Real-Time Sparse Tracing at Million-Token Context: IO-Optimal Compiled STREAM Kernels for Consumer GPUs

Liz Lemma Future Detective

January 18, 2026

Abstract

Mechanistic interpretability for long-context LLMs is bottlenecked by the quadratic cost of materializing attention maps. STREAM (Rosser et al., 2025) makes sparse attention tracing feasible in near-linear time by hierarchically pruning to top-k key blocks per query block, but its current implementation is not optimized for hardware and can be too slow for interactive use. We present a systems+algorithms co-design that compiles STREAM into fused GPU kernels that jointly (1) estimate hierarchical sparse masks, (2) apply masks to produce sparse attention summaries for interpretability, and (3) stream the resulting edges/weights to a lightweight tracing interface—without storing dense attention. We formalize sparse tracing as a streaming, block-wise top-k selection problem under a causal validity mask and analyze it in an external-memory/IO model. Our implementation achieves O(T log(T/b_k)) arithmetic work and O(T) memory for constant block sizes and sparsity, while its HBM traffic matches a natural lower bound up to constant factors. Empirically, we show interactive tracing for 100k–1M token contexts on consumer GPUs, with correctness verified against a reference STREAM specification and fidelity validated on long-context retrieval and reasoning traces. We release an open-source, reproducible benchmark suite and kernel library.

Table of Contents

1. 1. Introduction: long-context interpretability bottleneck; why asymptotics are not enough; goal of real-time sparse tracing; summary of contributions and guarantees.
2. 2. Background and Reference Spec: attention, sparse masks, STREAM/HiP hierarchical pruning; what needs to be computed for interpretability (masks, scores, flow edges).

3. 3. Problem Formulation: streaming sparse tracing under causal masks; define outputs (mask indices, weighted edges), fidelity hooks, and performance targets (latency, memory).

4. 4. Computational/IO Model: GPU memory hierarchy model (register/shared/HBM), block-sparse kernels, compilation constraints; define cost metrics (FLOPs, bytes moved, kernel launches).

5. 5. Kernel Design I — Mask Estimation: fused branch scoring + top-k selection; causal validity handling; deterministic tie-breaking; warp/block mapping; persistent kernels.

6. 6. Kernel Design II — Mask Application and Trace Emission: apply masks to attention computation (or to score summaries) and stream interpretability artifacts; overlap compute and output.

7. 7. Scheduling and Batching Strategies: head batching, layer pipelining, asynchronous prefetch, KV-cache interactions; handling variable k(q) (optional extension) without divergence blowups.

8. 8. Theoretical Analysis: work bounds, memory bounds, IO upper bounds; IO lower bounds and near-optimality; discussion of what is and is not provably optimal.

9. 9. Experimental Plan (for strengthening): profiling methodology; correctness checks vs reference; end-to-end latency and memory; scaling to 1M tokens; benchmark tasks (RULER, long CoT) and stability across hardware.

10. 10. Limitations and Future Work: generality beyond attention-only; multi-GPU/distributed; certified fidelity; variable sparsity schedules; integration with interpretability toolchains.

11. Appendices: reference spec, tie-breaking rules, kernel pseudocode details, reproducibility checklists.

1 Introduction

In decoder-only transformers with context length T in the range 10^5–10^6, interpretability procedures that rely on attention access encounter an immediate external-memory bottleneck: even if one avoids backpropagation and computes only forward-pass quantities, any attempt to log or post-process dense attention requires addressing $\Theta(T^2)$ token pairs. On commodity GPUs, both HBM capacity and HBM bandwidth make such a strategy infeasible. The obstruction is not merely algorithmic in the RAM model, but structural in an external-memory model: producing T^2 scores entails $\Omega(T^2)$ writes, and reading the keys required to form them entails commensurate traffic. Consequently, for long-context tracing, the central question is not how to accelerate dense attention, but how to change the output interface so that the amount of information emitted is itself subquadratic while remaining faithful to a precisely specified notion of sparse interpretability.

We therefore adopt the viewpoint that an interpretability-oriented attention procedure should output a sparse object of size $O(Tk)$, where k is a fixed (or scheduled) sparsity parameter, together with optional streamed edge weights, and should do so without materializing the dense score matrix $S = QK^\top$. However, asymptotic sparsity alone does not deliver an implementable method. Two further issues dominate at $T \gg 10^5$. First, the GPU cost is governed by global-memory traffic rather than floating-point throughput, so a method that is asymptotically sparse but revisits HBM repeatedly can still be dominated by IO. Second, interpretability pipelines are sensitive to nondeterminism: small perturbations in sparse selection rules (e.g., from race conditions in parallel top-k) can lead to qualitatively different trace graphs, obstructing reproducibility and regression testing. For these reasons we insist on a reference-equivalent specification with deterministic tie-breaking, and on an implementation strategy whose IO can be compared to information-theoretic lower bounds.

Our goal is real-time sparse tracing: for each layer and head, we seek to (i) estimate a block-sparse causal mask by selecting at most k key blocks per query block, (ii) optionally apply the resulting mask to compute sparse summaries or emit an edge list, and (iii) do so with auxiliary memory $O(T)$ rather than $O(T^2)$. We formalize this goal using a block structure with query block size b_q, key block size b_k, and a block-level validity mask $C_{\text{blk}} \in \{0,1\}^{(T/b_q)\times(T/b_k)}$ capturing causal and task-specific constraints. The resulting output is an index tensor $I \in \mathbb{N}^{(T/b_q)\times k}$ giving the retained key-block indices per query block (or empty entries when no valid key exists).

The contributions of this work are systems-theoretic and specification-driven. We present a compilation-friendly, GPU-executable implementation of STREAM-style hierarchical pruning that operates by iterative branch refinement over the key-block range, scoring only $O(k \log(T/b_k))$ representative key blocks per query block. The implementation is designed to be persistent and tiled: each query block loads Q once into on-chip storage, streams only the representative K blocks required by the refinement, performs per-candidate scoring via blockwise maxima of dot products in $\mathbb{R}^d$, and performs exact top-k selection with a fixed tie-breaking rule τ that induces a total order on candidates. Under these constraints, we obtain three guarantees. (1) *Reference-equivalence*: the emitted indices I are bitwise identical to those of a reference STREAM specification under τ. (2) *Asymptotic efficiency*: for constant (b_q, b_k, k, d), mask estimation has work $O(T \log(T/b_k))$ and auxiliary space $O(T)$ per head/layer, with no dense $T \times T$ buffers. (3) *Near-optimal IO*: the HBM traffic for mask estimation is within a constant factor of an information-theoretic lower bound that accounts for reading Q and accessing the selected key blocks. These guarantees jointly justify that interactive tracing at long context is attainable on a single consumer GPU only by avoiding dense attention materialization and by enforcing deterministic, specification-aligned sparsity throughout.

2 Background and Reference Specification

We work with a decoder-only transformer layer and fix a head. Let T be the context length and d the per-head embedding dimension. The layer provides query, key, and value matrices

$$Q, K, V \in \mathbb{R}^{T \times d},$$

stored in HBM. The dense attention score matrix would be

$$S \;=\; QK^\top \in \mathbb{R}^{T \times T}, \qquad S_{t,s} = \langle Q_{t,:}, K_{s,:} \rangle,$$

and a token-level validity mask $C \in \{0,1\}^{T \times T}$ enforces causal constraints (and optional task masks). Dense interpretability procedures that require access to S or to the dense attention probabilities $\mathrm{softmax}(S \odot C)$ are excluded by our constraints; hence we formalize a sparse interface that never requires materializing S.

We impose a block structure. Fix query-block and key-block sizes b_q and b_k, and assume padding so that T is divisible by $\mathrm{lcm}(b_q, b_k)$. We write the query-block index as $q \in \{1, \dots, T/b_q\}$ and the key-block index as $r \in \{1, \dots, T/b_k\}$. The block-level validity mask

$$C_{\mathrm{blk}} \in \{0,1\}^{(T/b_q) \times (T/b_k)}$$

summarizes C at block resolution (e.g. $C_{\mathrm{blk}}[q, r] = 1$ if and only if there exists at least one valid token pair (t, s) with t in query block q and s in key block r). For causal masking, $C_{\mathrm{blk}}[q, r] = 0$ when key block r lies strictly to the right of query block q, with the usual boundary conventions.

A sparse mask is specified by indices rather than by a dense binary matrix. Fix a sparsity parameter k (or a known schedule $k(q)$). The primary interpretability object for a head/layer is an index tensor

$$I \in \mathbb{N}^{(T/b_q)\times k},$$

where $I[q, j] = r$ indicates that key block r is retained for query block q (and entries may be empty when no valid key blocks exist). This induces a token-level sparse mask M conceptually defined by

$$M_{t,s} = 1 \quad \Longleftrightarrow \quad \exists j \in \{1, \ldots, k\} \text{ such that } s \in \text{block } I[q(t), j],$$

where $q(t)$ is the query-block index of token t. We emphasize that M is not materialized densely; it is represented by I and used only through streaming/tiled application.

We now fix a *reference* selection rule, in the style of STREAM/HiP hierarchical pruning, to make the sparse interface deterministic and testable. For each query block q, we consider the ordered range of key-block indices $\{1, \ldots, T/b_k\}$ and maintain k disjoint *branches*, each being a contiguous subrange of key-block indices. Initially these branches cover the full range in a fixed manner (e.g. equal-length partitioning, with a deterministic convention for remainders). For $n_{\text{it}} = \lceil \log_2(T/b_k) \rceil$ iterations, each branch is split into two sub-branches, producing $2k$ candidate subranges. Each candidate c is assigned a *representative* key block $r(c)$ defined as the first key-block index in the subrange that is valid under $C_{\text{blk}}[q, *]$; if no such index exists, the candidate is invalid.

Each valid candidate receives a scalar score intended to proxy the best achievable token-level interaction within that representative block:

$$\text{score}(c) \;=\; \max_{\substack{1 \le m \le b_q \\ 1 \le n \le b_k \\ C[t(q,m),\, s(r(c),n)]=1}} \big\langle Q_{t(q,m),:},\; K_{s(r(c),n),:} \big\rangle,$$

where $t(q, m)$ and $s(r, n)$ denote the corresponding token indices within blocks. Candidates are then filtered to the top-k by score, with ties broken by a fixed total order τ over candidates (equivalently, over (q, c) pairs). The retained k sub-branches define the next iteration. After n_{it} refinements, we output $I[q, 1..k]$ as the first valid key-block index within each final branch (or empty if none exists). This *reference specification* is purely functional: it defines I uniquely from $(Q, K, C_{\text{blk}}, b_q, b_k, k, \tau)$.

Finally, interpretability often requires more than indices. Given I, we may stream *trace artifacts* such as a sparse edge list over blocks, e.g. tuples (q, r, w) where $r \in I[q, *]$ and w is either the representative score above, a blockwise reduction of masked attention probabilities, or a downstream flow quantity. The essential constraint is that all such weights are computed by iterating only over retained blocks and by streaming Q, K, V tiles, never forming dense $T \times T$ intermediates.

3 Problem Formulation: Streaming Sparse Tracing Under Causal Masks

We formalize the systems task induced by sparse interpretability in a decoder-only layer as a *streaming* map from per-head/layer activations and masks to a compact trace. The inputs are the matrices $(Q, K, V) \in \mathbb{R}^{T\times d}$ residing in HBM together with the token-level validity mask $C \in \{0,1\}^{T\times T}$ (including causality) and its block summary C_{blk}, as well as fixed parameters (b_q, b_k, k) (or a known schedule $k(q)$) and a deterministic tie-breaker τ. The output is required to be representable with $O(T)$ auxiliary space per head/layer and must be computable without materializing dense score or probability matrices.

Primary output: block index mask. For each query block $q \in \{1, \dots, T/b_q\}$ we must output exactly k retained key-block indices (or empties when no valid keys exist), assembled as

$$I \in \mathbb{N}^{(T/b_q)\times k}.$$

We allow a fixed sentinel (e.g. 0) to denote an empty entry. The indices must satisfy the validity invariant

$$I[q,j] \neq 0 \implies C_{\text{blk}}[q, I[q,j]] = 1,$$

and must be *reference-equivalent*: for every (Q, K, C_{blk}) the produced I is bitwise identical to the indices returned by the functional reference specification of the previous section, under the same padding convention and tie-breaking rule τ. This reference-equivalence requirement is the fidelity hook that makes the procedure testable: it reduces correctness to equality with a deterministic oracle that can be evaluated for moderate T.

Secondary output: streamed trace artifacts. In addition to I, we permit (and in interpretability settings often require) a streaming edge list over retained blocks. Formally, for each head/layer we may emit a sequence

$$\mathcal{E} = \big\{(q, r, w)\big\} \qquad \text{with} \quad r \in \{I[q,1], \dots, I[q,k]\},$$

where the weight w is computed by an explicitly specified per-edge functional. We restrict admissible weight functionals to those evaluable by tiled streaming over the corresponding query and key/value blocks without dense intermediates. Concretely, w must be of the form

$$w \;=\; \Phi\Big(Q_{B_q(q),:},\, K_{B_k(r),:},\, V_{B_k(r),:},\, C_{B_q(q),B_k(r)}\Big),$$

where $B_q(q)$ and $B_k(r)$ denote the token index sets in the respective blocks. Typical choices include: (i) the representative selection score used during mask estimation; (ii) a blockwise attention mass, e.g. $\sum_{t \in B_q(q)} \sum_{s \in B_k(r)} p_{t,s}$ where $p = \mathrm{softmax}(S \odot C)$ is evaluated only on the masked support induced by I; or (iii) a blockwise value summary such as $\sum_{t \in B_q(q)} \sum_{s \in B_k(r)} p_{t,s} V_{s,:}$. The interface permits either writing $\mathcal{E}$ into a device buffer of size $O((T/b_q) \cdot k)$ or emitting it through a streaming sink (e.g. host-mapped memory), but in all cases the algorithm must not allocate $O(T^2)$ storage.

Streaming and composability constraints. We treat mask estimation and mask application/edge emission as a single end-to-end workload. The implementation may fuse phases, reorder within a head/layer, and pipeline across query blocks, but it must respect the following global constraints: (a) neither $S = QK^\top$ nor any dense mask/probability matrix may be materialized; (b) extra HBM allocations beyond the outputs and $O(1)$ staging buffers are disallowed; and (c) the produced indices and weights must be deterministic functions of inputs under τ, independent of parallel scheduling.

Performance targets. Our objective is to minimize end-to-end latency for producing $(I, \mathcal{E})$ for all query blocks, heads, and layers. Since long-context regimes are bandwidth dominated, we treat global-memory traffic as the primary cost to optimize, subject to keeping arithmetic work and synchronization overhead controlled. We thus target (i) HBM reads/writes close to the minimum implied by reading Q and the necessary key/value blocks, (ii) auxiliary space linear in the number of emitted edges, and (iii) a small number of kernel launches and global synchronization points so that tracing remains interactive at $T \in [10^5, 10^6]$ on a single GPU.

4 Computational and IO Model

We analyze STREAM-style mask estimation and trace emission in a GPU external-memory model with three storage levels: registers (REG), shared memory (SMEM), and high-bandwidth memory (HBM). We treat REG and SMEM as fast, explicitly managed caches with bounded capacity per thread block, whereas HBM is large but bandwidth limited. Our algorithms are required to stream through activations stored in HBM and to maintain only $O(1)$ on-chip state per concurrent query block, so that asymptotic feasibility is governed by HBM traffic rather than capacity.

Tiled block-sparse execution. All computation is organized around query blocks of b_q consecutive tokens and key blocks of b_k consecutive tokens. For each head and layer, we conceptually partition $Q, K, V \in \mathbb{R}^{T \times d}$ into blocks

$$Q_q \in \mathbb{R}^{b_q \times d}, \qquad K_r, V_r \in \mathbb{R}^{b_k \times d},$$

indexed by $q \in \{1, \dots, T/b_q\}$ and $r \in \{1, \dots, T/b_k\}$. The masking constraint is given at block resolution by $C_{\text{blk}} \in \{0,1\}^{(T/b_q)\times(T/b_k)}$, which subsumes causality and any additional user masks. The implementation is permitted to examine only a small, adaptively chosen subset of key blocks per query block, and must never form dense intermediates of size $T \times T$.

Cost metrics. We account for three costs. First, arithmetic work $\mathrm{Work}(\cdot)$ measured in floating-point operations; this includes dot products, reductions (e.g. top-k selection), and any masked softmax/value projections used to produce trace weights. Second, global-memory traffic $\mathrm{IO}(\cdot)$ measured in bytes transferred between HBM and the GPU (reads and writes), including both activation loads and output stores. Third, launch and synchronization overhead, measured by the number of kernel launches and any global barriers between phases. In long-context regimes, we assume runtime is well approximated by

$$\mathrm{Time} \;\approx\; \max\Big\{\mathrm{IO}/\mathrm{BW}_{\mathrm{HBM}},\ \mathrm{Work}/\mathrm{FLOP}_{\mathrm{peak}}\Big\} \;+\; \mathrm{Overhead}_{\mathrm{launch/sync}},$$

with the understanding that effective bandwidth depends on coalescing and reuse.

HBM as the dominant bottleneck. Since T is large and d, b_q, b_k, k are moderate constants, the primary opportunity is to reduce redundant HBM reads of key/value blocks and to avoid writing large intermediate tensors. We therefore design kernels around (i) loading each Q_q once into SMEM (or REG fragments), (ii) streaming only the representative key blocks required by the hierarchical procedure, and (iii) emitting only $O((T/b_q)\cdot k)$ outputs (indices and optional edge weights). The resulting objective is to make IO close to the unavoidable reads of Q and the necessary reads of the selected K/V blocks, and to ensure that any additional reads (e.g. for candidate evaluation during refinement) are bounded by polylogarithmic factors in T/b_k.

On-chip workspace constraints. For a thread block responsible for one query block (or a small fixed number thereof), the admissible on-chip footprint is on the order of

$$O(b_q d) \text{ for } Q_q \quad + \quad O(b_k d) \text{ for one streamed } K_r \text{ (and optionally } V_r) \quad + \quad O(k) \text{ scalars},$$

along with temporary storage for reductions. This requirement rules out buffering many key blocks simultaneously; instead we pipeline: load a key block, compute its score contribution against Q_q, reduce to a scalar representative score, and discard the block before loading the next.

Compilation-friendly specialization. We assume that (d, b_q, b_k) and either k or a small discrete family of schedules $k(q)$ are known at compile time (or selected from a small menu), enabling kernel specialization with static loop bounds, fixed SMEM allocation, and predictable register pressure. This assumption is not merely stylistic: exact top-k selection and deterministic tie-breaking τ are simplest to implement efficiently when k is small and fixed, so that the reduction network and comparison order can be compiled into straight-line or lightly branched code. Similarly, tensor-core utilization benefits from fixed d and tile shapes, allowing dot-product fragments to be scheduled without dynamic shape handling.

Determinism under parallelism. We treat determinism as a systems constraint that interacts with the cost model. Reductions over candidate scores must be performed in a fixed associative order with explicit tie-breaking τ, avoiding nondeterministic outcomes due to race conditions or unspecified warp scheduling. Practically, this means that within each query block we employ structured reduction patterns (warp-level primitives and block-level merges) whose comparison order is fixed by program structure; the extra comparisons required to enforce τ are counted in $\mathrm{Work}(\cdot)$ but are negligible relative to the dot-product evaluation for moderate $b_q b_k d$.

Implications for kernel structure. Under this model, the intended implementation strategy is to minimize (a) the number of distinct key blocks touched per query block during refinement and application, and (b) the number of passes over Q_q. Consequently, we favor persistent or quasi-persistent kernels that keep Q_q resident on chip while iteratively streaming candidate K_r blocks, performing branch scoring and top-k selection in situ, and emitting indices (and optionally weights) without materializing dense masks. These design commitments are instantiated concretely in the next section, where we specify the fused mask-estimation kernel and its mapping to warps and thread blocks.

5 Kernel Design I: Mask Estimation

We now specify the fused GPU kernel that computes the sparse mask indices $I[q, 1..k]$ for each query block q without materializing either dense scores $S = QK^\top$ or a dense mask M. The implementation we target is *reference-equivalent*: for fixed (b_q, b_k, k), padding, and tie-breaking rule τ, the emitted indices coincide bitwise with the reference STREAM procedure described in the global specification.

Fused refinement: branch scoring plus top-k. For a fixed head/layer, we launch a (quasi-)persistent kernel over query blocks $q \in \{1, \ldots, T/b_q\}$.

Each thread block (CTA) is responsible for one q (or a small fixed number), loads $Q_q \in \mathbb{R}^{b_q \times d}$ once into SMEM (or REG fragments), and then performs $n_{\text{it}} = \lceil \log_2(T/b_k) \rceil$ refinement iterations. At iteration i, the kernel maintains k active branches, each corresponding to an interval of key-block indices. We split each branch into two sub-branches, producing $2k$ candidates, and compute for each candidate c a scalar score

$$\text{score}[c] \;=\; \begin{cases} \max\limits_{\substack{m \le b_q,\, n \le b_k \\ \text{token-valid}}} \langle Q[q, m, :],\, K[r, n, :] \rangle, & r = \text{firstValid}(q, \text{range}(c)), \\ -\infty, & \text{if no valid } r \text{ exists in range}(c), \end{cases}$$

where firstValid is the deterministic representative rule (first valid key block in the sub-branch), and "token-valid" subsumes the token-level constraint C restricted to the chosen blocks. We then select the top-k candidates by (score, τ) and proceed with those k sub-branches.

Causal and user validity at block resolution. We assume a block mask $C_{\text{blk}}[q, r] \in \{0, 1\}$ is available (bit-packed in HBM), encoding causality and any additional user constraints. For each candidate interval $[a, b]$, we must (i) decide whether any valid $r \in [a, b]$ exists, and (ii) if so, find the smallest such r to satisfy the representative rule. We implement both using deterministic word-level scans over the packed row $C_{\text{blk}}[q, *]$: an "any" test is a fixed-order OR reduction over machine words covering $[a, b]$, and firstValid is a fixed-order scan for the first set bit. Since interval sizes halve each iteration, these scans contribute at most a polylogarithmic overhead and, crucially, do not require storing per-q auxiliary structures beyond a small constant amount of on-chip state.

Tensor-core scoring without dense materialization. Given Q_q resident on chip and a streamed key block $K_r \in \mathbb{R}^{b_k \times d}$, we must compute $\max_{m,n} \langle Q[q, m, :], K[r, n, :] \rangle$ without forming the $b_q \times b_k$ matrix. We proceed by tiled MMA: interpret Q_q and K_r as fragments and compute products for small tiles (e.g. 16×16) using tensor cores, but immediately reduce each produced tile to a running maximum in registers. Thus the only persistent per-candidate state is a small set of maxima scalars (one per thread or per warp, then reduced), rather than an explicit score matrix.

Warp/block mapping and pipelining. Within a CTA, warps are assigned to candidates in a fixed schedule: for moderate k, we map one warp to one candidate when $2k \le W$ (number of warps per CTA), otherwise we process candidates in rounds. Each warp iterates over tiles of Q_q and K_r, accumulating a local maximum, followed by a fixed-order warp reduction and a fixed-order block reduction to obtain score$[c]$. To reduce HBM stalls, we pipeline key-block loads with compute (e.g. double-buffer K_r in SMEM and use asynchronous copies where available), while keeping Q_q stationary.

Deterministic top-k with tie-breaking τ. Top-k selection is performed on the $2k$ scalar scores using a fixed comparison network whose comparison order is independent of runtime scheduling. We impose τ by comparing lexicographically on a tuple such as

$$\big(\text{score}[c],\ -r(c),\ -\text{branchId}(c)\big),$$

or any other globally fixed total order consistent with the reference specification; the precise fields are compile-time fixed and ensure that equal scores yield a unique winner. Implementationally, we keep the $2k$ tuples in registers, apply a deterministic bitonic/odd-even network (or an unrolled partial selection for small k), and retain the first k entries. Because both the reduction of scores and the selection network have fixed associativity/ordering, the result is invariant to warp interleavings, satisfying the determinism invariant required by Theorem 1.

Persistent-kernel rationale. The salient systems property is that each Q_q is loaded once and reused across all refinement iterations, while only $O(kn_{\text{it}})$ representative key blocks are streamed per query block. This is precisely the structure needed to make $\text{IO}(\cdot)$ approach the lower bound up to constant factors, while keeping on-chip storage bounded by $O(b_q d + b_k d + k)$. The next section uses the computed indices I to apply the resulting sparse mask and to emit trace artifacts in a similarly streaming fashion.

6 Kernel Design II: Mask Application and Trace Emission

Given mask indices $I[q, 1..k]$ from the estimation kernel, we next apply the induced sparse pattern to (i) produce the masked attention output (when required) and/or (ii) emit token- or block-level trace artifacts for interpretability. The central constraint remains that we do not materialize dense $T \times T$ scores or probabilities; all computation and emission proceed by streaming over only the selected key blocks.

Execution model and data movement. For each head/layer we launch a kernel over query blocks q. Each CTA loads $Q_q \in \mathbb{R}^{b_q \times d}$ once (or reuses it if mask estimation and application are fused). Then, for $t = 1, \ldots, k$, it reads the retained key-block index $r = I[q, t]$ (or skips if empty), streams $K_r \in \mathbb{R}^{b_k \times d}$ and, when producing attention outputs, also streams $V_r \in \mathbb{R}^{b_k \times d_v}$. The only global writes are (a) the attention output $O_q \in \mathbb{R}^{b_q \times d_v}$ when requested and (b) an $O(k)$-sized trace record per query block when tracing is enabled. Thus the HBM traffic scales with the touched key/value blocks rather than with T^2.

Sparse attention in a single streaming pass. When we must compute

$$O[q,m,:] \;=\; \sum_{(r,n)\in\mathcal{N}(q,m)} \alpha_{m,(r,n)}\, V[r,n,:], \qquad \alpha_{m,(r,n)} = \frac{\exp(\ell_{m,(r,n)})}{\sum_{(r',n')\in\mathcal{N}(q,m)} \exp(\ell_{m,(r',n')})},$$

we define $\ell_{m,(r,n)} = \langle Q[q,m,:], K[r,n,:]\rangle$ with $\ell = -\infty$ for token-invalid pairs under C. Here $\mathcal{N}(q,m)$ ranges over the k selected key blocks and their b_k tokens, restricted by causality and any user mask. We implement a numerically stable online softmax update (FlashAttention-style) without storing logits: for each query token m we keep registers $(\mu_m,\ \lambda_m,\ o_m)$ representing the running maximum, normalization, and partial output. Upon processing a new block r we compute logits for its $n \in \{1,\dots,b_k\}$, apply the token-validity predicate, and update

$$\mu'_m = \max\bigl(\mu_m,\ \max_n \ell_{m,(r,n)}\bigr), \qquad \lambda'_m = \lambda_m\, e^{\mu_m-\mu'_m} + \sum_n e^{\ell_{m,(r,n)}-\mu'_m},$$

$$o'_m = o_m\, e^{\mu_m-\mu'_m} + \sum_n e^{\ell_{m,(r,n)}-\mu'_m}\, V[r,n,:].$$

After all retained blocks, we output $O[q,m,:] = o_m/\lambda_m$. All dot products and accumulations are tiled; we use tensor cores for $Q_q K_r^\top$ tiles and immediately reduce to the scalars needed for (μ'_m, λ'_m) and the weighted V accumulation, never storing a $b_q \times b_k$ tile beyond registers.

Trace artifacts as streaming edge emission. For interpretability we optionally emit an edge stream whose granularity is per (q-block, r-block). For each retained $r = I[q,t]$ we compute a small summary $\phi(q,r)$, e.g. the blockwise maximum logit $\max_{m,n} \ell_{m,(r,n)}$, a blockwise mass contribution $\sum_{m,n} \alpha_{m,(r,n)}$ (accumulated consistently with the online softmax), or a fixed set of per-head scalars. We then write records

$$E \ni \bigl(\ell,\ h,\ q,\ r,\ \phi(q,r)\bigr)$$

in the deterministic order induced by $t = 1,\dots,k$ and the fixed head/layer schedule. To avoid global synchronization, CTAs reserve output space via a single atomic increment on a global write pointer, then perform coalesced writes. If buffering is needed, we use a small SMEM ring buffer and flush in fixed-size segments; the record order within a CTA is fixed by construction, hence independent of warp interleavings.

Overlapping compute with output. We pipeline (a) loads of K_r, V_r, (b) MMA-based dot products and softmax updates, and (c) trace writes. Concretely, we double-buffer K_r, V_r in SMEM and overlap asynchronous copies with computation on the previous buffer. Trace records are staged in registers/SMEM and written while the next key block is being prefetched. This overlap is essential when tracing is enabled, since the edge stream adds nontrivial write traffic of $\Theta((T/b_q)\,k)$ records.

Correctness and invariants. Mask application respects the validity invariant by applying C at token resolution within each selected block and by skipping empty $I[q, t]$. Determinism holds because (i) the order of visiting retained blocks is fixed by $I[q, 1..k]$, (ii) reductions use fixed associativity/ordering inside each CTA, and (iii) the emitted trace stream is written in a fixed per-CTA order with a unique reserved output segment. These properties allow scheduling and batching optimizations in the next section without changing observable outputs.

7 Kernel Design III: Scheduling and Batching Strategies

We describe schedules that minimize launch overhead and HBM traffic while preserving the determinism and validity invariants. The guiding constraint is that we must stream blocks in the fixed order prescribed by (ℓ, h, q, t) and $I[q, t]$, hence we admit only transformations that do not reorder observable outputs.

Head and sequence batching. We parallelize primarily over query blocks q and heads h. For fixed (b_q, b_k, d) we prefer a 2D grid with indices (q, h), assigning one CTA to one (q, h) pair. This yields uniform on-chip footprints (one Q_q tile plus one K_r/V_r tile) and avoids cross-head synchronization. When the number of heads is small relative to SM count, we batch multiple heads per CTA by storing Q_q for each head in registers and sharing the streamed K_r/V_r tiles in SMEM (the latter is beneficial only when K, V are shared across heads, e.g. in MQA/GQA). For multi-sequence batches with ragged lengths, we treat each sequence as an independent range of blocks and encode invalid blocks in C_{blk}; this keeps kernel shapes static while allowing empty work to be skipped by cheap predicates.

Persistent CTAs and load balancing. Both mask estimation and application have per-q work that depends on the number of valid keys (through C_{blk}) and on the realized pattern $I[q, *]$. To avoid tail effects we use persistent CTAs: each CTA repeatedly acquires the next available q from a global counter, processes it to completion (including trace emission), and then acquires another. Determinism is unaffected because the observable order is per-q local (fixed $t = 1, \ldots, k$), while the global edge stream is ordered by the reserved output segments rather than by CTA completion time.

Layer pipelining and fusion. Across layers ℓ, application depends on the layer input activations, hence full pipelining is limited by the forward dependency. Nevertheless, two optimizations are compatible with correctness. First, we fuse mask estimation and application within a single kernel when tracing requires only $\phi(q, r)$ and (optionally) O_q: we compute $I[q, *]$ in registers, immediately stream the corresponding K_r, V_r blocks, and emit both O_q and trace records, thereby avoiding an intermediate write/read of I from HBM. Second, when indices must be retained (e.g. for later reuse), we pipeline across heads: while head h performs application for query block q, head $h + 1$ can perform estimation for q in a separate CUDA stream, since the data are disjoint and determinism is enforced by a fixed head schedule and disjoint output segments.

Asynchronous prefetch and double buffering. For both estimation and application, the critical path is the streaming of K_r (and V_r) blocks for the realized representative indices. We therefore employ double-buffered SMEM tiles and asynchronous copies. Concretely, while computing on buffer p (MMA for logits and reductions for μ, λ, o or for candidate scoring), we prefetch the next required block into buffer $1 - p$ using asynchronous transactions, and we synchronize only at the tile boundary. This converts HBM latency into bandwidth-limited throughput provided the arithmetic intensity of the $b_q \times b_k \times d$ tile is sufficient.

KV-cache interactions. In autoregressive decoding, K, V are appended over time and typically stored in a paged KV cache. Our method is compatible provided the mapping from logical block index r to physical address is deterministic and queryable without synchronization. We store a page table in HBM (or, when small, in constant memory) and perform address translation per retained $r = I[q, t]$. To preserve coalescing, we choose the cache layout so that each logical block is contiguous and aligned, and we keep b_k equal to the cache block granularity. Causality is enforced by C_{blk} (and token-level C within the block), so the same kernels apply to both prefill and decode; in decode, only the final query block(s) are active and persistent CTAs prevent underutilization.

Handling variable $k(q)$ without divergence. When sparsity varies by query block, we require compilation-friendly control flow. We implement either (i) *padding*: fix a compile-time $k_{\max}$, run all loops for $t = 1, \ldots, k_{\max}$, and treat $t > k(q)$ as empty (scores $= -\infty$, no trace write) while still reserving a fixed-size output slot; or (ii) *bucketing*: partition query blocks into a small set of buckets by $k(q) \in \{k_1, \ldots, k_s\}$, and launch specialized kernels per bucket. Padding eliminates divergence entirely at the cost of redundant work, whereas bucketing preserves work-efficiency with a bounded number of kernel variants. In both cases, tie-breaking τ and the per-q visitation order are unchanged, hence reference-equivalence is preserved.

8 Theoretical Analysis: Work, Space, and IO Bounds

We analyze the compiled STREAM procedure in the external-memory model described above, under fixed (b_q, b_k, d) and fixed sparsity k (or a known schedule $k(q)$ implemented by padding/bucketing without changing the underlying control decisions). Let $n_q := T/b_q$ denote the number of query blocks and $n_k := T/b_k$ the number of key blocks. The hierarchical refinement depth is

$$n_{\text{it}} \;:=\; \lceil \log_2 n_k \rceil \,,$$

corresponding to repeatedly bisecting a contiguous key-block range until single blocks are isolated.

Work bound for mask estimation. For each query block q, each refinement iteration produces $2k$ candidate sub-branches. For each candidate, the compiled kernel selects a representative block index r (the first valid block under $C_{\text{blk}}[q, *]$ in that sub-branch, deterministically), and computes the representative score

$$\text{score}(q, r) \;=\; \max_{\substack{1 \le m \le b_q \\ 1 \le n \le b_k \\ C[q_m, k_n] = 1}} \langle Q[q, m, :],\, K[r, n, :] \rangle,$$

which evaluates $\Theta(b_q b_k)$ dot products in $\mathbb{R}^d$ and then reduces by a maximum. Hence each candidate score costs $\Theta(b_q b_k d)$ arithmetic operations, and each iteration costs $\Theta(2k\, b_q b_k d)$. Summing over n_{it} iterations and n_q query blocks yields

$$\text{Work}_{\text{est}} \;=\; \Theta\big(n_q \cdot k \cdot n_{\text{it}} \cdot b_q b_k d\big) \;=\; \Theta\big(T \cdot k \cdot b_k d \cdot \log_2(T/b_k)\big),$$

which is $O(T \log(T/b_k))$ for constant parameters, in agreement with Theorem 2. Exact top-k selection among $2k$ candidates per iteration incurs an additional $\Theta(k)$ comparison cost, which is subsumed by the dot-product work when $b_q b_k d$ is nontrivial, but is nevertheless asymptotically unavoidable in comparison-based models (cf. the top-k lower bound in the hardness discussion).

Auxiliary memory bound. The only asymptotically nonconstant auxiliary state that must persist beyond a single query block is the output index array $I \in \mathbb{Z}^{n_q \times k}$ (or $n_q \times k_{\max}$ under padding). Thus the auxiliary space is $O(n_q k) = O(T)$ integers per head/layer, plus an optional $O(n_q k)$ edge list if trace records are stored rather than streamed. On-chip working storage is $O((b_q + b_k)d)$ for tiles and $O(k)$ for candidate scores/indices; neither depends on T. In particular, no $T \times T$ score, probability, or mask matrices are materialized, satisfying the space constraint.

HBM traffic upper bound. We bound global-memory traffic for mask estimation by accounting for the distinct streamed reads. Each query block loads its Q tile once, costing $\Theta(b_q d)$ elements, hence $\Theta(Td)$ overall. For each query block, each iteration scores $2k$ candidates, and each scored candidate requires loading exactly one representative K block of size $\Theta(b_k d)$ (amortizing intra-block reuse across threads computing the $b_q \times b_k$ dot products). Therefore,

$$\mathrm{IO}_{\mathrm{est}} \;\le\; O\Big(Td \;+\; n_q \cdot k \cdot n_{\mathrm{it}} \cdot b_k d\Big)$$

(up to datatype width and constant factors), matching Theorem 3. Any additional writes are limited to the output indices (and optional trace records), i.e. $O(n_q k)$ words. For mask application (if we proceed to compute masked attention summaries), the dominant additional reads are the K/V blocks referenced by $I[q,t]$; this contributes $O(n_q k b_k d)$ reads for each of K and V, and $O(Tkd)$ arithmetic for the sparse projection, consistent with the complexity summary.

HBM traffic lower bound and near-optimality. We now state what can be proved optimal under exact reference-equivalence. First, in the worst case any correct algorithm must read $\Omega(Td)$ elements of Q: otherwise there exists an unread entry of Q whose adversarial modification changes the correct top-k outputs for some query block, contradicting correctness. Second, emitting k retained key-block indices per query block implies an unconditional output write lower bound of $\Omega(n_q k)$ words. Third, for exact selection, the algorithm must access (hence read) at least one block of key data per emitted output in the worst case, giving $\Omega(n_q k b_k d)$ reads. Together these yield Theorem 4:

$$\mathrm{IO}_{\mathrm{est}} \;=\; \Omega\Big(Td \;+\; n_q k b_k d\Big),$$

up to datatype width. Comparing with the upper bound, our compiled STREAM estimator is within a factor of $n_{\mathrm{it}} = \lceil \log_2(T/b_k) \rceil$ of this information-theoretic minimum. We emphasize what is and is not claimed: we do not assert that the logarithmic factor is globally unavoidable for all conceivable exact sparse-masking schemes; rather, for the exact STREAM specification it is intrinsic because the control flow performs n_{it} refinement steps, and our implementation does not asymptotically increase the number of distinct K blocks touched beyond those implied by the specification. Thus, within the class of implementations that are bitwise reference-equivalent under τ and that avoid T^2 materialization, the dominant HBM traffic is near-minimal up to the refinement depth, while the remaining optimization space lies in constant factors (tiling, datatype, coalescing) and in trading exactness for approximation (which changes the specification and therefore falls outside the present optimality statement).

9 Experimental Plan

Our experimental goal is to substantiate three claims simultaneously: (i) bitwise reference-equivalence of the emitted index tensor I under the fixed tie-breaker τ; (ii) end-to-end latency and memory behavior consistent with the external-memory analysis (in particular, the absence of T^2 allocations and the dominance of HBM traffic); and (iii) practical usability at $T \in [10^5, 10^6]$ on a single consumer GPU, including representative downstream workloads.

Profiling methodology. We will profile at the granularity of the major pipeline stages: (a) mask estimation (producing I), (b) optional trace emission (streaming an edge list), and (c) optional mask application (sparse attention summaries or masked attention outputs). Latency will be measured with CUDA events around each kernel (warm-up excluded), reporting medians and tail percentiles over repeated runs with fixed seeds. Global-memory traffic will be measured using Nsight Compute and/or CUPTI counters (e.g. DRAM read/write bytes) to estimate $\mathrm{IO}_{\mathrm{est}}$ and separate reads of Q from streamed reads of representative K blocks. We will additionally record achieved bandwidth, occupancy, and tensor-core utilization to diagnose whether performance is bandwidth-limited as predicted for long contexts. To avoid confounding factors, we will pin (b_q, b_k, d, k) to a small set of compile-time-specialized kernels and report results per specialization.

Correctness checks against the reference specification. We will implement a reference STREAM procedure (CPU and GPU versions) that follows the same branch initialization, bisection schedule, representative selection rule ("first valid key block"), and top-k with deterministic tie-breaking τ. For small and medium T we will exhaustively compare the emitted indices I for every head and layer, requiring exact equality. For large T where full reference execution may be expensive, we will use two strategies: (i) randomized differential testing on many sampled query blocks q (checking that all $I[q, *]$ match), and (ii) metamorphic tests that preserve the control decisions (e.g. permuting invalid keys within masked-out regions of C_{blk}) to verify that determinism and validity invariants are respected. We will also validate the validity invariant directly by checking that every emitted key-block index satisfies $C_{\text{blk}}[q, I[q,t]] = 1$ and that empty outputs occur iff a query block has no valid keys under C_{blk}.

End-to-end latency and memory footprint. We will report (1) peak HBM allocation measured via CUDA memory queries and allocator logs, and (2) the asymptotic scaling of allocated bytes in T for indices-only and trace-emission configurations. The key requirement is empirical confirmation that auxiliary memory scales as $O(n_q k)$ rather than $O(T^2)$. For latency, we will present throughput as tokens/s and query-blocks/s, and we will separate time spent in mask estimation from time spent in any subsequent masked computation.

Scaling to $T = 10^6$. We will conduct strong-scaling sweeps over T while holding (b_q, b_k, d, k) fixed, and record (i) kernel time, (ii) measured HBM bytes, and (iii) the number of representative key blocks touched per query block. We will test both benign masks (pure causal) and adversarial masks (sparse validity patterns in C_{blk}) to ensure that the representative selection and branch refinement do not induce pathological divergence. We will compare observed scaling against the predicted $O(T \log(T/b_k))$ work trend and the IO upper bound, with attention to whether the effective multiplicative factor tracks $n_{\text{it}} = \lceil \log_2(T/b_k) \rceil$.

Benchmark tasks and practical stability. To demonstrate utility beyond microbenchmarks, we will evaluate on long-context suites such as RULER and on long chain-of-thought style prompting where attention tracing is diagnostically valuable. For each task, we will report (a) overhead relative to the same model without tracing, (b) stability of the emitted sparse traces across prompts and seeds (indices I are deterministic for fixed inputs), and (c) qualitative sanity checks of trace artifacts (e.g. concentration on relevant context regions). Finally, we will repeat the full measurement suite across several GPUs spanning architectures and memory bandwidth tiers (e.g. RTX-class consumer GPUs and workstation variants), and report both absolute performance and normalized bandwidth efficiency to assess portability and sensitivity to hardware constraints. These experiments will delineate the regimes where STREAM-style tracing is reliable and cost-effective, and thereby motivate the limitations and extensions discussed next.

10 Limitations and Future Work

We delimit the present contribution to a compilation-friendly realization of a particular sparse tracing primitive: STREAM-style hierarchical selection of key blocks followed by optional masked computation and emission. This focus yields strong reference-equivalence guarantees under a fixed tie-breaking rule τ and enables an external-memory analysis, but it also restricts generality in several directions.

Generality beyond attention-only tracing. Our current interface assumes that the objects of interest are attention edges (query-block to key-block relations) induced by (Q, K) under a causal and optional task mask C, and that the downstream consumer is satisfied with indices I (optionally augmented with scores/weights). Many interpretability questions, however, concern phenomena not localized to attention alone (e.g. MLP feature activations, residual stream attribution, or cross-layer circuit motifs). A principled extension would treat STREAM as a generic sparse-selection subroutine over any bilinear (or more general) interaction that can be evaluated blockwise without T^2 materialization. The obstruction is that the scoring functional used here, $\max_{m,n}\langle Q_m, K_n\rangle$, is aligned with attention geometry and admits efficient tiling; alternative functionals (e.g. integrated gradients along the residual path) may not admit similarly cheap representatives. We view it as future work to identify a small family of score functionals that (i) are interpretable, (ii) preserve a determinism invariant analogous to τ, and (iii) remain IO-efficient under a streaming model.

Multi-GPU and distributed execution. The present analysis is single-device: Q and K reside in one HBM domain and the kernel streams representative K blocks to on-chip memory. For $T \approx 10^6$ this is plausible on high-memory consumer devices, but not universal, and it excludes settings where Q, K, V are already sharded (tensor parallelism) or where context is pipeline-parallel across devices. Extending STREAM to a multi-GPU regime introduces two coupled difficulties: (a) branch refinement depends on scores that may require reading remote K shards, and (b) deterministic top-k with tie-breaking τ must be maintained across asynchronous collectives. A natural direction is a two-level scheme in which each device proposes local candidates with local τ-consistent top-k, followed by a globally deterministic merge (e.g. via all-gather of $(\text{score}, \text{index})$ pairs and a total-order reduction). The resulting communication cost must be analyzed alongside HBM-IO; we expect regimes where interconnect bandwidth, rather than HBM bandwidth, becomes the bottleneck.

Certified fidelity and robustness. We claim exact reference-equivalence with respect to a specified STREAM procedure, but this is not a certificate of faithfulness to dense attention or to any semantic notion of "ground truth" importance. In particular, the representative selection rule (first valid key block in a branch) and the blockwise max score are design choices; they are stable under τ yet may be brittle under distribution shift or adversarial masking patterns in C_{blk}. One line of work is to develop post hoc certificates stating that omitted key blocks cannot change the top-k decision beyond a known margin, perhaps using upper bounds from blockwise norms (e.g. $\|Q\|_2\|K\|_2$ bounds) computed cheaply. Another is to formalize and test robustness properties: Lipschitz-type guarantees of I under small perturbations of Q, K, and sensitivity analyses with respect to the mask C.

Variable sparsity schedules and adaptivity. We have treated k as fixed (or as a known schedule $k(q)$). In practice, the effective sparsity required for faithful tracing may vary by layer, head, and query block; allocating a uniform k may waste IO on easy regions while under-allocating on hard regions. Adaptive schedules based on score gaps across candidates, entropy proxies, or branch-confidence measures are appealing, but they interact subtly with compilation (static shapes), determinism (the control flow must remain τ-deterministic), and IO predictability (worst-case reads may increase). We anticipate a constrained adaptivity model: select k from a small discrete set per query block with a deterministic rule derived from intermediate scores, enabling kernel specialization while retaining bounded divergence.

Integration with interpretability toolchains. Finally, utility depends on how traces are consumed. Existing toolchains often assume dense attention maps or per-token saliency arrays; our outputs are sparse, block-indexed, and naturally streamed. Bridging this gap requires standardized trace formats (edge lists with metadata, compression, and indexing), efficient CPU-side consumers, and alignment primitives to map block indices back to token spans under padding and masking. We also require careful engineering to ensure that trace emission does not dominate runtime (e.g. via buffered streaming, quantized scores, or on-GPU aggregation). A systematic integration plan, including APIs and reproducible visualization pipelines, remains an essential component of future work.

11 Appendices: reference spec, tie-breaking rules, kernel pseudocode details, reproducibility checklists.

A Reference Specification

We provide, for completeness and for downstream verification, a reference STREAM specification against which all compiled kernels are compared. The specification is stated at the level of query blocks and key blocks, assuming padding so that T is divisible by $\operatorname{lcm}(b_q, b_k)$. For each query block index $q \in \{1, \ldots, T/b_q\}$ and each layer/head pair, the reference procedure initializes a set of k disjoint branches whose union covers the admissible key-block interval implied by causality and the user mask. Each refinement iteration splits every retained branch into two sub-branches, yielding $2k$ candidates, and assigns to each candidate a *representative* key block r chosen deterministically as the smallest key-block index in that candidate interval satisfying $C_{\mathrm{blk}}[q, r] = 1$ (or declares the candidate invalid if none exists). The candidate score is then

$$\mathrm{score}(q, r) \;=\; \max_{\substack{0 \le m < b_q,\, 0 \le n < b_k \\ C[qb_q + m,\, rb_k + n] = 1}} \langle Q[qb_q + m, :],\, K[rb_k + n, :] \rangle \,,$$

with the convention that $\max \emptyset = -\infty$. The reference outputs $I[q, 1..k]$ as the representative indices associated to the final k branches after $n_{\mathrm{it}} = \lceil \log_2(T/b_k) \rceil$ iterations (or an empty entry if no valid key exists). This is the *only* correctness target for the compiled implementation; all reported equivalences are with respect to this specification rather than dense attention.

B Tie-Breaking and Total Order τ

Exact reference-equivalence requires that all parallel reductions implement a fixed total order τ on candidates. We therefore define τ as a lexicographic order on tuples

$$\big(\mathrm{score},\, r,\, \mathrm{branch\_id},\, \mathrm{split\_bit}\big),$$

ordered by decreasing score and then increasing $(r, \mathrm{branch\_id}, \mathrm{split\_bit})$. In particular, equal scores must resolve to smaller representative indices, and any remaining ambiguity is discharged by the stable branch identifiers induced by the initialization and split schedule. We further fix numerical corner cases: NaN scores are treated as $-\infty$, and $-\infty$ compares below all finite values. These conventions ensure that top-k selection is deterministic across warp-level and block-level reductions, independent of execution interleavings.

C Kernel Pseudocode Details

We include pseudocode for the compiled kernel at the granularity required to reproduce control flow and memory traffic accounting. The implementation is organized as a persistent kernel over query blocks, with one thread block responsible for one (or a small constant number of) query blocks. Each block loads Q for its query block once into SMEM (or registers when $b_q d$ permits), and then streams representative K blocks into a double-buffered SMEM tile to overlap memory latency with compute. The score functional is implemented as a fused tiled reduction: tensor-core dot products accumulate partial inner products for (m, n) pairs, followed by a max-reduction over the $b_q \times b_k$ grid subject to C. The top-k over $2k$ candidates uses a fixed comparator implementing τ and a deterministic sequence of warp shuffles; no atomics participate in the ordering.

D Reproducibility Checklist

To make the determinism and IO claims falsifiable, we provide a minimal checklist. We (i) record $(T, d, L, H, b_q, b_k, k)$, padding policy, and the exact definition of C and C_{blk}; (ii) specify datatype and accumulation mode (e.g. bf16 inputs, fp32 accumulation), along with NaN handling; (iii) fix compilation flags affecting associativity or contraction (e.g. fused multiply-add) and disable nondeterministic reductions; (iv) report GPU model, driver, CUDA toolkit, and clock settings; (v) log the emitted indices I and, when enabled, a checksum of streamed edge lists; and (vi) include a reference CPU implementation that replays the same τ order to validate bitwise identity of I on representative workloads.