Supplemental Material: Semi-Markov CRF Backend Algorithms

This supplement provides formal algorithmic details for the Semi-Markov CRF inference backends. We establish unified notation, present each algorithm with complexity analysis, describe the semiring abstraction enabling algorithm reuse, and summarize why sparse matrix representations fail for exact tree-structured inference.

1 Notation

Symbol	Description	Shape/Domain
T	Sequence length	scalar
K	Maximum segment duration	scalar
C	Number of labels (states)	scalar
B	Batch size	scalar
n	State-space size: $n = (K-1) \cdot C$	scalar
$\psi(t,k,c',c)$	Edge potential: segment ending at t with duration k, transition $c' \to c$	$(B, T-1, K, C, C)$
$\tilde{\alpha}_t(c)$	Log-forward message at position t, label c	(B, C)
$\mathbf{W}_t$	Transition matrix at position t for tree reduction	(n, n)
$\mathcal{S}_{t,c}$	Cumulative projected scores (streaming)	$(B, T+1, C)$
$\mathcal{T}_{c',c}$	Transition scores	(C, C)
$\mathcal{B}_{k,c}$	Duration bias	(K, C)

Table 1: Notation. Tilde ($\tilde{\cdot}$) denotes log-domain quantities.

Important Distinction: Edge Tensor Materialization. All backends in this supplement assume the edge potential tensor $\psi \in \mathbb{R}^{B\times(T-1)\times K\times C\times C}$ is **pre-materialized** before inference. This $O(TKC^2)$ tensor dominates memory for large T, regardless of which backend processes it.

The "streaming linear scan" (Section 9) streams through the *forward messages* α_t using a ring buffer, but still requires the full edge tensor as input. True T-independent memory—where edge potentials are computed on-the-fly from $O(TC)$ cumulative scores via Eq. (2)—is achieved only in the **fused Triton kernel** (see separate supplement: "Streaming Semi-CRF Inference").

2 Semi-Markov CRF Formulation

A Semi-Markov CRF models segmentations $S = ((s_1, e_1, \ell_1), \ldots, (s_M, e_M, \ell_M))$ where segments are contiguous, non-overlapping, cover $\{1, \ldots, T\}$, and have durations $d_m \in \{1, \ldots, K\}$:

$$p_\theta(S \mid x) = \frac{1}{Z_\theta(x)} \exp\left(\sum_{m=1}^{M} \psi_\theta(x_{s_m:e_m}, \ell_{m-1}, \ell_m, d_m)\right) \tag{1}$$

The edge potential decomposes as:

$$\psi(t,k,c',c) = \psi_{\text{emission}}(x_{t:t+k}, c) + \psi_{\text{transition}}(c', c) + \psi_{\text{duration}}(c, k) \tag{2}$$

The log-domain forward recursion is:

$$\tilde{\alpha}_t(c) = \text{LSE}_{k,c'}\left[\tilde{\alpha}_{t-k}(c') + \tilde{\psi}(t-k, k, c', c)\right] \tag{3}$$

with $\tilde{\alpha}_0(c) = 0$ and $\log Z = \text{LSE}_c\, \tilde{\alpha}_T(c)$.

3 Semiring Abstraction

A key design principle is that the *same* dynamic program computes different quantities by changing the semiring operations [2]. A semiring $(K, \oplus, \otimes, \bar{0}, \bar{1})$ provides:

- Addition $\oplus$ (commutative, associative) with identity $\bar{0}$
- Multiplication $\otimes$ (associative, distributes over $\oplus$) with identity $\bar{1}$

Semiring	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$	Computes
Log	logsumexp	$+$	$-\infty$	0	Partition; gradients $\rightarrow$ marginals
Max	max	$+$	$-\infty$	0	Viterbi score; gradients $\rightarrow$ argmax
K-Max	top-k	$+$	$[-\infty]^k$	$[0, -\infty, \ldots]$	k-best scores
Entropy	(expectation semiring)				Shannon entropy $H(P)$

Table 2: Semirings for structured prediction. The same DP algorithm yields different outputs.

Gradient Checkpointing Wrappers. For large state spaces, we provide semiring wrappers that trade compute for memory by recomputing forward passes during backward [3]:

- `CheckpointSemiring(cls)`: Basic checkpointing for $2\times$ compute, reduced memory
- `CheckpointShardSemiring(cls, max_size)`: Sharded checkpointing for very large matrices

4 Provenance

Backend	**Source**
Binary tree (dense)	pytorch-struct [1]
Binary tree (sharded)	pytorch-struct [1]
Linear scan (standard)	pytorch-struct [1]
Linear scan (vectorized)	This work
Streaming linear (PyTorch)	This work
Block-triangular	This work
Banded (analysis)	This work
Streaming (Triton kernel)	This work

Table 3: Algorithm provenance. The Triton kernel (bottom) is the key contribution: it computes edge potentials on-the-fly, achieving true T-independent memory and enabling genome-scale inference.

5 Backend 1: Binary Tree (Parallel Scan)

The binary tree algorithm is the default Semi-Markov implementation in `pytorch-struct` [1]. It reformulates forward recursion as semiring matrix products over an expanded state space of size $n = (K - 1) \cdot C$.

State Space. Each state (k, c) represents "k time steps remaining in a segment of label c." The transition matrix $\mathbf{W}_t \in \mathbb{R}^{n \times n}$ encodes valid transitions:

$$\mathbf{W}_t[(k_1, c_1), (k_2, c_2)] = \begin{cases} \tilde{\psi}(t, k_2, c_1, c_2) & \text{if } k_1 = 1 \\ 0 & \text{if } k_1 > 1, k_2 = k_1 - 1, c_2 = c_1 \\ -\infty & \text{otherwise} \end{cases} \tag{4}$$

Parallel Prefix. The forward pass is $\mathbf{A}_{1:T} = \mathbf{W}_1 \otimes \mathbf{W}_2 \otimes \cdots \otimes \mathbf{W}_T$, computed via binary tree reduction in $O(\log T)$ parallel depth.

Complexity. Time: $O(T(KC)^3)$. Space: $O(T(KC)^2)$. The quadratic memory in KC causes OOM before the depth advantage is realized.

Sharded Variant. The sharded variant, also from `pytorch-struct` [1], adds gradient checkpointing [3] to reduce memory to $O(\sqrt{T}(KC)^2)$ with $2\times$ compute overhead. Extends viable regime from $n < 100$ to $n < 150$.

6 Backend 2: Block-Triangular Sparsity (Experimental)

At a tree node of span S, the duration constraint $k_1 + k_2 \leq S$ means only $\frac{K(K+1)}{2}$ duration pairs are feasible, yielding block-triangular sparsity.

Representation. Store only blocks (k_1, k_2) satisfying $k_1 + k_2 \leq S$, each block being dense $C \times C$. This halves storage vs. dense at relevant spans.

Sparse Multiplication. For block-triangular matrices $\mathbf{C}, \mathbf{D}$:

$$\mathbf{E}[k_1, k_3] = \bigoplus_{k_2 : k_1 + k_2 \leq S,\, k_2 + k_3 \leq S} \mathbf{C}[k_1, k_2] \otimes \mathbf{D}[k_2, k_3] \tag{5}$$

Practical Outcome. In PyTorch/CUDA experiments, block-triangular storage did **not** yield speedups at typical sparsity levels. The overhead of indirect indexing and non-contiguous memory access outweighed savings from skipping $\sim$50% of blocks. GPUs prefer well-tuned dense kernels unless sparsity exceeds $\sim$90%.

7 Backend 3: Banded Storage (Not Viable)

Bounded segment length K suggests locality, motivating banded matrix storage. Two structural facts defeat this:

Local Geometry Mismatch. The constraint $k_1 + k_2 \leq S$ induces *anti-diagonal triangular* sparsity, not diagonal-banded. States with $d \leq \lfloor S/2 \rfloor$ form a clique of size $\lfloor S/2 \rfloor \cdot C$, forcing bandwidth $\geq C\lfloor S/2 \rfloor - 1$ under any permutation.

Fill-in Under Composition. Even for genuinely banded one-step operators, m-step reachability has bandwidth $\min(T, mK)$. In a balanced tree, m doubles per level, so bandwidth saturates at dense width after $O(\log(T/K))$ levels.

Empirical Result. Bandwidth ratios $\text{bw}_{\text{best}}/(n-1) > 0.90$ for spans $S > K/2$, confirming negligible benefit at tree levels that dominate computation.

See standalone supplement "Why Banded Matrices Do Not Provide a Viable Exact Backend" for complete proofs.

8 Backend 4: Linear Scan

The standard linear scan is the reference Semi-Markov implementation in `pytorch-struct` [1]. It iterates sequentially over positions, avoiding the $O((KC)^2)$ intermediate storage of tree backends.

Algorithm 1: Linear Scan Forward

```
Input: Edge potentials $\tilde{\boldsymbol{\Psi}} \in \mathbb{R}^{B\times(T-1)\times K\times C\times C}$
Output: Log partition $\log Z \in \mathbb{R}^B$
1 $\boldsymbol{\beta}[0] \leftarrow \mathbf{0}$;
2 for $t \leftarrow 1$ to $T$ do
3     for $k \leftarrow 1$ to $\min(K, t)$ do
4         $\boldsymbol{\alpha}[t-1,k] \leftarrow \text{LSE}_{c'}\left(\boldsymbol{\beta}[t-k] + \tilde{\boldsymbol{\Psi}}_{t-1,k}\right)$;
5     $\boldsymbol{\beta}[t] \leftarrow \text{LSE}_k\, \boldsymbol{\alpha}[t-k,k]$;
6 $\log Z \leftarrow \text{LSE}_c\, \boldsymbol{\beta}[T]$;
```

Vectorized Variant. Replaces inner loops with batched tensor operations (broadcasting, `gather`), yielding 2–3× speedup.

Complexity. Time: $O(TKC^2)$. Space: $O(TKC)$ or $O(KC)$ with ring buffer. Depth: $O(T)$.

9 Backend 5: Streaming Linear Scan

The streaming linear scan reduces *forward message* memory from $O(TKC)$ to $O(KC)$ via a ring buffer. However, this PyTorch implementation still requires the pre-materialized edge tensor as input.

Ring Buffer. Store only the K most recent α values: $\tilde{\alpha}_t(c) = \boldsymbol{\alpha}[t \bmod K, c]$.

Complexity. Time: $O(TKC^2)$. Space: $O(TKC^2)$ for edge tensor + $O(KC)$ for ring buffer.

Edge Decomposition (Triton Kernel). True T-independent memory requires computing edges on-the-fly:

$$\tilde{\psi}(t,k,c',c) = \underbrace{(\mathcal{S}_{t+k,c} - \mathcal{S}_{t,c})}_{\text{content}} + \mathcal{B}_{k,c} + \mathcal{T}_{c',c} \tag{6}$$

where $\mathcal{S}_{t,c} = \sum_{\tau=0}^{t-1} s_{\tau,c}$ are $O(TC)$ cumulative scores. This decomposition, combined with the ring buffer and checkpoint-boundary normalization, enables the fused Triton kernel to achieve $O(KC)$ memory independent of T.

See "Supplement: Streaming Semi-CRF Inference" for the complete Triton kernel algorithm including checkpoint normalization, backward pass, and numerical stability at $T > 100K$.

10 Complexity Summary

Backend	Work	Memory	Depth	Notes
Binary tree (dense)	$O(Tn^3)$	$O(Tn^2)$	$O(\log T)$	$n < 100$
Binary tree (sharded)	$O(Tn^3)$	$O(\sqrt{T}n^2)$	$O(\sqrt{T})$	$n < 150$
Block-triangular	$O(Tn^3)$	$\sim \frac{1}{2}O(Tn^2)$	$O(\log T)$	Not faster
Banded	$O(Tn^3)$	$\to O(Tn^2)$	$O(\log T)$	Not viable
Linear scan	$O(TKC^2)$	$O(TKC^2)^*$	$O(T)$	General
Streaming linear (PyTorch)	$O(TKC^2)$	$O(TKC^2)^*$	$O(T)$	Ring buffer for α
Streaming (Triton kernel)	$O(TKC^2)$	$O(KC)^\dagger$	$O(T)$	**Genome-scale**

Table 4: Backend comparison. $n = (K-1)\cdot C$. *Dominated by pre-materialized edge tensor. †Edges computed on-the-fly; see streaming supplement.

11 Duration Distributions

The duration bias $\mathcal{B}_{k,c}$ can be learned freely or constrained to parametric forms:

Distribution	$\log P(k \mid c)$	Parameters
Learned	$\mathcal{B}_{k,c}$ (free)	$K \times C$
Geometric	$\log p_c + (k-1)\log(1-p_c)$	$p_c \in (0,1)$ per class
Negative Binomial	$\log\binom{k+r_c-2}{k-1} + r_c \log p_c + (k-1)\log(1-p_c)$	p_c, r_c per class
Poisson	$k \log \lambda_c - \log k! - \lambda_c$	$\lambda_c > 0$ per class
Uniform	0	None

Table 5: Duration distribution options. Parametric forms reduce parameters and encode inductive bias.

References

[1] A. M. Rush. Torch-Struct: Deep Structured Prediction Library. In *ACL*, 2020.

[2] J. Goodman. Semiring Parsing. *Computational Linguistics*, 25(4):573–605, 1999.

[3] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training Deep Nets with Sublinear Memory Cost. *arXiv:1604.06174*, 2016.