ELSA: Exact Linear-Scan Attention for Fast and Memory-Light Vision Transformers

Supplementary Material

Supplementary Overview

This supplementary provides theoretical foundations, additional experimental results, and implementation details for ELSA, organized as follows.

A. Monoid Proof for $\oplus$

ELSA's parallelization strategy rests on casting online softmax attention as a parallel prefix scan over an associative operator (Sec. **??** of the main paper). This section provides the complete proof that our merge operator $\oplus$ forms a monoid, thereby guaranteeing that **any ordering of pairwise merges yields the same result**—the property that makes Hillis–Steele and Blelloch scans provably correct for ELSA.

A.1. State Space and Operations

Let $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$ denote the extended reals, and let b, h, d, d_v denote batch size, number of heads, key/query dimension, and value dimension respectively. Define the state space

$$G = \bar{\mathbb{R}} \times \mathbb{R}_{\geq 0} \times \mathbb{R}^{d_v}, \tag{1}$$

whose elements are triples (m, S, W): the running maximum logit m, the normalized cumulative sum $S \in \mathbb{R}_{\geq 0}$, and the weighted value accumulator $W \in \mathbb{R}^{d_v}$. For $a = (m_a, S_a, W_a)$ and $b = (m_b, S_b, W_b)$ in G, we define two operations that underpin $\oplus$ (Eq. (**??**) of the main paper).

Unnormalize:

$$\mathrm{un}(m, S, W) \;=\; (m,\; Se^{m},\; We^{m}) \tag{2}$$

exposes the raw accumulated sums prior to numerical stabilization, reversing the e^{-m} re-anchoring applied during the scan.

Renormalize:

$$\mathrm{renorm}(m, \tilde{S}, \tilde{W}) \;=\; (m,\; \tilde{S}e^{-m},\; \tilde{W}e^{-m}) \tag{3}$$

restores numerical stability by rescaling with the running maximum, ensuring all exponents remain ≤ 0 (Lemma 1 of the main paper).

The merge operator $\oplus$ composes two block states via **un-sum-renorm**:

$$\begin{aligned} a \oplus b \;&=\; \mathrm{renorm}\Big(m_c,\; \tilde{S}_a + \tilde{S}_b,\; \tilde{W}_a + \tilde{W}_b\Big), \\ m_c \;&=\; \max(m_a, m_b), \end{aligned} \tag{4}$$

where $(m_a, \tilde{S}_a, \tilde{W}_a) = \mathrm{un}(a)$ and $(m_b, \tilde{S}_b, \tilde{W}_b) = \mathrm{un}(b)$.

A.2. Monoid Structure

Identity element. Let $\boldsymbol{e} = (-\infty, 0, \mathbf{0})$. Since $\mathrm{un}(\boldsymbol{e}) = (-\infty, 0, \mathbf{0})$, by Eq. (4), $\boldsymbol{e}$ contributes zero to both $\tilde{S}$ and $\tilde{W}$, and $m = -\infty$ never dominates any finite m under max. Hence $\boldsymbol{e}$ is a **two-sided identity**:

$$a \oplus \boldsymbol{e} \;=\; \boldsymbol{e} \oplus a \;=\; a \qquad \forall\, a \in G, \tag{5}$$

which enables Blelloch's down-sweep to initialize boundary states correctly without special-casing empty prefixes.

Associativity. Let $u_a, u_b, u_c \in G$ with $u_i = (m_i, S_i, W_i)$ for $i \in \{a, b, c\}$, and let $\tilde{S}_i = S_i e^{m_i}$, $\tilde{W}_i = W_i e^{m_i}$ denote their unnormalized sums. For $u_{ab} = u_a \oplus u_b$, Eq. (4) gives $m_{ab} = \max(m_a, m_b)$ and $\tilde{S}_{ab} = \tilde{S}_a + \tilde{S}_b$, $\tilde{W}_{ab} = \tilde{W}_a + \tilde{W}_b$. Merging with u_c and writing $m_* = \max(m_a, m_b, m_c)$:

$$S_{(ab)c} = (\tilde{S}_a + \tilde{S}_b + \tilde{S}_c)e^{-m_*}, \quad W_{(ab)c} = (\tilde{W}_a + \tilde{W}_b + \tilde{W}_c)e^{-m_*}. \tag{6}$$

The right-associative grouping $u_a \oplus (u_b \oplus u_c)$ yields the same m_* and the same unnormalized totals, so

$$S_{a(bc)} = (\tilde{S}_a + \tilde{S}_b + \tilde{S}_c)e^{-m_*}, \quad W_{a(bc)} = (\tilde{W}_a + \tilde{W}_b + \tilde{W}_c)e^{-m_*}. \tag{7}$$

Since Eqs. (6) and (7) are identical,

$$(u_a \oplus u_b) \oplus u_c \;=\; u_a \oplus (u_b \oplus u_c). \tag{8}$$

Together with Eq. (5), $(G, \oplus, \boldsymbol{e})$ is a **monoid**, guaranteeing the correctness of the two-level parallel prefix scan in Sec. **??** and the FP32 error bound in Theorem 1 of the main paper. □

A.3. Derivation of the Floating-Point Error Bound

We derive the bound stated in Theorem 1 of the main paper. Throughout, we adopt the standard IEEE-754 model $\mathrm{fl}(x \,\mathrm{op}\, y) = (x \,\mathrm{op}\, y)(1+\delta)$ with $|\delta| \le u$ (unit roundoff), and rely on Lemma 1 (bounded exponents) and Assumption 1 (exp-accuracy) from the main paper.

Per-merge error. Consider a single pairwise merge $S_{\text{out}} = S_a e^{m_a - m_{\text{out}}} + S_b e^{m_b - m_{\text{out}}}$. Rounding errors arise from (1) the exponential scaling factors and (2) the subsequent addition. By Lemma 1, all scaling exponents satisfy $m_a - m_{\text{out}} \le 0$, so $e^{m_a - m_{\text{out}}} \in (0, 1]$—bounding the mantissa and preventing error amplification. Letting ϵ_k denote the relative error at depth k of the reduction tree, the recursive step satisfies

$$\hat{S}_k \;\approx\; S_{\text{true}}\,(1+\epsilon_{k-1})(1+\delta_{\text{exp}})(1+\delta_{\text{add}}). \tag{9}$$

Dropping second-order terms $\mathcal{O}(u^2)$, the error grows **linearly in depth**:

$$|\epsilon_k| \;\le\; |\epsilon_{k-1}| + c\,u, \tag{10}$$

where c counts the FLOPs per merge (one subtraction, one exponential, one multiplication, one addition).

Total depth and error bound. The two-level scan (Sec. **??** of the main paper) contributes

$$L(n,B) \;=\; \underbrace{\lceil \log_2 B \rceil}_{\text{intra-block (Hillis–Steele)}} \;+\; \underbrace{2\lceil \log_2 (n/B) \rceil}_{\text{inter-block (Blelloch up/down)}} \;+\; \text{const} \tag{11}$$

matching Eq. (**??**) of the main paper. Telescoping Eq. (10) over $L(n,B)$ levels gives

$$\frac{\|\hat{y} - y\|_2}{\|y\|_2} \;\le\; \gamma_{L(n,B)} \;\approx\; L(n,B)\cdot u, \tag{12}$$

where $\gamma_n = nu/(1-nu) \approx nu$ (Higham [3]). Since $L(n,B) = \mathcal{O}(\log n)$, the error grows logarithmically with sequence length—in contrast to naive summation ($\mathcal{O}(\sqrt{n}\,u)$) or sequential recurrence ($\mathcal{O}(n\,u)$)—confirming the numerical stability of ELSA as $n \to \infty$ and establishing Theorem 1.

A.4. Algorithm

This subsection provides the full pseudocode and a step-by-step walk-through of the per-query ELSA scan, which was condensed in the main paper (Sec. **??**) due to space constraints. Algorithm 1 writes the scan in its sequential single-query form for clarity; the parallel two-level realization on GPU follows directly from the monoid structure of $\oplus$ and is described in Sec. H.

State triple and invariants. The algorithm maintains a triple (m_j, S_j, W_j) summarizing the prefix $1{:}j$ of the key/value stream, where m_j is the running maximum logit, $S_j = \sum_{i \le j} \exp(s_i - m_j)$ is the rescaled normalizer, and $W_j = \sum_{i \le j} \exp(s_i - m_j)\, V[i]$ is the rescaled weighted value sum. These three quantities together encode the exact softmax-weighted output for the prefix as W_j/S_j, while remaining numerically stable because all exponentials are evaluated relative to the current maximum m_j. The identity element $\boldsymbol{e} = (-\infty, 0, \mathbf{0})$ initializes the recursion (Line 3) so that the first iteration encounters $m_0 = -\infty$ and immediately enters the "new max" branch.

Per-token update. For each new token j (Lines 4–16), the algorithm first computes the scaled inner product $s_j = \langle q, K[j] \rangle / \sqrt{d}$ (Line 5). It then dispatches on whether the previous running maximum still dominates ($m_{j-1} \ge s_j$, Lines 6–10) or the new logit takes over ($m_{j-1} < s_j$, Lines 11–15):

- **Case A: maximum unchanged.** The previous m_{j-1} is preserved as m_j (Line 7). The new contribution is folded in by computing $\alpha = \exp(s_j - m_j)$ (Line 8) and adding it to both the normalizer S and the weighted value sum W (Lines 9–10). No rescaling of past terms is required because the normalization anchor m_j has not moved.
- **Case B: new maximum.** The current logit s_j becomes the new anchor (Line 12). All previously accumulated quantities are rescaled by $\beta = \exp(m_{j-1} - m_j) \le 1$ (Line 13) so that they are expressed relative to the new maximum, and the contribution of token j itself is added with weight $\exp(0) = 1$ (Lines 14–15).

This conditional rescaling is precisely the online-softmax update of Eq. (**??**): it guarantees that, after every step, (m_j, S_j, W_j) encodes the exact softmax of the prefix without ever materializing the $n \times n$ score matrix and without floating-point overflow, since all exponentials use non-positive arguments.

Final normalization. After processing all n tokens, the attention output is recovered by a single division $y = W_n/S_n$ (Line 17). Because W_n and S_n are expressed in the same exponential frame anchored at m_n, the $\exp(\cdot)$ scale cancels in the ratio, yielding the exact softmax output up

Algorithm 1 ELSA per-query sequential formulation. The loop is written sequentially for clarity; parallel execution follows from the monoid structure of $\oplus$ (Sec. **??**).

Require: Query $q \in \mathbb{R}^d$, Keys $K[1..n] \in (\mathbb{R}^d)^n$, Values $V[1..n] \in (\mathbb{R}^{d_v})^n$
Ensure: Attention output $y \in \mathbb{R}^{d_v}$

$(m_0, S_0, W_0) \leftarrow (-\infty, ; 0, ; \mathbf{0})$ ▷ initialize state triple: max logit, normalized sum, weighted sum
for $j = 1$ to n **do**
  $s_j \leftarrow \frac{\langle q, K[j] \rangle}{\sqrt{d}}$ ▷ compute attention score (logit) for token j
  if $m_{j-1} \geq s_j$ **then** ▷ current max remains; rescale new contribution
    $m_j \leftarrow m_{j-1}$ ▷ keep previous maximum
    $\alpha \leftarrow \exp(s_j - m_j)$ ▷ normalized weight for token j
    $S_j \leftarrow S_{j-1} + \alpha$ ▷ accumulate normalized sum
    $W_j \leftarrow W_{j-1} + \alpha, V[j]$ ▷ accumulate weighted value sum
  else ▷ new max found; rescale all previous contributions
    $m_j \leftarrow s_j$ ▷ update maximum to current score
    $\beta \leftarrow \exp(m_{j-1} - m_j)$ ▷ rescaling factor for previous terms
    $S_j \leftarrow S_{j-1} \cdot \beta + 1$ ▷ rescale previous sum and add $\exp(0) = 1$
    $W_j \leftarrow W_{j-1} \cdot \beta + V[j]$ ▷ rescale previous weighted sum and add current value
  end if
end for
$y \leftarrow W_n / S_n$ ▷ normalize: divide accumulated weighted sum by total weight

to floating-point round-off bounded by $u\, L(n, B)$ (Theorem **??**).

From sequential to parallel. Although Algorithm 1 is presented as a left-to-right loop for readability, its correctness depends only on the associativity of $\oplus$ (Proposition **??**), not on the order of combination. Concretely, any binary tree of $\oplus$-merges over the state triples $u_1, \ldots, u_n$ produces the identical final state $u_1 \oplus \cdots \oplus u_n$. This is exactly the property exploited by the GPU implementation: intra-block reductions use a Hillis–Steele tree of depth $\lceil \log_2 B \rceil$ in shared memory, and inter-block combinations use a Blelloch up-/down-sweep of depth $2\lceil \log_2(n/B) \rceil$ in global memory, yielding a total scan depth $L(n, B)$ (Sec. H). The pseudocode below should therefore be read as a *specification* of $\oplus$ applied left-to-right; any parallel evaluation order matching the same monoid contract produces bit-equivalent results in exact arithmetic and bounded round-off in floating point.

B. I/O Analysis and Bounds

We analyze ELSA's I/O complexity at two levels, complementing the I/O scope paragraph in Sec. **??** of the main paper. Throughout, *linear-scan* refers to $\mathcal{O}(n)$ extra memory and one-pass I/O per query; total arithmetic work remains $\mathcal{O}(n^2)$.

B.1. Per-Query Streaming

Any exact attention computation for a fixed query must read its K, V inputs and write its output at least once, incurring $\Omega(n)$ I/O. ELSA **attains this lower bound** by streaming K and V exactly once per query while keeping the running state (m, S, W) in registers or fast memory—a direct consequence of the monoid structure: the identity $\boldsymbol{e} = (-\infty, 0, \mathbf{0})$ allows the scan to initialize without reading any additional data.

B.2. All-Queries Tiling

For the full $QK^\top$ interaction and subsequent PV accumulation, classical red–blue pebble arguments imply the I/O lower bound

$$\Omega\left(\frac{n^2(d + d_v)}{\sqrt{M_f}}\right) \tag{13}$$

for fast-memory capacity M_f. ELSA adopts a blocked schedule with query tile T_q and key/value tile T_k subject to

$$T_q d + T_k(d + d_v) \;\leq\; c\, M_f \tag{14}$$

for a small constant c. The resulting total DRAM traffic is

$$\Theta(nd + nd_v) \;+\; \Theta\left(\frac{n^2(d + d_v)}{T_q}\right). \tag{15}$$

Setting $T_q \asymp \sqrt{M_f/(d + d_v)}$ reduces this to

$$\mathcal{O}\left(\frac{n^2(d + d_v)}{\sqrt{M_f}}\right) + \Theta(nd + nd_v), \tag{16}$$

matching the lower bound (13) up to constants (Eq. (**??**) of the main paper). Hence *linear-scan* refers strictly to per-query memory ($\mathcal{O}(n)$); the global all-queries schedule meets the rectangular-matrix I/O lower bound within constant factors via tiling.

C. Numerical Equivalence Verification

Standard softmax attention materializes the full $n \times n$ score matrix and normalizes in a single pass—a numerically well-characterized reference implementation. Milakov and Gimelshein [6] showed that the same result can be obtained by an *online* procedure that maintains running statistics $(m_i, s_i, \mathbf{w}_i)$ and updates them token by token, without ever storing the full matrix. ELSA lifts this sequential scan to a parallel prefix computation by casting those statistics as elements of an associative monoid (Sec. **??**); because the merge operator $\oplus$ is associative, any evaluation order—sequential or parallel—yields identical results in exact arithmetic. To validate this claim empirically, we use the vectorized batch formulation as a verification oracle: it computes $(\hat{P}, \hat{Y})$ via standard matrix operations and we measure the drift against the monoid scan on GPU. In floating point the two agree to within $\mathcal{O}(u \log n)$ (Sec. A.3), with both sharing the same asymptotic cost—$\mathcal{O}(bhnd)$ for the query–key product and $\mathcal{O}(bhnd_v)$ for value accumulation.

Algorithm 2 Vectorized equivalence check

Require: $Q, K, V \in \mathbb{R}^{b\times h\times n\times d}$; reference outputs P_{ref}, Y_{ref} (high precision)
Ensure: Drift metrics $\{\|\delta P\|_\infty, \|\delta Y\|_\infty, \ldots\}$
$\mathbf{S} \leftarrow (QK^\top)/\sqrt{d}$
$m \leftarrow \max(\mathbf{S},\ \text{dim}=-1,\ \text{keepdim=True})$
$E \leftarrow \exp(\mathbf{S} - m)$
$s \leftarrow \sum(E,\ \text{dim}=-1,\ \text{keepdim=True})$
$\hat{P} \leftarrow E \oslash s$; $\hat{Y} \leftarrow (EV) \oslash s$
return $\|\hat{P} - P_{\text{ref}}\|_\infty,\ \|\hat{Y} - Y_{\text{ref}}\|_\infty$

C.1. Vectorized Reference Algorithm

Let $Q, K \in \mathbb{R}^{b\times h\times n\times d}$ and $V \in \mathbb{R}^{b\times h\times n\times d_v}$ denote queries, keys, and values with batch size b, h heads, and sequence length n. The reference procedure computes attention probabilities $\hat{P}$ and output $\hat{Y}$ as

$$\begin{aligned} \mathbf{S} &= \tfrac{1}{\sqrt{d}} QK^\top, & m_i &= \max_j \mathbf{S}_{:,:,i,j}, \\ E &= \exp(\mathbf{S} - m), & s_i &= \textstyle\sum_j E_{:,:,i,j}, \\ \hat{P} &= E \oslash s, & \hat{Y} &= (EV) \oslash s, \end{aligned} \tag{17}$$

where $\oslash$ denotes row-wise division. This mirrors the accumulation rules of Algorithm 1: row-wise max subtraction for stability, exponentiation, sum accumulation, and normalization. Numerical drift relative to the scan is reported in Sec. C.2.

C.2. Evaluation Results and Analysis

Metrics and Setup. We define six complementary drift metrics on $\delta P = \hat{P} - P_{\text{ref}}$ and $\delta Y = \hat{Y} - Y_{\text{ref}}$, chosen to probe different failure modes of a numerical reformulation:

1. **Maximum absolute probability drift** $\|\delta P\|_\infty = \max_{b,h,i,j} |\delta P_{b,h,i,j}|$: worst-case pointwise error in the attention matrix—detects any single catastrophically mis-computed weight.
2. **Relative L_2 probability error** $\|\delta P\|_2 / \|P_{\text{ref}}\|_2$: aggregate error normalized by scale—captures global distributional shift that pointwise maxima may mask by averaging over all entries.
3. **Mean Jensen–Shannon divergence** $\text{JS}_{\text{mean}} = \frac{1}{bhn} \sum_{b,h,i} \text{JS}(\hat{P}_{b,h,i,:} \,\|\, P_{\text{ref},b,h,i,:})$: per-row distributional discrepancy—JSD is symmetric and bounded, so it measures whether the *shape* of each query's attention distribution is preserved, beyond individual entry magnitudes.
4. **Argmax disagreement rate** $\text{argRate} = \frac{1}{bhn} \sum_{b,h,i} \mathbf{1}[\arg\max_j \hat{P}_{b,h,i,j} \neq \arg\max_j P_{\text{ref},b,h,i,j}]$: fraction of query positions where the top-attended token changes— the most behaviorally critical metric, as argmax disagreement directly alters which context the model attends to.
5. **Maximum absolute output drift** $\|\delta Y\|_\infty = \max_{b,h,i,k} |\delta Y_{b,h,i,k}|$: worst-case error in the final output embeddings propagated to the next layer—the downstream analogue of metric 1.
6. **Relative L_2 output error** $\|\delta Y\|_2 / \|Y_{\text{ref}}\|_2$: normalized aggregate output error—confirms that overall output magnitude is faithfully preserved across the full tensor.

Metrics 1–4 probe the attention probability matrix P, progressing from pointwise to distributional to functional correctness; metrics 5–6 repeat the same hierarchy for the output Y. All are computed in a fully vectorized manner on GPU; we report the median, 95th, and 99th percentiles across all tokens and heads.

Results. Table 1 reports 95th-percentile drift between the monoid scan (FP32) and a float64 oracle reference across all scenarios. **Discrepancies reach at most 3.3×10^{-16} for probabilities and 3.3×10^{-15} for outputs**—well below single-precision unit roundoff $u \approx 6 \times 10^{-8}$, confirming that the scan result is indistinguishable from the float64 reference at FP32 precision. Crucially, the argmax disagreement rate is identically zero in all scenarios, confirming that the monoid scan **exactly preserves the top attention weight**. Long-sequence and stress scenarios exhibit slightly larger absolute errors owing to accumulated partial sums, yet all values remain at float64-oracle-level drift—as predicted by the logarithmic depth $L(n, B)$ (Eq. (11)).

Figure 1 further corroborates these findings: the heatmap (left) reveals no spatial structure in $|\delta P|$ across token pairs—the uniformly dark field confirms that errors are random floating-point noise rather than systematic accumulation—while the log-scale histogram (right) shows $|\delta Y|$ heavily concentrated near zero even under the stress scenario, with its tail bounded well below 2×10^{-14}, orders of magnitude beneath single-precision unit roundoff. Finally, Figure 2 directly validates monoid composition (Proposition 1) by partitioning tokens into blocks of $B = 128$, computing (m, S, W) per block in parallel, and reducing via $\oplus$; the resulting per-block drift is indistinguishable from the float64 oracle result, confirming correctness of the associative merge at the core of ELSA.

D. Ablation Studies

We examine the impact of block size (B), accumulator precision, and resource-constrained deployment (Fig. 3). Block size B=128 offers the best latency-occupancy tradeoff across all configurations. Using FP16 for the (S, W) accumulator causes numerical error to increase dramatically beyond 2K tokens, while FP32 remains stable throughout (Fig. 3, bottom). On Jetson TX2, baseline MHSA kernels and FA2 cannot process long sequences due to memory limitations, whereas ELSA maintains efficiency across all to-

Metric (p95)	Regular	Long	Stress
$\|\delta P\|_\infty$	3.12×10^{-17}	2.34×10^{-17}	3.33×10^{-16}
$\|\delta P\|_2 / \|P_{\text{ref}}\|_2$	1.73×10^{-15}	3.42×10^{-15}	3.50×10^{-15}
JS_{mean}	3.56×10^{-16}	3.77×10^{-16}	3.07×10^{-16}
argRate	0	0	0
$\|\delta Y\|_\infty$	4.99×10^{-16}	4.99×10^{-16}	3.28×10^{-15}
$\|\delta Y\|_2 / \|Y_{\text{ref}}\|_2$	2.39×10^{-15}	4.72×10^{-15}	4.94×10^{-15}

Table 1. **Vectorized equivalence (oracle-based).** 95th-percentile drift metrics against a high-precision (float64) reference, across all batches, heads, and tokens. All three scenarios show near-machine-precision agreement.

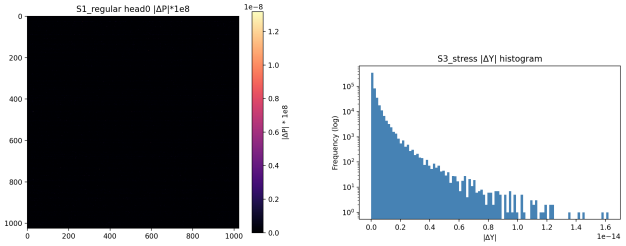

Figure 1. **Drift visualization.** Left: heatmap of $|\delta P|$ (scaled by 10^8) for head 0, regular scenario; the uniformly dark field indicates no systematic spatial accumulation, errors are random floating-point noise throughout the $n \times n$ attention matrix. Right: log-scale histogram of $|\delta Y|$ under the stress scenario; the distribution is heavily concentrated near zero, with the tail bounded below 2×10^{-14}, confirming machine-precision agreement in the worst case.

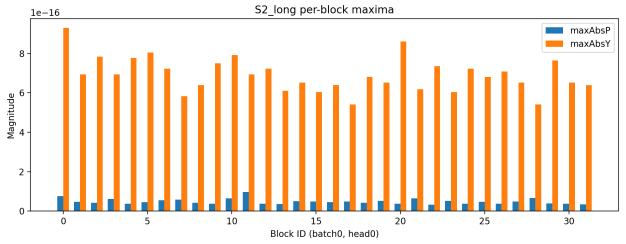

Figure 2. **Blockwise monoid validation.** Tokens are partitioned into blocks of 128; (m, S, W) is computed per block in parallel and then reduced via $\oplus$. The resulting drift metrics are indistinguishable from the global vectorized procedure, validating correctness of the associative merge.

ken lengths (Fig. 3, top).

E. Variable-Length Sequence Benchmark

To ensure that performance differences are not confounded by data-layout conversions, we benchmark variable-length

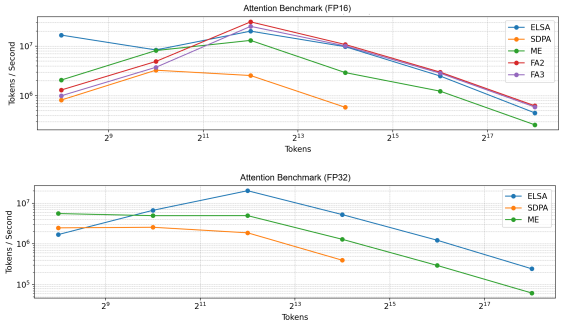

Figure 3. **Memory footprint on synthetic benchmarks.** ELSA's advantage widens with sequence length in both precision modes.

Method	Total (ms)	Kernel (ms)	Convert (ms)	Mem (GB)	Tput (M)
ELSA	1.79	1.12	**0.66**	**0.05**	5.66
FA2	**1.65**	**0.31**	1.34	0.10	**6.14**

Table 2. **Variable-length sequence benchmark.** Random sequences sampled from $[256, 1792]$ (FP32, medians over 200 runs). Total: end-to-end latency; Kernel: attention kernel time; Convert: layout conversion overhead; Mem: peak GPU memory; Tput: throughput in millions of tokens/s. ELSA's zero-copy design reduces conversion overhead at the cost of longer kernel time.

sequences. We generate 8 sequences with lengths uniformly sampled from $[256, 1792]$ and compare ELSA (direct processing) against FA2 in unpadded mode (`cu_seqlens`), accounting for FA2's conversion cost between padded and packed layouts.

Table 2 reports end-to-end latency, kernel time, conversion overhead, peak memory, and throughput (medians over 200 runs). ELSA's zero-copy pipeline reduces FA2's padded-to-packed conversion overhead (from 1.34 ms to 0.66 ms), achieving comparable end-to-end latency despite a longer kernel time.

F. Window Attention and Offloading Analysis

F.1. Block Size Sensitivity

Window-ELSA introduces a block size B controlling the work assigned to each prefix-scan step. We swept $B \in \{64, 128, 256\}$ on Swin-T and Swin-S (A100, FP32) to characterize the throughput-memory trade-off. Smaller blocks reduce buffer pressure at the cost of more scan steps; larger blocks amortize scan overhead but may spill to slower memory. Concretely, B=64 reduces throughput by $\sim$15% while cutting memory by $\sim$20%; B=256 yields negligible speedup but increases memory by $\sim$20%. Relaxing synchronization to every two blocks saves $\sim$5% runtime at a cost of $\sim$8% extra memory. We therefore adopt B=128 with per-block synchronization as the default operating point; memory-constrained platforms may prefer

B=64.

F.2. Offloading Performance

Methodology. We quantify PCIe–compute overlap using LLaMA-13B instantiated as a meta-graph with no materialized weights; layer weights are loaded one layer at a time during the forward pass via asynchronous double-buffering (layer $n{+}1$ transfers while layer n computes). Each layer is instrumented with three CUDA event pairs measuring (i) H2D transfer time (`copy_ms`), (ii) kernel computation time (`compute_ms`), and (iii) idle time (`wait_ms`). We summarize overlap via

$$\text{Overlap}_{\text{eff}} = 1 - \frac{\texttt{wait\_ms}}{\texttt{total\_ms}}, \tag{18}$$

which approaches 1.0 when PCIe transfers are fully hidden by computation.

Results. Table 5 reports results for LLaMA-13B (FP32, batch 1) across four sequence lengths. At 4K and 8K tokens the workload is PCIe-bound: transfers complete faster than ELSA's scan overhead, yielding higher idle time (0.727 ms and 1.005 ms vs. 0.228 ms and 0.322 ms for SDPA) and lower throughput (-42.4% and -25.2% respectively). At 32K tokens, ELSA's lower memory footprint allows computation to begin earlier, reducing idle time from 540 ms to 467 ms/layer and delivering a **17.8%** throughput gain (1.460 vs. 1.239 M tok/s). At 65K tokens ELSA sustains **20.2%** higher throughput (0.882 vs. 0.734 M tok/s), confirming that memory efficiency compounds into end-to-end gains as sequences grow.

G. On 3D Large Reconstruction Model

To validate ELSA's effectiveness on 3D vision tasks, we evaluate it within the Visual Geometry-Grounded Transformer (VGGT) [11], a feed-forward Large Reconstruction Model (LRM) for multi-view 3D reconstruction. VGGT alternates between frame-wise and global attention layers; the global attention aggregates cross-view information but becomes the computational bottleneck as the number of frames grows.

Experimental Setup. We compare ELSA against FA2-FP16, xFormers-FP32 (ME-SDPA), and PyTorch Math under two configurations: (1) **VGGT Baseline** on 50–150 frames, and (2) **FastVGGT** [9], a memory-optimized variant with token merging. All experiments use 350×518 input resolution, yielding 1,041 tokens per frame (28×37 patches plus camera and register tokens).

Results on VGGT Baseline. Table 3 shows that ELSA-FP32 achieves a **1.46× speedup** over xFormers at 50 frames (12.56 s vs. 18.36 s) while matching memory usage (15.74 GB, within 0.01 GB). The advantage compounds with sequence length, reaching **2.34**× at 150 frames (55.97 s vs. 130.83 s) with an identical 37.59 GB footprint. FA2-FP16 remains the fastest baseline (3.67 s at 50 frames) but consumes 17.16 GB—9% more than ELSA-FP32.

Method	Frames	Time (s)	Mem (GB)	Speedup
xFormers-FP32	50	18.36	15.74	1.00×
	100	61.75	26.67	1.00×
	150	130.83	37.59	1.00×
FA2-FP16	50	**3.67**	17.16	–
	100	**9.44**	27.13	–
	150	17.88	37.43	–
ELSA-FP32	50	**12.56**	**15.74**	1.46×
	100	**29.49**	**26.68**	2.09×
	150	**55.97**	**37.59**	2.34×

Table 3. **VGGT baseline.** ELSA-FP32 achieves 1.46–2.34× speedup over xFormers-FP32 while matching its memory footprint. Speedup is relative to xFormers-FP32; shading intensity reflects speedup magnitude.

Results on FastVGGT Scaling. Table 4 demonstrates ELSA's scalability on longer sequences. At 400 frames, ELSA-FP32 completes inference in 158.04 s versus xFormers' 218.30 s (**1.38× speedup**), with identical 32.70 GB memory. The Math baseline runs out of memory beyond 30 frames due to its $O(n^2)$ attention matrix materialization. Across all frame counts, ELSA's memory overhead relative to xFormers remains negligible (<0.01 GB), confirming that the $O(n)$ auxiliary state (m, S, W) imposes no practical burden at these scales.

Analysis. ELSA's $O(\log n)$ scan depth confers two key advantages for multi-view 3D reconstruction: (1) efficient parallelization over long token sequences (10K–100K tokens across views), and (2) reduced global synchronization overhead compared to xFormers' $O(n/T_k)$ sequential tiling. At 400 frames with 1,041 tokens per frame (416,400 tokens total), the number of global synchronization points drops from ∼3,250 (xFormers, $T_k = 128$) to ∼15 (scan-tree depth $\lceil \log_2(400 \times 1041/128) \rceil + 3 \approx 15$), directly explaining the observed 1.38× end-to-end speedup.

H. CUDA C++ Backend and Portability

While our primary implementation uses Triton for rapid prototyping, we provide a CUDA C++ backend for deploy-

Method	Frames	Time (s)	Mem (GB)	Speedup
FA2-FP16	15	1.76	4.42	–
	30	2.38	4.42	–
	50	3.35	5.04	–
	100	6.26	7.85	–
	150	9.97	10.66	–
	200	14.75	13.47	–
	300	27.08	19.09	–
	400	46.14	24.72	–
Math-FP32	15	4.93	8.58	0.68×
	30	13.06	18.08	0.45×
	>30		*OOM*	
Math-FP16	15	5.76	7.13	0.59×
	30	17.68	18.72	0.34×
	>30		*OOM*	
xFormers-FP32	15	3.37	6.26	1.00×
	30	5.94	6.55	1.00×
	100	23.94	11.50	1.00×
	150	43.05	15.03	1.00×
	200	65.81	18.56	1.00×
	300	129.97	25.63	1.00×
	400	218.30	32.70	1.00×
ELSA-FP32	15	**3.58**	**6.26**	0.94×
	30	**6.66**	**6.55**	0.89×
	100	**20.22**	**11.50**	1.18×
	150	**34.93**	**15.03**	1.23×
	200	**50.76**	**18.56**	1.30×
	300	**97.89**	**25.63**	1.33×
	400	**158.04**	**32.70**	1.38×

Table 4. **FastVGGT scaling.** ELSA-FP32 sustains 1.18–1.38× speedup at long sequences with identical memory overhead. Math baselines fail beyond 30 frames (OOM); shading intensity reflects speedup magnitude, red denotes below-baseline performance.

ments without JIT support or requiring finer control over the memory hierarchy. Both backends share the same algorithmic core—a two-level prefix scan over the monoid (m, S, W)—differing only in how that scan is realized: within each block we apply a Hillis–Steele [4] or Kogge–Stone [5] scan, and across blocks a Blelloch scan [1] via cooperative groups and CUB's `BlockScan` primitive [7], ensuring portability across GPU generations. Vectorization

Method	Metric	4K	8K	32K	65K
FP32 offloading, LLaMA-13B, batch 1					
ELSA	Throughput (M tok/s)	0.364	0.956	**1.460**	**0.882**
	Wait (ms/layer)	0.727	1.005	**467.2**	**1694.5**
SDPA	Throughput (M tok/s)	**0.632**	**1.278**	1.239	0.734
	Wait (ms/layer)	**0.228**	**0.322**	540.1	2069.4
Throughput gain		−42.4%	−25.2%	**+17.8%**	**+20.2%**

Table 5. **LLaMA-13B host-device offloading.** At ≤8K tokens the workload is PCIe-bound and ELSA's per-layer overhead exceeds its memory savings. At ≥32K tokens, ELSA's reduced footprint hides transfer latency more effectively, delivering **17.8–20.2%** throughput gains. *bestgray* shading marks the winner per metric per sequence length.

Sequence length	ELSA (tok/s)	SDPA (tok/s)	Gain	Latency ↓
4K	5,729.6	5,636.5	+1.7%	1.6%
8K	4,862.9	4,440.6	**+9.5%**	**8.7%**
16K	3,636.4	3,237.7	**+12.3%**	**11.0%**

Table 6. **LLaMA-8B direct inference.** ELSA vs. ME-SDPA; gains scale with sequence length, reaching +12.3% throughput and 11.0% latency reduction at 16K tokens.

width and block size are exposed as template parameters and autotuned at runtime. Critically, the backend avoids Tensor-Core (HMMA/GMMA) instructions, enabling FP32 execution on devices such as Jetson TX2 where Tensor Cores are absent. Benchmarks on A100, L4, and Jetson confirm that the C++ backend achieves performance within 10–15% of the Triton implementation while preserving cross-hardware compatibility.

The C++ backend's fine-grained memory control proves particularly advantageous under host-device offloading, where layer weights are streamed from CPU memory one layer at a time. Table 5 reports results for LLaMA-13B [10] (FP32, batch 1) using asynchronous double-buffering across four sequence lengths. At 4K and 8K tokens the workload is PCIe-bound: transfers complete faster than ELSA's scan overhead, causing higher per-layer idle time (0.727 ms and 1.005 ms vs. 0.228 ms and 0.322 ms for SDPA), and SDPA holds a throughput edge (−42.4% and −25.2% for ELSA). At ≥32K tokens, however, ELSA's lower memory footprint allows computation to begin earlier, reducing per-layer wait from 540 ms to 467 ms and from 2069 ms to 1695 ms respectively, translating to **17.8% and 20.2% throughput gains** over SDPA, confirming that ELSA's memory efficiency compounds into end-to-end gains as sequences and transfer times grow. Table 6 further reports LLaMA-8B direct inference (no offloading), where ELSA's throughput gain scales from +1.7% at 4K to **+12.3%** at 16K tokens.

Method	Throughput (M tok/s)	Memory (GB)
ME-SDPA	0.40	0.289
ELSA-FP32-strict	1.36	0.336
ELSA-TF32-Turbo	**1.43**	**0.288**
ELSA-FP32-lean (η=0.25)	0.13	**0.288**

Table 7. **FP32 variant comparison.** ELSA variants versus ME-SDPA at 16K tokens. Throughput in millions of tokens per second; memory is peak GPU usage. Strict FP32 (`allow_tf32=0`) is used throughout the main paper; TF32-Turbo is an optional higher-throughput variant.

I. ELSA Variants

We examine ELSA's performance across different precision modes, long-context workloads, and offload scenarios. Unless otherwise stated, throughput is reported in millions of tokens per second (M tok/s) and memory refers to peak GPU usage.

FP32 variants and memory trade-offs. The main paper reports both strict FP32 and TF32-Turbo results. Strict FP32 is the primary headline for exact-attention claims; TF32-Turbo is an optional higher-throughput variant. We additionally benchmark a *memory-lean* variant (streaming smaller key-value chunks controlled by η), which trades memory for throughput. Table 7 compares these variants against ME-SDPA at 16K tokens.

Strict FP32 achieves $3.4\times$ throughput improvement (1.36M vs. 0.40M tok/s) at a modest memory increase from 0.289 GB to 0.336 GB. ELSA-TF32-Turbo delivers an additional 5% speedup (1.43M tok/s) while preserving FP32-quality outputs; numerical error remains within 10^{-4} relative to strict FP32. The memory-lean variant (η=0.25) matches ME-SDPA's 0.288 GB footprint while sustaining 0.13M tok/s. Taken together, these variants span a continuum from maximum throughput to minimal memory, allowing practitioners to select the operating point best suited to their hardware constraints.

FP16 regime and FlashAttention comparison. At 16K tokens in FP16, ELSA delivers 2.49M tok/s with 0.19 GB memory. FlashAttention-2 [2] achieves 2.88M tok/s ($\sim$15% faster) but requires 0.29 GB; FlashAttention-3 [8] reaches 2.73M tok/s at 0.36 GB. ME-SDPA sustains only 1.47M tok/s despite matching ELSA's 0.19 GB footprint. Across sequence lengths, FA2 leads in raw throughput, but ELSA consistently achieves the lowest memory usage among exact kernels and narrows the speed gap as sequences grow. ELSA thus carries its long-context advantages into FP16 while remaining competitive with hardware-fused kernels.

Batch size	Latency (ms)		Memory (GB)		Speed-up
	Math	ELSA	Math	ELSA	
1	0.405	**0.399** (+1.5%)	0.0098	**0.0095** (+3.1%)	1.02
4	0.427	**0.424** (+0.7%)	0.0107	**0.0100** (+6.5%)	1.01
8	0.430	**0.411** (+4.4%)	0.0131	**0.0116** (+11.5%)	1.05
16	0.431	**0.409** (+5.1%)	0.0175	**0.0138** (+21.1%)	1.05

Table 8. **Swin-v2 window attention (FP16) across batch sizes.** Latency and memory vs. Math baseline; ELSA columns shaded, green denotes improvement. ELSA consistently reduces both latency and memory with no retraining.

J. Limitations

Arithmetic Complexity. ELSA eliminates the $O(n^2)$ memory bottleneck but preserves quadratic arithmetic complexity: the prefix-scan formulation parallelizes computation and reduces memory traffic to $O(n)$, yet still performs all pairwise interactions. The term *linear-scan* refers strictly to the additional memory and I/O overhead, not to arithmetic complexity; ELSA is therefore best suited to regimes where *memory* rather than *compute* is the primary bottleneck.

Short-Sequence and Window Attention. For windowed MHSA (e.g., Swin with $n \leq 196$), the $O(\log n)$ scan-depth advantage diminishes and per-level merge overhead becomes visible, while FA2/3 are highly optimized for small tiles. Table 8 shows that ELSA consistently outperforms the unfused Math baseline in both latency and memory—improving by up to 5.1% and 21.1% respectively—yet FA2 remains faster in this short-window FP16 regime. This is consistent with ELSA's design goal: long/global contexts and FP32 portability, where fused kernels provide no compatible alternative.

Training and Multi-GPU Scaling. The prefix-scan formulation introduces inter-block data dependencies that complicate sequence parallelism: the monoid reduction across blocks must complete before downstream layers proceed, creating synchronization barriers that do not exist in standard attention. Efficient multi-GPU scaling therefore requires careful pipeline scheduling to overlap these reductions with communication, which remains future work.

Hardware Coverage. The current kernel is validated primarily on Ampere GPUs (A100, L4) and Jetson TX2; broader coverage across Hopper, Ada Lovelace, and non-NVIDIA accelerators is left to future work.

References

[1] Guy E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, Carnegie Mellon University, 1990. 8

[2] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. arXiv preprint arXiv:2307.08691, 2023. 9

[3] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, second edition, 2002. 3

[4] W. Daniel Hillis and Guy L. Steele. Data parallel algorithms. Communications of the ACM, 29(12):1170–1183, 1986. 8

[5] Peter M. Kogge and Harold S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. IEEE Transactions on Computers, 22(8): 786–793, 1973. 8

[6] Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. arXiv preprint arXiv:1805.02867, 2018. 4

[7] NVIDIA Corporation. CUB: CUDA unbound. https://github.com/NVIDIA/cccl, 2024. 8

[8] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. NeurIPS, 37:68658–68685, 2024. 9

[9] You Shen, Zhipeng Zhang, Yansong Qu, and Liujuan Cao. FastVGGT: Training-free acceleration of visual geometry transformer. arXiv preprint arXiv:2509.02560, 2025. 7

[10] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971, 2023. 8

[11] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual geometry grounded transformer. In CVPR, 2025. 7