10.52202/079017-0522.

Jinwei Yao, Kaiqi Chen, Kexun Zhang, Jiaxuan You, Binhang Yuan, Zeke Wang, and Tao Lin. DeFT: Decoding with flash tree-attention for efficient tree-structured LLM inference. In *International Conference on Learning Representations*, 2025.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000.

A Decode-Stage Time Breakdown

Figure 7 shows the measured decode-stage time breakdown across two models, three batch sizes, and three KV lengths. Attention accounts for 30.62–68.36% of decode time across these configurations; at batch size 32, its share is at least 60.88% for both models at every KV length tested. This substantial attention cost motivates optimizing its execution during token-tree verification, although its time share alone does not determine the achievable end-to-end speedup.

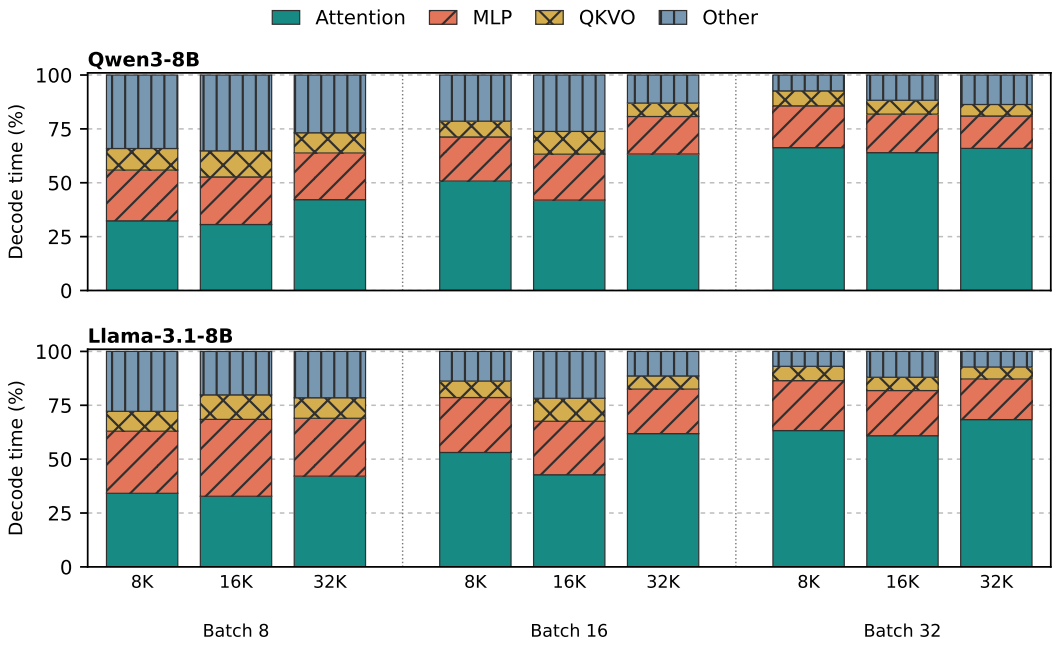


Figure 7: Decode-stage time breakdown for Qwen3-8B and Llama-3.1-8B. Each bar shows the percentage of time spent in attention, MLP, QKVO, and other operations; KV length is shown below each bar. Percentages may differ from 100% by 0.01 percentage points due to rounding.

B Bounded KV-split scheduling

The scheduling procedure enumerates a bounded set of legal split policies, filters policies that exceed the workspace budget, and ranks the remaining plans using Equation 2.

Algorithm 1 Bounded KV-split scheduling

Require: jobs J; legal policies P_j; workspace limit $B_{\max}$; search budget F
Ensure: selected schedule π^*

for all $j \in J$ **do**
  $C_j \leftarrow \varnothing$
  for all $p \in P_j$ **do**
    $A \leftarrow \textsc{GenerateCTAs}(j, p)$
    if $\textsc{Workspace}(A) \le B_{\max}$ **then**
      for all $a \in A$ **do**
        $\hat{t}_a \leftarrow \textsc{ServiceModel}(Q_a, K_a, R_a)$
      $C_j \leftarrow C_j \cup \{(p, A)\}$
$\Pi \leftarrow \textsc{BoundedCombine}(C_1, \ldots, C_{|J|}, F, B_{\max})$
for all $\pi \in \Pi$ **do**
  for all resource classes c in π **do**
    sort $A_c(\pi)$ by nonincreasing $\hat{t}_a$
    $W_c(\pi) \leftarrow \textsc{PackResidentWaves}(A_c(\pi))$
  score π with Equation 2
return $\arg\min_{\pi \in \Pi} \widehat{T}(\pi)$

C Topology Sensitivity

We compare the prefix–suffix hybrid and the topology-compiled region path while varying only the candidate-tree topology. Batch size ($B = 16$), KV length ($K = 16$K), candidate count (eight nodes per request), head ratio ($H_q/H_{kv} = 8$), and head dimension ($D = 128$) remain fixed. Table 1 reports median warm-cache attention latency. The region path is 0.6% faster on a chain but 4.2–4.9% slower on the branching trees. The latter have additional dense ancestor regions, consistent with higher CTA and merge costs. Both paths compute the same tree attention; their maximum measured absolute differences in output and log-sum-exp (LSE) are 4.77×10^{-7} and 9.54×10^{-7}, respectively. Thus, region compilation need not improve latency when branching introduces more execution regions.

Table 1: Topology sensitivity. Median warm-cache attention latency in μs (lower is better). Depth is the maximum root-to-leaf distance in edges; Paths and Regions count heavy paths and dense light-ancestor regions per tree, respectively. The random recursive tree is fixed.

Topology	Depth	Paths	Regions	Hybrid	Region
Chain	7	1	0	478.46	475.49
Star	1	7	6	476.98	500.28
Balanced	3	4	3	476.98	497.05
Random	4	3	2	476.88	497.20

D Exactness of Partitioned Attention

Fix one query and attention head, and let S be a nonempty subset of its visible keys. For each $k \in S$, let $s_k = QK_k^\top/\sqrt{d_k}$ be its attention score, with positional transformations applied at the original token positions. Define the online-softmax state

$$m_S = \max_{k\in S} s_k, \qquad \ell_S = \sum_{k\in S} e^{s_k - m_S}, \qquad u_S = \sum_{k\in S} e^{s_k - m_S} V_k. \tag{3}$$

The state represents both the attention output $O_S = u_S/\ell_S$ and the log-sum-exp $\mathrm{LSE}_S = m_S + \log \ell_S$. For disjoint nonempty subsets S and T, their union has state

$$\begin{aligned} m_{S\cup T} &= \max(m_S, m_T) = m, \\ \ell_{S\cup T} &= e^{m_S - m}\ell_S + e^{m_T - m}\ell_T, \\ u_{S\cup T} &= e^{m_S - m}u_S + e^{m_T - m}u_T. \end{aligned} \tag{4}$$

Indeed, $e^{m_S - m}\ell_S = \sum_{k\in S} e^{s_k - m}$ and $e^{m_S - m}u_S = \sum_{k\in S} e^{s_k - m}V_k$; the same identities hold for T. Disjointness therefore gives

$$\ell_{S\cup T} = \sum_{k\in S\cup T} e^{s_k - m}, \qquad u_{S\cup T} = \sum_{k\in S\cup T} e^{s_k - m} V_k. \tag{5}$$

The merged output $u_{S\cup T}/\ell_{S\cup T}$ and LSE $m + \log \ell_{S\cup T}$ equal those of attention over $S \cup T$ in exact arithmetic. Repeated merges thus give the same result for any disjoint partition and merge order; empty partitions are skipped.

E Correctness of Variable-Granularity Execution

We show that the execution decisions in RIVET preserve the visible key set and its attention result. The argument assumes that region descriptors follow the tree topology, each valid query–head row is emitted once, padded rows are discarded, and position-dependent operations use the original token positions.

Visibility coverage. Fix request i and a candidate query v on heavy path H_j with head h_j. Let $U_j = \mathrm{Anc}_i(h_j) \setminus \{h_j\}$ be the strict candidate ancestors of the path head. The ancestors of v consist exactly of U_j and the nodes from h_j to v within H_j. Since the shared prefix P_i is disjoint from candidate keys,

$$\mathrm{Visible}_i(v) = P_i \,\dot\cup\, U_j \,\dot\cup\, \big(H_j \cap \mathrm{Anc}_i(v)\big). \tag{6}$$

PrefixDense covers P_i; LightAncestor covers U_j; and PathCausal covers precisely $H_j \cap \mathrm{Anc}_i(v)$. The first two sets are dense for their respective queries, while the last is a causal prefix of the heavy path. The compiler excludes prefix and root overlaps when producing descriptors. Dense and causal blocks contribute only their permitted pairs, zero blocks contribute none, and mixed blocks test ancestry using $\mathrm{tin}(u) \le \mathrm{tin}(v) < \mathrm{tout}(u)$. For a DFS of the candidate tree, this interval condition is equivalent to $u \in \mathrm{Anc}_i(v)$. Thus, each visible query–key pair is included exactly once and every invisible pair is excluded.

Execution invariance. Packing the valid query–head rows into Q16, Q32, or Q64 tiles changes their physical layout, not their associated query, KV head, or original position. Padding adds no visible queries. For any valid row, KV splitting subdivides each compiled region into disjoint key subsets without changing its union. By Equation 6, these subsets partition $\mathrm{Visible}_i(v)$; by Equation 4, merging their states returns the tree-attention output and LSE in exact arithmetic. The chosen split size, CTA assignment, and merge order therefore affect execution cost but not the mathematical result.

Numerical scope. This establishes the absence of an algorithmic approximation, conditional on correct descriptor construction and kernel execution. Floating-point MMA, exponentiation, accumulation, and reassociation can change the rounded result, so the claim does not imply bitwise equality or a zero-error bound for a concrete kernel. Table 1 reports measured differences for its tested topologies; those measurements are not a universal accuracy bound.