Supplemental Material: Linear and Near-Linear CRF Implementations

This supplement documents the specialized $K = 1$ (linear CRF) and $K = 2$ (near-linear CRF) implementations in `torch-semimarkov`. These optimized paths eliminate ring buffer overhead for common sequence labeling tasks while maintaining compatibility with the streaming Semi-CRF interface.

1 Overview

The streaming Semi-CRF module automatically dispatches to specialized implementations based on the maximum segment duration K:

K	Implementation	Rationale
$K = 1$	`LinearCRFStreaming`	Standard linear-chain CRF; no duration loop
$K = 2$	`SemiCRFK2Streaming`	Explicit 2-step history; avoids ring buffer edge cases
$K \geq 3$	`SemiCRFStreamingTriton`	Full ring buffer architecture

Table 1: Automatic dispatch by maximum segment duration.

Triton Kernel Scope. The specialized $K = 1$ and $K = 2$ paths are implemented **only in PyTorch**. The Triton streaming kernel requires $K \geq 3$ for correct ring buffer operation. This is intentional: the PyTorch implementations are already efficient for small K, and the Triton kernel's complexity is justified only when ring buffers and checkpointing provide substantial memory savings.

2 Notation

Symbol	**Description**	**Shape**
T	Sequence length	scalar
C	Number of labels (states)	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)

Table 2: Notation. Tilde $(\tilde{\cdot})$ denotes log-domain quantities.

3 K=1: Linear CRF

When $K = 1$, every segment has duration 1, reducing the Semi-CRF to a standard linear-chain CRF. The forward recurrence simplifies to:

$$\tilde{\alpha}_t(c) = \text{emit}_t(c) + \log \sum_{c'=1}^{C} \exp\bigl(\tilde{\alpha}_{t-1}(c') + \mathcal{T}_{c',c}\bigr) \tag{1}$$

where the emission score is computed via prefix-sum difference:

$$\text{emit}_t(c) = \mathcal{S}_{t,c} - \mathcal{S}_{t-1,c} + \mathcal{B}_{0,c} \tag{2}$$

3.1 Initialization Convention

Following the `pytorch-crf` convention, we use **uniform initialization**:

$$\tilde{\alpha}_0(c) = 0 \quad \forall c \in \{1, \ldots, C\} \tag{3}$$

This is equivalent to assuming a uniform distribution over initial states. At $t = 1$:

$$\tilde{\alpha}_1(c) = \text{emit}_1(c) + \log \sum_{c'=1}^{C} \exp(\mathcal{T}_{c',c}) \tag{4}$$

Comparison with Explicit Start Transitions. Some implementations (e.g., `pytorch-crf`) use explicit start transition parameters π_c^{start}:

$$\tilde{\alpha}_1(c) = \pi_c^{\text{start}} + \text{emit}_1(c) \tag{5}$$

The two approaches are **functionally equivalent** when:

$$\pi_c^{\text{start}} = \log \sum_{c'=1}^{C} \exp(\mathcal{T}_{c',c}) \tag{6}$$

Both define valid probability distributions; the choice affects only the parameterization, not predictive accuracy.

3.2 Algorithm

Algorithm 1 presents the optimized $K = 1$ forward pass.

Algorithm 1: Linear CRF Forward ($K = 1$)

Input: Cumulative scores $\mathcal{S} \in \mathbb{R}^{B \times (T+1) \times C}$, Transitions $\mathcal{T} \in \mathbb{R}^{C \times C}$, Duration bias $\mathcal{B} \in \mathbb{R}^{1 \times C}$ (optional), Lengths $\ell \in \mathbb{Z}^B$

Output: Log partition $\log Z \in \mathbb{R}^B$

$\tilde{\boldsymbol{\alpha}} \leftarrow \mathbf{0} \in \mathbb{R}^{B \times C}$; `// Uniform initialization`

for $t \leftarrow 1$ **to** T **do**

  $\mathbf{e}_t \leftarrow \mathcal{S}[:, t, :] - \mathcal{S}[:, t-1, :] + \mathcal{B}_0$; `// Emission`

  `// Standard linear CRF recurrence`

  $\tilde{\boldsymbol{\alpha}}_{\text{new}} \leftarrow \text{LogSumExp}_{c'}(\tilde{\boldsymbol{\alpha}}[:, c'] + \mathcal{T}_{c',:}) + \mathbf{e}_t$;

  `// Update only active sequences`

  $\tilde{\boldsymbol{\alpha}} \leftarrow$ `where`$(t \le \ell, \tilde{\boldsymbol{\alpha}}_{\text{new}}, \tilde{\boldsymbol{\alpha}})$;

$\log Z \leftarrow \text{LogSumExp}_c(\tilde{\boldsymbol{\alpha}}[\ell])$; `// Final reduction`

return $\log Z$;

3.3 Complexity

- **Time**: $O(TC^2)$ — matrix-vector products at each timestep
- **Space**: $O(BC)$ — single α vector per batch element
- **No checkpointing**: Full α history stored for backward pass ($O(BTC)$)

The space overhead for storing full α history is acceptable because $K = 1$ implies short-to-moderate sequences where the $O(BTC)$ cost is manageable.

4 K=2: Near-Linear CRF

When $K = 2$, segments can have duration 1 or 2. Rather than using the general ring buffer (which has edge cases at $K = 2$ due to modular arithmetic with small indices), we use explicit 2-step history variables.

4.1 Forward Recurrence

At each position t, we combine contributions from both durations:

$$\tilde{\alpha}_t(c) = \text{LogSumExp}\Big(\underbrace{\text{score}_{k=1}(t,c)}_{\text{duration 1}}, \underbrace{\text{score}_{k=2}(t,c)}_{\text{duration 2}}\Big) \tag{7}$$

where:

$$\text{score}_{k=1}(t,c) = \text{emit}_{t-1:t}(c) + \log \sum_{c'} \exp(\tilde{\alpha}_{t-1}(c') + \mathcal{T}_{c',c}) \tag{8}$$

$$\text{score}_{k=2}(t,c) = \text{emit}_{t-2:t}(c) + \log \sum_{c'} \exp(\tilde{\alpha}_{t-2}(c') + \mathcal{T}_{c',c}) \tag{9}$$

The emission scores are:

$$\text{emit}_{t-1:t}(c) = \mathcal{S}_{t,c} - \mathcal{S}_{t-1,c} + \mathcal{B}_{0,c} \tag{10}$$

$$\text{emit}_{t-2:t}(c) = \mathcal{S}_{t,c} - \mathcal{S}_{t-2,c} + \mathcal{B}_{1,c} \tag{11}$$

4.2 Algorithm

Algorithm 2 presents the optimized $K = 2$ forward pass using explicit history variables.

Algorithm 2: Semi-CRF Forward ($K = 2$)

Input: Cumulative scores $\mathcal{S}$, Transitions $\mathcal{T}$, Duration bias $\mathcal{B} \in \mathbb{R}^{2\times C}$, Lengths ℓ
Output: Log partition $\log Z \in \mathbb{R}^{B}$

$\tilde{\boldsymbol{\alpha}}^{(1)} \leftarrow \mathbf{0} \in \mathbb{R}^{B\times C}$; `//` $\alpha[t-1]$
$\tilde{\boldsymbol{\alpha}}^{(2)} \leftarrow -\infty \in \mathbb{R}^{B\times C}$; `//` $\alpha[t-2]$ `(invalid at` $t = 1$`)`
for $t \leftarrow 1$ **to** T **do**
  `// Duration` $k = 1$`: segment from` $t-1$ `to` t
  $\mathbf{e}_{k=1} \leftarrow \mathcal{S}[:, t, :] - \mathcal{S}[:, t-1, :] + \mathcal{B}_0$;
  $\mathbf{s}_{k=1} \leftarrow \text{LogSumExp}_{c'}(\tilde{\boldsymbol{\alpha}}^{(1)}[:, c'] + \mathcal{T}_{c',:}) + \mathbf{e}_{k=1}$;
  if $t \geq 2$ **then**
    `// Duration` $k = 2$`: segment from` $t-2$ `to` t
    $\mathbf{e}_{k=2} \leftarrow \mathcal{S}[:, t, :] - \mathcal{S}[:, t-2, :] + \mathcal{B}_1$;
    $\mathbf{s}_{k=2} \leftarrow \text{LogSumExp}_{c'}(\tilde{\boldsymbol{\alpha}}^{(2)}[:, c'] + \mathcal{T}_{c',:}) + \mathbf{e}_{k=2}$;
    $\tilde{\boldsymbol{\alpha}}_{\text{new}} \leftarrow \text{LogSumExp}(\mathbf{s}_{k=1}, \mathbf{s}_{k=2})$;
  else
    $\tilde{\boldsymbol{\alpha}}_{\text{new}} \leftarrow \mathbf{s}_{k=1}$;
  `// Shift history`
  $\tilde{\boldsymbol{\alpha}}^{(2)} \leftarrow \tilde{\boldsymbol{\alpha}}^{(1)}$;
  $\tilde{\boldsymbol{\alpha}}^{(1)} \leftarrow \texttt{where}(t \leq \ell, \tilde{\boldsymbol{\alpha}}_{\text{new}}, \tilde{\boldsymbol{\alpha}}^{(1)})$;
$\log Z \leftarrow \text{LogSumExp}_c(\tilde{\boldsymbol{\alpha}}^{(1)}[\ell])$;
return $\log Z$;

4.3 Complexity

- **Time**: $O(TC^2)$ — two matrix-vector products per timestep
- **Space**: $O(BC)$ — two α vectors per batch element
- **No ring buffer**: Explicit variables avoid modular arithmetic overhead

5 Backward Pass and Gradients

Both $K = 1$ and $K = 2$ implementations use the standard forward-backward algorithm for gradient computation. Because these paths do not use checkpointing, the full α history is stored during the forward pass.

5.1 Marginal Computation

The marginal probability for a segment spanning $[t-k, t-1]$ with transition $c' \to c$ is:

$$\mu(t, k, c, c') = \frac{\exp\bigl(\tilde{\alpha}_{t-k}(c') + \tilde{\psi}(t, k, c, c') + \tilde{\beta}_t(c)\bigr)}{\exp(\log Z)} \tag{12}$$

where the edge potential is:

$$\tilde{\psi}(t, k, c, c') = (\mathcal{S}_{t,c} - \mathcal{S}_{t-k,c}) + \mathcal{B}_{k-1,c} + \mathcal{T}_{c',c} \tag{13}$$

5.2 Gradient Formulas

Cumulative Scores. The gradient for $\mathcal{S}_{t,c}$ accumulates contributions from segments ending at t (positive) and starting after t (negative):

$$\nabla \mathcal{S}_{t,c} = \sum_{k,c'} \mu(t,k,c,c') - \sum_{k,c'} \mu(t+k,k,c,c') \tag{14}$$

Transitions. The gradient sums marginals over all positions and durations:

$$\nabla \mathcal{T}_{c',c} = \sum_{b} \frac{\partial \mathcal{L}}{\partial Z_b} \cdot \sum_{t,k} \mu_b(t,k,c,c') \tag{15}$$

Duration Bias. The gradient for duration k sums over all segments of that duration:

$$\nabla \mathcal{B}_{k,c} = \sum_{b} \frac{\partial \mathcal{L}}{\partial Z_b} \cdot \sum_{t,c'} \mu_b(t,k,c,c') \tag{16}$$

6 Viterbi Decoding

Both $K = 1$ and $K = 2$ implementations support Viterbi decoding for MAP inference. The algorithms follow the same structure as the forward pass, replacing LogSumExp with max and maintaining backpointers.

6.1 K=1 Viterbi

Standard linear CRF Viterbi with backpointers for the best previous label at each position.

6.2 K=2 Viterbi

Extended backpointers track both the best previous label **and** the best duration (1 or 2) at each position. The traceback reconstructs the segmentation by stepping back by the recorded duration.

7 Why Triton Kernels Require $K \geq 3$

The Triton streaming kernel is designed for the general Semi-CRF case where ring buffers and checkpointing provide substantial memory savings. For small K, ring buffer indexing creates edge cases:

- $K = 1$: All positions map to ring index 0. The ring buffer degenerates to a single slot, providing no benefit over a simple variable.
- $K = 2$: Indices alternate between 0 and 1. Wraparound logic is fragile and provides minimal memory savings over explicit variables.
- $K \geq 3$: Ring buffer provides meaningful history separation. Checkpointing amortizes over multiple segments, justifying the implementation complexity.

The PyTorch implementations for $K = 1$ and $K = 2$ are already efficient (no kernel launch overhead, simple control flow), making Triton acceleration unnecessary.

K	Time	Space	Checkpointing	Ring Buffer	Backend
1	$O(TC^2)$	$O(BC)$	No	No	PyTorch
2	$O(TC^2)$	$O(BC)$	No	No	PyTorch
≥ 3	$O(TKC^2)$	$O(BKC)$	Yes	Yes	Triton (GPU) / PyTorch (CPU)

Table 3: Implementation characteristics by maximum segment duration.

8 Implementation Summary

Automatic Dispatch. The `semi_crf_streaming_forward` function automatically routes to the appropriate implementation based on K. Users need not manage dispatch manually.

9 Functional Equivalence

All three implementations ($K = 1$, $K = 2$, $K \geq 3$) compute the same partition function and gradients for their respective segment duration constraints. Specifically:

- The $K = 1$ path produces identical results to the general streaming kernel with $K = 1$ (verified numerically).
- The $K = 2$ path produces identical results to the general streaming kernel with $K = 2$ (verified numerically).
- All paths support variable-length batches with proper masking.

The specialized paths exist purely for performance optimization, not behavioral differences.

10 References

1. Lafferty, J., McCallum, A., & Pereira, F. (2001). *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data.* ICML.
2. Sarawagi, S., & Cohen, W. W. (2004). *Semi-Markov Conditional Random Fields for Information Extraction.* NeurIPS.
3. `pytorch-crf`: `https://github.com/kmkurn/pytorch-crf`