Supplemental Material: Streaming Semi-CRF Inference

This supplement provides formal algorithmic details for the streaming Semi-CRF implementation.

1 Notation

Symbol	Description	Shape
T	Sequence length	scalar
K	Maximum segment duration	scalar
C	Number of labels (states)	scalar
C_{PAD}	Padded label count (next power of 2)	scalar
B	Batch size	scalar
$\mathcal{S}_{t,c}$	Cumulative projected scores	$(B, T+1, C)$
$\mathcal{T}_{c',c}$	Transition scores (source c' to dest c)	(C, C)
$\mathcal{B}_{k,c}$	Duration bias for duration k, label c	(K, C)
$\tilde{\alpha}_t(c)$	Log-forward message at position t, label c	(B, C)
$\tilde{\beta}_t(c)$	Log-backward message	(B, C)
$\boldsymbol{\alpha}$	Forward ring buffer	(B, K, C)
$\boldsymbol{\beta}$	Backward ring buffer	$(B, 2K, C)$
Ω	Checkpointed ring buffer states	(B, N, K, C)
$\mathcal{N}$	Cumulative log-normalization factors	(B, N)
Δ	Checkpoint interval ($\approx \sqrt{TK}$)	scalar
τ	Tile size for label dimension	scalar
$\log Z$	Log partition function	$(B,)$

Table 1: Notation. Tilde ($\tilde{\cdot}$) denotes log-domain quantities. $N = \lceil T/\Delta \rceil$ is the number of checkpoints.

2 Edge Potential Decomposition

The key innovation enabling $O(KC)$ memory is computing edge potentials on-the-fly from cumulative scores:

$$\tilde{\psi}(t, k, c, c') = \underbrace{(\mathcal{S}_{t,c} - \mathcal{S}_{t-k,c})}_{\text{segment content}} + \underbrace{\mathcal{B}_{k,c}}_{\text{duration bias}} + \underbrace{\mathcal{T}_{c',c}}_{\text{transition}} \tag{1}$$

where t is the segment end position (1-indexed), $k \in \{1, \ldots, K\}$ is the duration, c is the destination state, and c' is the source state. The prefix-sum decomposition allows $O(1)$ edge computation instead of $O(k)$.

3 Streaming Forward Algorithm

Algorithm 1 maintains a ring buffer $\boldsymbol{\alpha} \in \mathbb{R}^{K \times C}$ storing the K most recent forward messages. Following the Flash Attention [1] and Mamba [2] pattern, we normalize at checkpoint boundaries to prevent overflow at extreme T.

Memory: $O(KC)$ ring buffer + $O(\sqrt{T/K} \cdot KC)$ checkpoints. **Time**: $O(TKC^2)$.

Algorithm 1: Streaming Semi-CRF Forward Scan

Input: $\mathcal{S}$, $\mathcal{T}$, $\mathcal{B}$, checkpoint interval Δ
Output: $\log Z$, checkpoints $(\Omega, \mathcal{N})$

$\boldsymbol{\alpha} \leftarrow -\infty$; $\quad \boldsymbol{\alpha}[0, :] \leftarrow 0$; $\quad \mathcal{N}_{\text{accum}} \leftarrow 0$;
$\Omega[0] \leftarrow \boldsymbol{\alpha}$; $\quad \mathcal{N}[0] \leftarrow 0$;
for $t \leftarrow 1$ **to** T **do**
  $\mathbf{v}_t \leftarrow -\infty \in \mathbb{R}^C$;
  for $k \leftarrow 1$ **to** $\min(K, t)$ **do**
    $\tilde{\boldsymbol{\alpha}}_{\text{prev}} \leftarrow \boldsymbol{\alpha}[(t-k) \bmod K, :]$;
    $\mathbf{h} \leftarrow \left(\mathcal{S}_{t,:} - \mathcal{S}_{t-k,:}\right) + \mathcal{B}_{k,:}$;
    $\mathbf{E} \leftarrow \mathbf{h}[:, \text{None}] + \mathcal{T}^\top$;
    $\mathbf{s}_k \leftarrow \text{LSE}(\tilde{\boldsymbol{\alpha}}_{\text{prev}}[\text{None}, :] + \mathbf{E}, \text{axis} = 1)$;
    $\mathbf{v}_t \leftarrow \text{LSE}(\mathbf{v}_t, \mathbf{s}_k)$;
  $\boldsymbol{\alpha}[t \bmod K, :] \leftarrow \mathbf{v}_t$;
  if $t \bmod \Delta = 0$ **then** `// Normalize at checkpoint [1]`
    $n \leftarrow t/\Delta$; $\quad s \leftarrow \max_c \mathbf{v}_t(c)$;
    $\mathcal{N}_{\text{accum}} \leftarrow \mathcal{N}_{\text{accum}} + s$; $\quad \boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - s$;
    $\Omega[n] \leftarrow \boldsymbol{\alpha}$; $\quad \mathcal{N}[n] \leftarrow \mathcal{N}_{\text{accum}}$;
$\log Z \leftarrow \text{LSE}(\boldsymbol{\alpha}[T \bmod K, :]) + \mathcal{N}_{\text{accum}}$;
return $\log Z$, $(\Omega, \mathcal{N})$;

4 Backward Pass with Checkpointing

Algorithm 2 processes segments in reverse, recomputing α from checkpoints following gradient checkpointing [3]. The saved $\mathcal{N}_i$ restores the true scale when computing marginals.

5 $2K$ Beta Ring Buffer

While the forward pass uses a ring buffer of size K (sufficient since we only look back K positions), the backward pass requires a ring buffer of size $2K$. This asymmetry arises from the different access patterns:

Forward pass access pattern. At position t, we read α values from positions $\{t-K, t-K+1, \ldots, t-1\}$—exactly K previous positions. A ring buffer indexed by $t \bmod K$ suffices.

Backward pass access pattern. At position t, we read β values from positions $\{t+1, t+2, \ldots, t+K\}$—$K$ future positions. However, we also write $\beta[t]$ while potentially still reading $\beta[t+K]$. If we used a ring buffer of size K, we would have:

$$\text{write index:} \quad t \bmod K \tag{2}$$

$$\text{read index for } k = K: \quad (t+K) \bmod K = t \bmod K \tag{3}$$

This creates a **write-before-read hazard**: we would overwrite $\beta[t+K]$ before reading it. The $2K$ ring buffer eliminates this conflict:

$$t \bmod 2K \neq (t+K) \bmod 2K \quad \text{for all } t \geq 0 \tag{4}$$

Memory cost. The additional KC floats is negligible compared to the $O(\sqrt{T/K} \cdot KC)$ checkpoint storage.

Algorithm 2: Streaming Semi-CRF Backward Pass

Input: $\mathcal{S}$, $\mathcal{T}$, $\mathcal{B}$, checkpoints $(\Omega, \mathcal{N})$, $\log Z$, upstream $\partial\mathcal{L}/\partial Z$
Output: $\nabla\mathcal{S}$, $\nabla\mathcal{T}$, $\nabla\mathcal{B}$

$\boldsymbol{\beta} \leftarrow -\infty \in \mathbb{R}^{2K \times C}$; $\quad \boldsymbol{\beta}[T \bmod 2K, :] \leftarrow 0$;
Initialize gradient accumulators;
for $i \leftarrow N_{ckpts} - 1$ **to** 0 **do**
  $t_{\text{start}} \leftarrow i \cdot \Delta$; $\quad t_{\text{end}} \leftarrow \min((i+1) \cdot \Delta, T)$;
  $\mathcal{N}_i \leftarrow \mathcal{N}[i]$;
  $\boldsymbol{\alpha}_{\text{local}} \leftarrow$ RecomputeAlpha($\Omega[i]$, t_{start}, t_{end});
  for $t \leftarrow t_{end} - 1$ **to** t_{start} **do**
    $\tilde{\boldsymbol{\alpha}}_t \leftarrow \boldsymbol{\alpha}_{\text{local}}[t - t_{\text{start}}, :]$;
    $\tilde{\boldsymbol{\beta}}_t \leftarrow -\infty$;
    for $k \leftarrow 1$ **to** $\min(K, T - t)$ **do**
      $\tilde{\boldsymbol{\beta}}_{\text{next}} \leftarrow \boldsymbol{\beta}[(t+k) \bmod 2K, :]$; `// 2K ring buffer`
      $\tilde{\psi} \leftarrow$ Eq. (1);
      `// Log-norm correction restores true` α `scale`
      $\log \mu \leftarrow \tilde{\boldsymbol{\alpha}}_t[\text{None}, :] + \tilde{\psi} + \tilde{\boldsymbol{\beta}}_{\text{next}}[:, \text{None}] + \mathcal{N}_i - \log Z$;
      $\mu \leftarrow \exp(\log \mu)$;
      Accumulate $\nabla\mathcal{S}$, $\nabla\mathcal{T}$, $\nabla\mathcal{B}$ from μ;
      $\tilde{\boldsymbol{\beta}}_t \leftarrow \text{LSE}(\tilde{\boldsymbol{\beta}}_t, \text{LSE}(\tilde{\psi} + \tilde{\boldsymbol{\beta}}_{\text{next}}[:, \text{None}], \text{axis} = 0))$;
    $\boldsymbol{\beta}[t \bmod 2K, :] \leftarrow \tilde{\boldsymbol{\beta}}_t$; `// 2K ring buffer`
return $\nabla\mathcal{S}$, $\nabla\mathcal{T}$, $\nabla\mathcal{B}$;

6 Adaptive Loop Tiling

The marginal computation requires a $(C_{\text{PAD}} \times C_{\text{PAD}})$ matrix per (t, k) pair, which at $C_{\text{PAD}} = 64$ demands approximately 384 registers per thread. With 4+ warps, this exceeds available registers and causes spilling to slow local memory.

Tiled computation. We process the destination label dimension c_{dst} in tiles of size τ (TILE_C):

1. Load a $(\tau \times C_{\text{PAD}})$ tile of the marginal matrix
2. Accumulate gradients from the tile
3. Use online logsumexp for β update across tiles (Flash Attention pattern [1])

This reduces peak register demand to approximately 120 per thread, enabling 4–8 warps without spilling.

Adaptive tile sizing. The tile size τ is selected based on C to balance compile time, register pressure, and iteration count:

Online logsumexp for β. Since β reduction spans multiple tiles, we use the Flash Attention online pattern:

$$m^{(i)} = \max(m^{(i-1)}, m^{(i)}_{\text{tile}}) \tag{5}$$

$$\ell^{(i)} = \ell^{(i-1)} \cdot e^{m^{(i-1)} - m^{(i)}} + \sum_j e^{x_j^{(i)} - m^{(i)}} \tag{6}$$

C_{PAD}	τ	Iterations	Rationale
≤ 8	4	2	Minimal iteration count
≤ 16	8	2	Minimal iteration count
32	16	2	Balanced
64	16	4	Moderate register pressure
≥ 128	32	≤ 8	Bounded compile time

Table 2: Adaptive tile size selection via `_compute_tile_c()`. The algorithm bounds iteration count to ≤ 8 even at $C = 256$ while keeping register pressure manageable.

where m tracks the running maximum and ℓ tracks the running sum of exponentials, rescaled at each tile boundary.

7 Reduced Atomics Strategy

GPU atomic operations are expensive and introduce non-determinism due to floating-point non-associativity. We employ three strategies to minimize their use:

7.1 Local Accumulation for Per-Position Gradients

For $\nabla\mathcal{S}_{t,c}$, the negative contribution (from segments starting at t) is the same position for all k values. Instead of $K \times$ tiles atomic operations, we accumulate locally:

Algorithm 3: Local Accumulation for $\nabla\mathcal{S}_t$

```
grad_cs_t_local ← 0 ∈ ℝ^{C_PAD};
for k ← 1 to K do
    for each tile do
        grad_cs_t_local −= Σ_{c'} μ_tile(t, k, c, c');          // Register accumulation
atomic_add(∇S_t, grad_cs_t_local);                               // Single write
```

Speedup: $K \times$ tiles $\to 1$ atomic per position (e.g., $1000 \times 4 = 4000 \to 1$ at $K = 1000$, $C = 64$).

7.2 Per-Duration Accumulation for Duration Bias

For $\nabla\mathcal{B}_{k,c}$, we accumulate across all tiles for each k, then write once:

Algorithm 4: Per-Duration Accumulation for $\nabla\mathcal{B}_k$

```
for k ← 1 to K do
    grad_db_k_local ← 0 ∈ ℝ^{C_PAD};
    for each tile do
        grad_db_k_local += Σ_{c'} μ_tile(t, k, c, c');
    atomic_add(∇B_k, grad_db_k_local);                           // Once per k
```

Speedup: tiles $\to 1$ atomic per (t, k) pair.

7.3 Segment-Isolated Workspace Buffers

For shared parameters ($\nabla\mathcal{T}$, $\nabla\mathcal{B}$), cross-segment atomic contention introduces non-determinism. We allocate per-segment workspace buffers:

$$\texttt{grad\_tr\_workspace} \in \mathbb{R}^{B \times N_{\text{segments}} \times C \times C} \tag{7}$$

Each checkpoint segment writes to its own slice, eliminating inter-segment atomics. The final reduction uses deterministic host-side operations:

$$\nabla \mathcal{T}_{c',c} = \texttt{einsum}(\text{“bsij, b} \to \text{ij”}, \texttt{workspace.sum(dim=1)}, \nabla_{\text{out}}) \tag{8}$$

$$\nabla \mathcal{B}_{k,c} = \texttt{einsum}(\text{“bskc, b} \to \text{kc”}, \texttt{workspace.sum(dim=1)}, \nabla_{\text{out}}) \tag{9}$$

Determinism: The segment-wise sum is deterministic (fixed order), and `einsum` performs a single reduction pass.

Memory cost: $O(B \cdot N_{\text{segments}} \cdot K \cdot C^2)$ workspace, which is acceptable since $N_{\text{segments}} \approx \sqrt{T/K}$ is typically small (e.g., 10–100 for $T = 100\text{k}$, $K = 1000$).

8 Gradient Computation

Gradients are computed via marginal probabilities:

$$\mu(t, k, c, c') = \exp\big(\tilde{\alpha}_{t-k}(c') + \tilde{\psi}(t, k, c, c') + \tilde{\beta}_t(c) + \mathcal{N}_i - \log Z\big) \tag{10}$$

For per-sequence parameters:

$$\nabla \mathcal{S}_{t,c} = \frac{\partial \mathcal{L}}{\partial Z_b} \cdot \left(\sum_{k,c'} \mu_b(t, k, c, c') - \sum_{k,c'} \mu_b(t + k, k, c, c') \right) \tag{11}$$

For shared parameters, we accumulate per-batch then reduce via einsum:

$$\nabla \mathcal{T}_{c',c} = \sum_b \frac{\partial \mathcal{L}}{\partial Z_b} \cdot \sum_{t,k} \mu_b(t, k, c, c') \tag{12}$$

$$\nabla \mathcal{B}_{k,c} = \sum_b \frac{\partial \mathcal{L}}{\partial Z_b} \cdot \sum_{t,c'} \mu_b(t, k, c, c') \tag{13}$$

9 Numerical Stability

Zero-centering. Cumulative scores are zero-centered before cumsum to bound magnitude at large T: $\bar{s}_{t,c} = s_{t,c} - \frac{1}{T} \sum_\tau s_{\tau,c}$.

Log-domain. All computations use logsumexp: $\text{LSE}(\mathbf{x}) = \max(\mathbf{x}) + \log \sum_i \exp(x_i - \max(\mathbf{x}))$.

NEG_INF guards. When all logsumexp inputs are -10^9, the subtraction $x - \max(x) = 0$ instead of remaining at $-\infty$. Guards detect this case ($\max < -10^9 + 1$) and return $-\infty$ directly.

Float64 accumulation. Gradient tensors for shared parameters use float64 to prevent non-determinism from atomic_add floating-point non-associativity. Error scales as $O(\sqrt{T \times K \times C})$ per operation; float64 reduces this from $\sim 10^{-3}$ (float32) to $\sim 10^{-10}$ (negligible).

Log-marginal clamping. Before exp(), log-marginals are clamped to $[-700, 700]$ to prevent float64 overflow ($\exp(710) \approx \infty$).

Masking. Invalid positions use -10^9 (not $-\infty$) to avoid NaN in gradients.

Variable-length batches. For sequences ending at $L < T$, the normalization shift is masked to zero for $t > L$, freezing $\mathcal{N}_{\text{accum}}$ at the correct value.

Math	Code Variable	Notes
$\mathcal{S}_{t,c}$	`cum_scores[:, t, c]`	Boundary at index 0
$\mathcal{T}_{c',c}$	`transition[c_src, c_dst]`	Source-first storage
$\mathcal{B}_{k,c}$	`duration_bias[k-1, c]`	0-indexed in code
$\boldsymbol{\alpha}$	`ring_buffer` / `alpha_ring`	Size K
$\boldsymbol{\beta}$	`beta_ring`	Size $2K$ (Section 5)
Ω	`ring_checkpoints`	Saved at intervals
$\mathcal{N}$	`log_norm_checkpoints`	Cumulative log-norm
Δ	`checkpoint_interval`	$\approx \sqrt{TK}$
C_{PAD}	`C_PAD`	`_next_power_of_2(C)`
τ	`TILE_C`	Adaptive (Section 6)
—	`grad_tr_workspace`	(B, N, C, C) or (B, N, K, C, C)
—	`grad_db_workspace`	(B, N, K, C)

Table 3: Mapping between mathematical notation and implementation. N denotes number of segments/checkpoints.

Component	Time	Memory
Forward scan	$O(TKC^2)$	$O(KC)$ ring buffer
Checkpoints	—	$O(\sqrt{T/K} \cdot KC)$
Backward scan	$O(TKC^2)$	$O(KC)$ ring buffer ($2K$ slots)
Alpha recompute	$O(TKC^2)$	$O(\Delta \cdot C)$ per segment
Gradient workspace	—	$O(B \cdot N \cdot KC^2)$
Total	$O(TKC^2)$	$O(KC + \sqrt{T/K} \cdot KC)$

Table 4: Complexity analysis. Memory is independent of T (excluding checkpoints).

10 Implementation Correspondence

11 Complexity Summary

References

[1] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *NeurIPS*, 2022.

[2] Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[3] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training Deep Nets with Sublinear Memory Cost. *arXiv preprint arXiv:1604.06174*, 2016.