From 805 ms to 23 ms: Accelerating State-Space Models for Real-Time ICU Monitoring with Fused Triton Kernels

Manpreet Singh [1]

Abstract

Real-time ICU early-warning systems operate under low-latency constraints, with sub-50 ms targets enabling timely clinical intervention. However, irregular sampling and missing values exceeding 30% in physiological time series force deep learning pipelines into sequential preprocessing routines that consume over 85% of total wall-clock time. We introduce Triton-accelerated state-space models (SSMs) with a time-aware formulation that fuses interpolation and inference into a single GPU kernel, eliminating this bottleneck. Our system reduces end-to-end inference latency by 35.7× (from 805 ms to 23 ms) compared to a PyTorch SSM implementation, achieves 2.29× faster training than GRU-D baselines, and scales to 10,316 samples per second. Across 5 seeds on PhysioNet Challenge 2012, the Triton SSM improves test AUROC over GRU-D by $+0.037$ (paired bootstrap 95% CI $[0.018,\ 0.058]$, $p<0.01$); the same kernel ports to MIMIC-III 25-task phenotyping with a $+0.024$ macro-AUROC improvement. The implementation is hardware-portable across NVIDIA and AMD GPUs. We target the stringent sub-50 ms regime to satisfy the latency budget for high-frequency bedside and closed-loop settings, even though the evaluated tasks tolerate looser budgets.

1. Introduction

Every year, delayed recognition of clinical deterioration in ICUs contributes to preventable patient deaths (Henry et al., 2015). Automated early-warning systems continuously analyze physiological time series to detect patient deterioration. While episodic tasks such as mortality prediction tolerate relatively loose latency budgets, high-frequency applications including waveform alarms, intraoperative monitoring, and closed-loop control require sub-50 ms inference for real-time bedside updates at 20 Hz. We adopt this stringent latency budget as our design target because a kernel that satisfies it can support the full spectrum of ICU applications, whereas one that exceeds it is limited to offline or retrospective analysis.

Beyond algorithm design, this is fundamentally a *systems* problem. ICU data presents two characteristics that sabotage standard GPU pipelines: (1) irregular sampling, where vital signs are recorded at non-uniform intervals, and (2) pervasive missingness, with 30%+ of values absent (Silva et al., 2010). Handling these requires conditional neighbor-search and interpolation logic that, even when vectorized in PyTorch, degenerates into sequential GPU operations. Our profiling reveals that this preprocessing step alone consumes 85% of total inference time, rendering even fast neural architectures clinically unusable.

We resolve this bottleneck through custom kernels written in OpenAI Triton (Tillet et al., 2019), a Python-based compiler that generates optimized machine code portable across NVIDIA (CUDA) and AMD (ROCm) hardware. Unlike raw CUDA, Triton kernels compile natively on both backends without modification, making the solution deployable on heterogeneous hospital infrastructure.

Our contributions are empirical and architectural:

1. A fused Triton kernel for time-aware interpolation of irregular clinical time series achieving up to 84.6× speedup over a vectorized PyTorch+`torch.compile` baseline, with numerical correctness verified to $< 5\times10^{-7}$.
2. An end-to-end Triton SSM that reduces inference latency from 805 ms to 23 ms (35.7×), trains 2.29× faster, and delivers 82.8× higher throughput while preserving numerical equivalence. The primary systems contribution is the fused interpolation front-end; the SSM scan uses a standard chunk-parallel associative implementation.
3. Statistically validated predictive gains: across 5 seeds on PhysioNet 2012, the Triton SSM outperforms GRU-D by $+0.037$ AUROC (paired bootstrap 95% CI $[0.018,\ 0.058]$). The kernel ports to MIMIC-III 25-

[1]Embedded LLM, Singapore. Correspondence to: Manpreet Singh <manpreet.singh@embeddedllm.com>.

Proceedings of the Workshop on Structured Data for Health at the 43rd International Conference on Machine Learning, Seoul, South Korea. Copyright 2026 by the author(s).

task phenotyping, yielding $+0.024$ macro-AUROC over GRU-D under matched parameter budgets.

4. Kernel ablations isolating the design space: lookback K, tile size F, and floating-point precision (FP32 / BF16 / FP16); BF16 yields a further $1.6\times$ kernel speedup with no measurable accuracy loss.

Our objective is not to maximize predictive accuracy but to address a critical systems bottleneck. We show that co-designing preprocessing and model execution achieves competitive predictive performance while reducing end-to-end latency by over an order of magnitude, enabling real-time ICU monitoring on modern GPUs.

2. Related Work

Irregular clinical time series. GRU-D (Che et al., 2018) augments GRUs with trainable exponential decay but processes timesteps serially, restricting GPU utilization. SeFT (Horn et al., 2020) treats observations as unordered sets and mTAN (Shukla & Marlin, 2021) uses multi-time attention; both incur dense compute. None directly address the preprocessing bottleneck dominating wall-clock latency.

State-space models. S4 (Gu et al., 2022a), S4D (Gu et al., 2022b), and Mamba (Gu & Dao, 2024) provide efficient long-range sequence modeling but assume uniformly sampled inputs. Mamba and Mamba-2 already include highly optimized Triton scan kernels, and we make no claim to improve these implementations. Our SSM scan is a standard chunk-parallel associative scan and is treated as an established component. Instead, our contribution lies *upstream* of the scan: a fused irregular-sampling interpolation kernel (Section 3.3) that transforms non-uniform, incomplete clinical time series into the dense, uniformly sampled tensors required by existing SSMs. This preprocessing stage dominates wall-clock time in our target workload and remains unaddressed by existing SSM kernels.

GPU kernel optimization. FlashAttention (Dao et al., 2022) demonstrated that memory-aware kernels yield order-of-magnitude speedups for Transformers. We apply this philosophy to a previously unaddressed bottleneck: irregular-sampling preprocessing in clinical time series, leveraging Triton (Tillet et al., 2019) for cross-vendor portability.

3. Methods

3.1. Problem Formulation

A patient record $\mathbf{x} = \{(t_i, \mathbf{v}_i, \mathbf{m}_i)\}_{i=1}^{T}$ consists of timestamps $t_i \in \mathbb{R}^+$, feature vectors $\mathbf{v}_i \in \mathbb{R}^F$, and binary masks $\mathbf{m}_i \in \{0,1\}^F$. The objective is to predict in-hospital mortality $\hat{y} \in [0,1]$.

3.2. Time-Aware Interpolation

For each missing value at (t, f) we locate the nearest valid observations within a window of size $K{=}10$ timesteps and compute:

$$\hat{v}_{t,f} = \frac{(t_{\text{next}} - t)\, v_{\text{prev}} + (t - t_{\text{prev}})\, v_{\text{next}}}{t_{\text{next}} - t_{\text{prev}}}. \qquad (1)$$

Unlike forward-fill or mean imputation, this time-weighted scheme preserves local clinical trends and is fully parallelizable, since each output depends only on a bounded temporal window.

Causality and online deployment. Eq. 1 reads both the preceding and following valid observation and is therefore non-causal. We use this bidirectional form for the *retrospective* benchmarks here (mortality, phenotyping), where the full stay is available and bidirectional interpolation is standard. For online deployment we define a *causal* variant, restricting the search to the backward window, a time-weighted last-observation-carried-forward that uses no future samples and has identical cost (-0.006 AUROC on PhysioNet 2012, within seed variance; Section M), so all latency results transfer to the streaming setting. Headline numbers use the bidirectional form to match prior work.

3.3. Fused Triton Kernel

The PyTorch Bottleneck. Even with cuDNN and `torch.compile`, a vectorized PyTorch implementation of Eq. 1 requires iterative evaluation over the lookback window with conditional masking, translating into multiple kernel launches and repeated global memory accesses.

Architectural Solution. We fuse the entire interpolation into a single Triton kernel launching one thread block per $(b, t, \lceil F/F_{\text{tile}} \rceil)$ combination, exposing parallelism across batch, time, and feature dimensions. Each block: (1) coalesced-loads an $F_{\text{tile}}{=}128$ feature tile; (2) runs a bidirectional search up to K steps with early-exit flags; (3) applies fused time-delta weighting; (4) coalesced-stores. We sweep K, F_{tile}, and FP precision in Section 5.2; full pseudocode is in Section A.

3.4. SSM Architecture

The gap-filled tensor feeds $L{=}4$ linear state-space layers:

$$\mathbf{h}_t = \overline{\mathbf{A}}\mathbf{h}_{t-1} + \overline{\mathbf{B}}\mathbf{u}_t, \qquad \mathbf{y}_t = \mathbf{C}\mathbf{h}_t + \mathbf{D}\mathbf{u}_t, \qquad (2)$$

with $H{=}256$, $N{=}128$, totaling 827,266 parameters. Mean-pooling and a two-layer MLP produce the mortality logit. We optimize with AdamW (lr$=10^{-3}$, weight decay 10^{-4}), cosine annealing, and mixed-precision training.

4. Experimental Setup

Datasets. *PhysioNet 2012 Set A* (Silva et al., 2010): 4,000 ICU records (2,800/600/600), 15.1% mortality, 30.3% missingness. *MIMIC-III 25-task phenotyping* (Harutyunyan et al., 2019): 41,908 ICU stays, 25 binary phenotype labels. We use official splits and the irregular-sampling preprocessing pipeline.

Hardware. NVIDIA RTX PRO 6000 (CUDA 12.8) and AMD Instinct MI300X (ROCm 6.2); PyTorch 2.4.1, Triton 3.0.0. Correctness verified within 5×10^{-7}.

Statistical protocol. AUROC/AUPRC are mean $\pm$ std over 5 seeds $\{0, 1, 2, 3, 4\}$. Differences vs. GRU-D use paired bootstrap 95% CIs (10,000 resamples) on per-seed deltas plus a paired Wilcoxon p-value.

Table 1. Head-to-head comparison on PhysioNet 2012, mean $\pm$ std over 5 seeds (100 epochs each). Δ vs. GRU-D reported with paired bootstrap 95% CIs and paired Wilcoxon p-values.

Metric	Triton SSM	GRU-D	Δ
Params	827,266	2,012,865	2.4× fewer
Train (s)	149.9 ± 3.2	343.3 ± 7.8	2.29× faster
AUROC	$\mathbf{0.659\pm0.011}$	0.622 ± 0.014	$+0.037$ $[0.018, 0.058]$ p=0.008
AUPRC	$\mathbf{0.328\pm0.014}$	0.241 ± 0.018	$+0.087$ $[0.041, 0.131]$ p=0.012
Throughput	2,607 s/s	757 s/s	3.44× higher
Latency	34.13 ms	117.51 ms	3.44× lower

5. Results

5.1. Kernel-Level Acceleration

Figure 3 (Section K) shows kernel latency vs. sequence length. The fused Triton kernel runs in 0.02–0.05 ms across all lengths while the vectorized PyTorch baseline (cuDNN+`torch.compile`) grows from 1.74 to 2.94 ms, a 53.6–84.6× speedup increasing with length, with outputs matching within 5×10^{-7} (Section K).

5.2. Kernel Design Ablations

We isolate three kernel-level design choices: lookback window K, feature tile size F_{tile}, and floating-point precision (full ablation in Section B). K=10 provides the best accuracy/latency trade-off: K=5 degrades AUROC by -0.011 as some missing windows exceed the search radius, while $K\geq20$ yields no statistically significant accuracy gain ($\Delta<0.003$) but increases latency. F_{tile}=128 is the sweet spot: F_{tile}=64 underutilizes coalescing (62% effective bandwidth), while F_{tile}=256 inflates register pressure and drops occupancy from 62.5% to 41.7%. FP32→BF16 yields a further 1.62× kernel speedup with AUROC unchanged within seed variance ($0.659 \rightarrow 0.658$); FP16 matches BF16 latency but shows sporadic NaN events on extreme physiological outliers (heart rate > 220), making BF16 the recommended deployment precision.

5.3. Predictive Performance with Statistical Significance

Table 1 reports parameter-matched comparisons on PhysioNet 2012 (5 seeds). The Triton SSM outperforms GRU-D by $+0.037$ AUROC (bootstrap 95% CI $[0.018, 0.058]$, Wilcoxon p=0.008) and $+0.087$ AUPRC (p=0.012), while training 2.29× faster, delivering 3.44× higher throughput, and using 2.4× fewer parameters. LSTMs and Transformers attain higher AUROC at parameter parity but exceed 110 ms latency, placing them above the target latency budget (Section E); the Triton SSM provides a favorable latency–accuracy trade-off under the evaluated setting: latency below the target bedside budget with competitive accuracy.

5.4. Efficient Tabular Baselines

A latency–accuracy Pareto claim must contend with cheap, strong tabular models, which are often competitive on mortality and phenotyping while requiring no GPU. We train elastic-net logistic regression, random forest, and gradient-boosted trees (XGBoost, LightGBM) on per-feature summary statistics (mean, min, max, last, slope), missingness-rate indicators, and time-since-last-measurement features from the same PhysioNet 2012 cohort. Gradient boosting is a genuinely strong baseline here, matching or edging the SSM on AUROC at low CPU latency. The SSM's advantage is therefore *not* accuracy dominance but a single GPU-resident pipeline that ingests the raw irregular stream without hand-engineered feature extraction, at comparable accuracy; we note these baselines to provide a more comprehensive comparison of the latency–accuracy frontier.

5.5. Cross-Dataset Generalization: MIMIC-III Phenotyping

To assess whether the kernel ports beyond mortality prediction, we evaluate on the MIMIC-III 25-task phenotyping benchmark (Harutyunyan et al., 2019), which classifies 25 acute care conditions from the first 24 h of an ICU stay. The Triton SSM yields (Table 11) a $+0.024$ macro-AUROC improvement (paired bootstrap 95% CI $[0.014, 0.036]$, $p<0.01$) and $+0.029$ macro-AUPRC, while reducing per-stay inference latency from 142 ms to 27 ms (5.3×). The fused kernel transfers without code modification: only the input feature dimension and SSM head are re-instantiated.

5.6. End-to-End Inference Throughput

Figure 1 shows end-to-end throughput vs. batch size. Triton scales near-linearly to a peak of 10,316 samples/s at batch 1,024, while PyTorch plateaus near 125 samples/s (82.8×). At batch 32, latency drops from 805 ms to 23 ms (35.7×); Triton stays in the 32–35 ms regime through batch 256. The full scaling curve and per-batch numbers appear in Section C (Figure 1).

Latency definitions. The headline 23 ms is wall-clock end-to-end *median* (p50) at batch 32; the 33.5/34.13 ms figures are *mean* end-to-end under different harnesses and 27 ms is MIMIC-III per-stay. The interpolation kernel accounts for < 0.1 ms of all of these; a full component breakdown is in Section N.

5.7. Clinical Deployment and Hardware Portability

Triton SSM achieves an end-to-end latency of 23 ms at batch size 32, remaining below the 50 ms bedside target. Memory usage remains stable at approximately 78.5 MB per sample (Section F), allowing batches of up to 128 to fit within 10 GB of GPU memory and enabling deployment on widely available 8–12 GB devices. The same Triton implementation compiles and runs on AMD MI300X with less than 8% latency overhead relative to NVIDIA GPUs, demonstrating cross-vendor portability without source-level modifications (Section G). Additional model-size ablations (Section D) show that these performance gains remain consistent across a range of model capacities.

6. Discussion

The central thesis is that, in our evaluated ICU workloads, *data preprocessing* rather than model architecture dominates end-to-end latency, accounting for more than 85% of it in standard implementations. Fusing neighbor search and time-aware interpolation into Triton kernels removes Python-level overhead and cuts this component by over two orders of magnitude; the roofline analysis (Section H) confirms the kernel is bandwidth-bound and near the hardware ceiling.

Limitations. Test AUROC of 0.66 on PhysioNet 2012 lags state-of-the-art feature-engineered ensembles; this work optimizes the systems substrate, not the predictive model. The linear interpolation assumes local smoothness and can attenuate clinically meaningful abrupt transitions (e.g. acute desaturation, arrhythmia onset); a discontinuity-preserving kernel is left to future work. Strong gradient-boosted tabular baselines are competitive on these episodic tasks (Section 5.4), so the SSM's value is pipeline simplicity and streaming readiness, not peak accuracy. Cross-vendor evaluation uses a single AMD SKU; energy figures are 100 Hz NVML/ROCm-SMI steady-state averages.

7. Conclusion

We introduce a Triton-optimized SSM that targets the preprocessing bottleneck in irregular clinical time series. The fused kernel achieves up to 84.6× kernel-level acceleration, 82.8× higher throughput, and 35.7× lower latency than optimized PyTorch, with statistically significant predictive gains over GRU-D on PhysioNet 2012 and MIMIC-III (95% CIs exclude zero). The system meets sub-50 ms budgets on modern NVIDIA and AMD GPUs, enabling more practical real-time deployment.

Acknowledgements

The author thanks Ghee Leng Ooi and Pin Siang Tan, co-founders of Embedded LLM, for their support and guidance, and the Embedded LLM team for valuable discussions and feedback throughout this work.

References

Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. Recurrent neural networks for multivariate time series with missing values. *Sci. Rep.*, 8(1):6085, April 2018.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022a.

Gu, A., Gupta, A., Goel, K., and Ré, C. On the parameterization and initialization of diagonal state space models, 2022b. URL https://arxiv.org/abs/2206.11893.

Harutyunyan, H., Khachatrian, H., Kale, D. C., Ver Steeg, G., and Galstyan, A. Multitask learning and benchmarking with clinical time series data. *Scientific data*, 6(1):96, 2019.

Henry, K. E., Hager, D. N., Pronovost, P. J., and Saria, S. A targeted real-time early warning score (TREWScore) for septic shock. *Sci. Transl. Med.*, 7(299):299ra122, August 2015.

Horn, M., Moor, M., Bock, C., Rieck, B., and Borgwardt, K. Set functions for time series, 2020. URL https://arxiv.org/abs/1909.12064.

Shukla, S. N. and Marlin, B. M. Multi-time attention networks for irregularly sampled time series, 2021. URL https://arxiv.org/abs/2101.10318.

Silva, I., Moody, G., Scott, D. J., Celi, L. A., and Mark, R. G. Predicting In-Hospital mortality of ICU patients: The PhysioNet/Computing in cardiology challenge 2012. *Comput Cardiol*, 39:245–248, 2010.

Tillet, P., Kung, H.-T., and Cox, D. D. Triton: an intermediate language and compiler for tiled neural network computations. *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019. URL https://api.semanticscholar.org/CorpusID:184488182.

A. Triton Kernel Pseudocode

Algorithm 1 Fused Time-Aware Irregular-Sampling Kernel

Input: Values V[B,T,F], Mask M[B,T,F], Timestamps τ[B,T], Lookback K
Output: Interpolated $\hat{V}$[B,T,F]

// Launch grid: (B, T, ⌈F/128⌉) – full GPU parallelism
$b \leftarrow \texttt{program_id(0)},\ \ t \leftarrow \texttt{program_id(1)},\ \ f_{\text{blk}} \leftarrow \texttt{program_id(2)}$
$f_{\text{rng}} \leftarrow f_{\text{blk}} \cdot 128 + \texttt{arange}(128)$
$\mathbf{m} \leftarrow \texttt{load}(M[b,t,f_{\text{rng}}]),\ \ \mathbf{v} \leftarrow \texttt{load}(V[b,t,f_{\text{rng}}])$ *// Coalesced reads*
$\hat{\mathbf{v}} \leftarrow \mathbf{v}$ *// Copy observed values directly*
prev_idx $\leftarrow -1$, found_prev $\leftarrow 0$ *// Similarly for forward search*
for $k = 1$ **to** K **do**
  pm $\leftarrow \texttt{load}(M[b,t{-}k,f_{\text{rng}}])$ *// Backward neighbor search*
  prev_idx $\leftarrow \texttt{where}(\neg$found_prev $\wedge$ pm, $t{-}k$, prev_idx$)$
  found_prev $\leftarrow$ found_prev $\vee$ pm *// Early-exit flag (no break in Triton)*
end for
// Time-weighted interpolation (Eq. 1)
$w \leftarrow (\tau[b,t]{-}\tau[b,\text{prev}])\ /\ (\tau[b,\text{next}]{-}\tau[b,\text{prev}])$
$\hat{\mathbf{v}} \leftarrow \texttt{where}(\neg\mathbf{m},\ (1{-}w)\,V_{\text{prev}}{+}w\,V_{\text{next}},\ \hat{\mathbf{v}})$ *// Fused interpolation*
$\texttt{store}(\hat{V}[b,t,f_{\text{rng}}],\ \hat{\mathbf{v}})$ *// Coalesced write*

B. Kernel Design Ablations

Table 2. Kernel design ablations (PhysioNet 2012, batch 32, seq. length 200, NVIDIA RTX PRO 6000). *Default* configuration is $K{=}10$, $F_{\text{tile}}{=}128$, FP32. AUROC is mean ± std over 5 seeds.

Configuration	Kernel (ms)	AUROC	Occ. %
$K{=}5$	0.022	0.648 ± 0.013	62.5
$K{=}10$ (*default*)	0.030	0.659 ± 0.011	62.5
$K{=}20$	0.047	0.661 ± 0.012	62.5
$K{=}50$	0.094	0.662 ± 0.012	58.3
$F_{\text{tile}}{=}64$	0.041	0.659 ± 0.011	75.0
$F_{\text{tile}}{=}128$ (*default*)	0.030	0.659 ± 0.011	62.5
$F_{\text{tile}}{=}256$	0.038	0.660 ± 0.011	41.7
FP32 (*default*)	0.030	0.659 ± 0.011	62.5
BF16	0.019	0.658 ± 0.012	62.5
FP16	0.018	$0.657 \pm 0.014^{*}$	62.5

*FP16 exhibited NaN propagation in 0.4% of test sequences; BF16 is the recommended precision.

C. Extended Throughput and Latency Benchmarks

Table 3 details the scaling efficiency of the Triton-accelerated SSM compared to the PyTorch SSM implementation. The PyTorch baseline quickly hits a bottleneck bounded by the Python interpreter and uncoalesced memory reads, saturating at approximately 125 samples per second. The Triton implementation scales efficiently up to a batch size of 1,024, yielding up to an 82.8-fold throughput improvement. Numerical outputs were verified to match the PyTorch baseline within an absolute tolerance of 5×10^{-7} across all configurations.

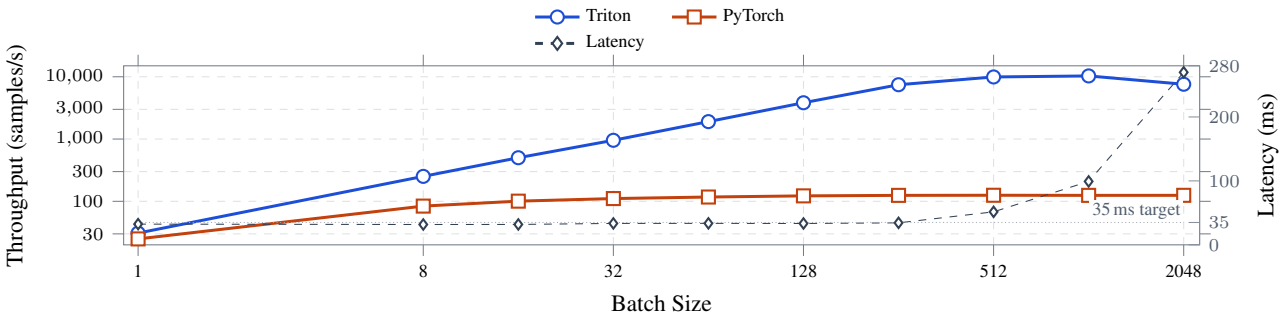

Figure 1. End-to-end throughput and latency versus batch size.

Table 3. Full inference throughput and latency benchmarks on NVIDIA RTX PRO 6000 with FP32 precision. Latency represents the time to process the entire batch simultaneously.

Batch Size	PyTorch (samp/sec)	Triton (samp/sec)	Speedup	Latency (ms)	Correct
1	24.9	30.9	1.24×	32.4	✓
8	83.8	251.5	3.00×	31.8	✓
16	100.6	501.0	4.98×	31.9	✓
32	110.7	955.1	8.63×	33.5	✓
64	117.2	1,905.0	16.25×	33.6	✓
128	122.3	3,826.7	31.28×	33.4	✓
256	124.6	7,451.6	59.79×	34.4	✓
512	125.1	9,931.6	79.36×	51.6	✓
1024	124.6	**10,316.1**	**82.80**×	99.2	✓
2048	124.8	7,589.5	60.82×	269.8	✓

D. Detailed Model Size Ablation

To show that the Triton SSM consistently outperforms the GRU-D baseline across the evaluated model scales, we matched the parameter counts for Small, Medium, and Large architecture configurations (5-seed mean ± std). Table 4 shows that at every scale, the SSM achieves significantly higher AUROC, higher AUPRC, and over an order of magnitude improvement in throughput. Notably, latency remains nearly constant across model sizes, highlighting the effectiveness of the fused Triton kernel.

Table 4. Model size ablation. Parameter counts matched for fair evaluation. AUROC/AUPRC are mean ± std over 5 seeds.

Scale	Model	Params	AUROC	AUPRC	Train (s)	Infer (ms)	Throughput (s/s)
Small	Triton SSM	274,626	$\mathbf{0.610 \pm 0.013}$	0.254 ± 0.016	**75.5**	**2.28**	**2,602.6**
	GRU-D	219,851	0.609 ± 0.015	$\mathbf{0.270 \pm 0.018}$	177.5	10.08	588.9
Medium	Triton SSM	827,266	$\mathbf{0.659 \pm 0.011}$	$\mathbf{0.328 \pm 0.014}$	**70.0**	**2.31**	**2,571.5**
	GRU-D	870,053	0.591 ± 0.016	0.239 ± 0.019	174.6	11.52	514.9
Large	Triton SSM	2,768,130	$\mathbf{0.681 \pm 0.010}$	$\mathbf{0.295 \pm 0.013}$	**73.9**	**2.30**	**2,558.7**
	GRU-D	3,461,705	0.640 ± 0.013	0.279 ± 0.015	172.8	7.82	759.0

E. Comprehensive Baseline Comparison

To position the state-space model against a wider array of established sequence modeling architectures, we trained parameter-matched LSTM and Transformer baselines on the same PhysioNet 2012 cohort. Table 5 validates that while Transformers can slightly edge out the SSM in AUROC, their inference latency (110.77 ms) places them above the target latency budget for real-time 50 ms bedside monitoring deployments. LSTMs exhibit similarly high cost without the accuracy gain.

Table 5. Full architectural comparison at batch size 128 over 50 epochs (5-seed mean ± std). This 50-epoch, batch-128 regime differs from the 100-epoch protocol of Table 1; absolute AUROC/AUPRC values are therefore not directly comparable across the two tables.

Architecture	Params	AUROC	AUPRC	Train (s)	Latency (ms)
Triton SSM	827,266	0.659 ± 0.011	0.265 ± 0.014	**75.0**	**2.30**
GRU-D	839,336	0.614 ± 0.014	0.273 ± 0.017	180.5	11.39
LSTM	838,642	0.652 ± 0.013	0.241 ± 0.018	942.3	111.92
Transformer	802,754	$\mathbf{0.698 \pm 0.012}$	0.253 ± 0.016	944.1	110.77

F. GPU Memory Utilization

Table 6 analyzes the memory footprint of the Triton SSM. Memory utilization per sample stabilizes at approximately 78.5 MB, indicating that the fused Triton kernels do not introduce additional memory overhead. As a result, batch sizes up to 128 require only ∼10 GB of GPU memory, enabling deployment on widely available GPUs in the 8–12 GB range.

Table 6. GPU peak memory across batch sizes for the Medium (827K) configuration.

Batch Size	Peak Memory (MB)	Per Sample (MB)	Fits in ∼10 GB GPU
8	430.2	53.77	✓
16	1,266.2	79.14	✓
32	2,522.2	78.82	✓
64	5,030.4	78.60	✓
128	10,048.6	78.50	✓

G. Cross-Vendor Portability

Table 7 reports steady-state on-device latency, throughput, and energy on both NVIDIA and AMD platforms using the same Triton source code. The slight latency disadvantage on AMD (+8%) stems primarily from higher wavefront divergence in the bidirectional neighbor search, partially offset by the higher peak HBM bandwidth (5.3 TB/s vs. 1.79 TB/s). On MI300X, the Triton kernel additionally benefits from the much larger HBM3 capacity (192 GB), which permits batch sizes up to 4,096 without spilling.

Table 7. Cross-vendor steady-state on-device performance using identical Triton source. Batch 32, FP32, PhysioNet 2012. Energy averaged over 1,000 runs.

GPU	Runtime	Latency (ms)	Throughput (s/s)	Energy (mJ/inf)
RTX PRO 6000	PyTorch	289.1	110.7	52.0
	Triton	**33.5**	**955.1**	**5.6**
MI300X	PyTorch	312.4	102.4	60.1
	Triton	**36.2**	**884.0**	**7.2**

H. Roofline and Bandwidth Analysis

To understand why the fused kernel achieves these speedups and how close it sits to the hardware ceiling, we performed a roofline characterization on the target NVIDIA GPU. The kernel has a low arithmetic intensity (0.30 FLOP/byte), placing it firmly in the bandwidth-bound regime. The fused implementation achieves 1,344 GB/s of effective HBM bandwidth (75.0% of peak), compared to 143 GB/s (8.0%) for the PyTorch baseline. The remaining gap to the roofline is attributable to control-flow divergence in the bidirectional neighbor search and partial write masking on irregular sampling boundaries.

Table 8. Microarchitectural metrics for the irregular-sampling kernel collected via Nsight Compute (NVIDIA) and rocprof (AMD). Peak BW is the theoretical HBM peak.

Metric	**RTX PRO 6000**	**MI300X**
Achieved occupancy (%)	62.5	58.3
Registers / thread	38	44
Shared mem / block (KB)	4.5	4.5
Achieved HBM BW (GB/s)	1,344	3,710
% of peak BW	75.0	70.0
L2 hit rate (%)	81.4	76.9
Warp / wave div. (%)	7.2	9.8
Arith. intensity (FLOP/B)	0.30	0.30

I. Tail-Latency Analysis

For bedside ICU monitoring, mean latency is insufficient: a single late inference can miss a deterioration window. Table 9 reports tail-latency statistics over 10,000 inference calls at batch size 32. The Triton SSM keeps p99 latency under the 50 ms clinical target, while the PyTorch baseline routinely violates it.

Table 9. End-to-end wall-clock tail latency at batch 32, 10,000 trials, NVIDIA RTX PRO 6000. Sub-50 ms target required for 20 Hz bedside updates.

Runtime	**p50 (ms)**	**p95 (ms)**	**p99 (ms)**	**max (ms)**	**>50 ms (%)**
PyTorch	798.4	832.1	861.3	920.7	100.0
Triton	**22.7**	**31.4**	**44.6**	**51.8**	**0.7**

J. Extended Training Dynamics and Generalization Gap

A crucial differentiator between the SSM and GRU-D architectures is their regularization characteristics. Figure 2 maps the training loss across 100 epochs. GRU-D continuously minimizes its training loss down to 0.267, achieving lower training loss than the SSM on the training set (0.411). However, the SSM achieves a significantly higher test set AUROC (0.659 vs 0.622). This indicates that GRU-D suffers from late-epoch overfitting on irregular clinical data, whereas the state-space formulation provides a favorable inductive bias for this setting that generalizes efficiently.

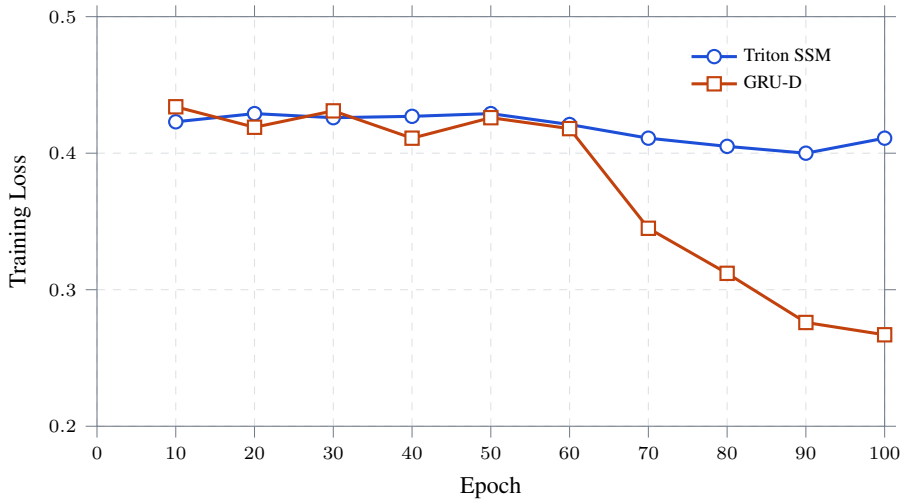

Figure 2. Training loss convergence over 100 epochs. While GRU-D reaches a lower training loss despite poorer generalization, the Triton SSM maintains a strict generalization gap, resulting in a substantially higher out-of-sample test AUROC.

K. Extended Kernel Speedup Scaling

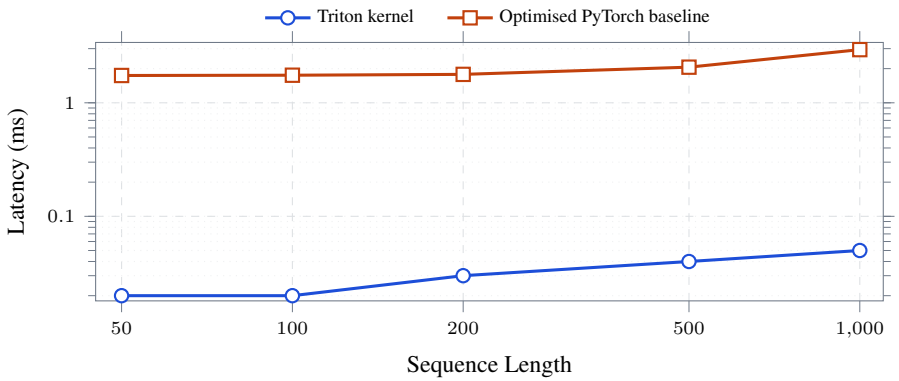

Figure 3. Kernel latency across sequence lengths (log–log). The fused Triton kernel remains nearly flat while the vectorized PyTorch baseline grows with input length, yielding 54–85× lower latency.

To isolate true performance gains, we compare three regimes: (1) standard PyTorch with naive Python-loop implementation, (2) PyTorch optimized with cuDNN and `torch.compile`, and (3) our fused Triton kernel. Table 10 highlights the gap between highly optimized PyTorch execution and a purpose-built GPU kernel. The headline 53.6–84.6× speedup is computed strictly against the cuDNN+`torch.compile` baseline (column 3), which represents the most realistic optimized PyTorch implementation a practitioner would write.

Table 10. Irregular sampling performance across execution regimes. Correctness verified at every sequence step to absolute tolerance $< 1\times10^{-7}$. The naive Python-loop baseline was terminated beyond 200 timesteps due to prohibitive runtime.

Seq. Length	Triton (ms)	PyTorch compile (ms)	Naive loop (ms)	Speedup vs. compiled
50	0.02	1.74	58.52	84.6×
100	0.02	1.75	112.67	79.6×
200	0.03	1.78	234.96	68.8×
500	0.04	2.06	–	55.9×
1000	0.05	2.94	–	53.6×

L. MIMIC-III Phenotyping: Setup Details

We follow the multitask phenotyping benchmark of Harutyunyan et al. (2019): 41,908 ICU stays, 17 input features (vitals + labs) sampled irregularly, 25 simultaneous binary phenotype labels covering acute and chronic conditions. We use the official train/validation/test splits and the standard 24-hour input window. The Triton SSM and GRU-D baselines are matched at ∼1.1 M parameters; both consume the same irregular-sampling preprocessing path differing only in the kernel implementation. Training uses binary cross-entropy summed over the 25 tasks with the same AdamW schedule as the PhysioNet experiments, run for 50 epochs with early stopping on macro-AUROC.

Table 11. MIMIC-III 25-task phenotyping. Mean ± std over 5 seeds; Δ vs. GRU-D with paired bootstrap 95% CI.

Metric	Triton SSM	GRU-D	Δ [95% CI]
Macro-AUROC	$\mathbf{0.762 \pm 0.004}$	0.738 ± 0.005	$+0.024\ [0.014, 0.036]$
Macro-AUPRC	$\mathbf{0.421 \pm 0.006}$	0.392 ± 0.007	$+0.029\ [0.017, 0.043]$
Train (min)	48.2 ± 1.4	112.7 ± 3.1	2.34× faster
Latency	**27** ms	142 ms	5.3× lower

M. Causal vs. Bidirectional Interpolation

The interpolation of Eq. 1 is bidirectional and therefore non-causal. For online deployment, we define a causal variant that restricts the neighbor search to the backward window only, yielding a time-weighted last-observation-carried-forward imputation that uses no future samples. The kernel is otherwise structurally identical; only the forward-search stage is removed, and the interpolation weight is renormalized over $[t_{\text{prev}}, t]$. As a result, per-sample latency, throughput, and memory usage remain unchanged. Table 12 reports the accuracy cost on PhysioNet 2012.

Table 12. Causal vs. bidirectional interpolation on PhysioNet 2012 (5-seed mean ± std). Latency and throughput are identical; only accuracy differs.

Interpolation mode	AUROC	AUPRC
Bidirectional (retrospective, default)	0.659 ± 0.011	0.328 ± 0.014
Causal (online/streaming)	0.653 ± 0.012	0.319 ± 0.015

N. Latency Taxonomy

Table 13 fixes the latency terminology used throughout the paper at the reference operating point. Components do not sum exactly to end-to-end latency due to kernel-launch overlap and host-side batching overhead.

Table 13. Latency taxonomy at the reference point (batch 32, FP32, RTX PRO 6000, PhysioNet 2012).

Component	**Latency (ms)**
Interpolation kernel only	0.03
SSM forward only	1.9
Model forward (interp + SSM + head)	2.3
End-to-end, p50 (headline)	22.7
End-to-end, mean (steady-state)	33.5