Progressive Memory Transformer: Memory-Aware Attention for Time-Series

Tord Sture Stangeland[1,2] tord.stangeland@norsar.no
Andreas Köhler[2,3] andreas.kohler@norsar.no
Steffen Mæland[2,4] steffen.meland@hvl.no
Adín Ramírez Rivera[1] adinr@uio.no
[1]University of Oslo, Department of Informatics, Oslo, Norway
[2]NORSAR, Lillestrøm, Norway
[3]University of Tromsø, Department of Geosciences, Tromsø, Norway
[4]Western Norway University of Applied Sciences, Bergen, Norway

Abstract

Time-series carry structure simultaneously at multiple scales (fine-grained variation, mid-range motifs, and global properties) and downstream tasks operate at correspondingly different scales. Most existing self-supervised learning approaches supervise representations globally via instance-level contrastive losses and limited temporal neighborhood supervision, but do not explicitly exploit the structural hierarchy. We propose a learning framework that explicitly enforces a structural hierarchy across three scales independently: a local objective for token continuity, a mid-range objective for window-level motifs, and a global objective for sequence-level agreement. Realizing this framework requires the backbone to expose a representation at each scale; we introduce **Progressive Memory Transformer** (PMT), which augments a transformer with writable, window-aligned memory that exposes the mid-range scale alongside the token and sequence-level representations conventional transformers already provide. Across seven UCR/UEA/UCI classification benchmarks, a cue-retention probe, and forecasting benchmarks, PMT learns representations that probe well at the global, mid-range, and local scales—strong low-label classification (1–5% labels), competitive forecasting performance across multiple horizons, and quantitative and qualitative evidence that memory states capture mid-range motifs.

1 Introduction

Time-series carry structure simultaneously at multiple scales: fine-grained variation within short windows, recurring patterns at intermediate ranges, and global properties of the whole sequence. Representations that are useful for downstream tasks must capture this structure at every scale where it exists, since downstream tasks (such as local forecasting, motif detection, and sequence-level classification) operate at correspondingly different scales.

Learning these multi-scale representations is not trivial. Conventional self-supervised learning (SSL) for time-series typically relies on global instance-level contrastive losses and limited temporal neighborhood supervision [1–5]—at the readout states shown in Fig. 1. Although these methods provide some temporal alignment, they do not explicitly exploit the underlying structural hierarchy of the signal. Consequently, the intermediate scale (where much of the meaningful temporal structure lives) is often shaped indirectly or destructively compressed (e.g., via max-pooling), rather than explicitly targeted through a dedicated representation interface—shown in gray in Fig. 1. Even

Code: https://github.com/dsb-ifi/pmt

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

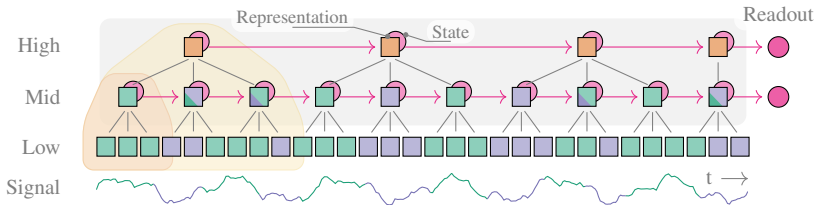

Figure 1: An illustration of the three levels of representation and how they capture distinct structural patterns. Shaded regions indicate expanding receptive fields. Exposing the intermediate representations (gray background) enables direct supervision at multiple scales, unlike standard global readouts.

architectures designed to propagate context across time (e.g., recurrent hidden states or read-only segment caches) treat their intermediate state as an internal optimization device rather than as a representation exposed to direct, scale-appropriate supervision. Furthermore, while recent memory-augmented transformers introduce persistent writable slots, they typically operate as global latent banks decoupled from any local temporal alignment, making them unsuitable for targeted mid-range supervision.

To address this, we propose a learning framework that explicitly enforces a structural hierarchy across three scales independently: token-level (fine-grained), mid-range (intermediate motifs), and sequence-level (global)—cf. Fig. 1. Realizing this framework requires the backbone to expose a representation at each scale. To this end, we introduce the **Progressive Memory Transformer** (PMT), which augments a transformer with a *sample-specific*, writable, window-aligned memory that exposes the mid-range scale alongside the token and sequence-level outputs that conventional transformers already provide. We use contrastive objectives at each scale, chosen to match the granularity of the representation: local within-window continuity at the token level, mid-range motif consistency at the memory level, and sequence-level agreement at the `[CLS]` level.

Contributions. (1) We propose a multi-scale contrastive learning framework for time-series that supervises representation at three scales independently: a token-level Hierarchical Gaussian Contrastive Loss (*HGCL*), a mid-range memory-state loss (*PCL*), and a sequence-level instance loss (*ICL*). (2) We introduce PMT, a memory-augmented transformer backbone whose *window-aligned writable memory* exposes the mid-range scale as a directly supervisable representation, complementing the token and sequence-level outputs of conventional transformers. (3) We evaluate the effectiveness of the resulting representations at three scales: forecasting on standard benchmarks for the local level, cue retention plus qualitative analysis for the mid-range level, and low-label linear-probe classification (1–5% labels) on seven UCR/UEA/UCI benchmarks for the global level, alongside fully supervised validations (Appendix E.1) demonstrating its utility across different tasks.

2 Progressive Memory Transformers

Standard time-series architectures compress temporal dynamics into a single global vector, treating intermediate states as transient variables that are aggregated away. By simply discarding these hidden states (shown in the gray background of Fig. 1), existing methods lose the distinct recurring motifs that emerge at intermediate scales (denoted by the shaded patterns on the representations in the figure). This section instantiates our alternative: a multi-scale framework that explicitly exposes and supervises representations at three distinct resolutions—fine-grained tokens, mid-range motifs, and global summaries. While conventional transformers already expose token and sequence-level outputs, the intermediate representations remain hidden. We introduce **Progressive Memory Transformer (PMT)**, which augments a transformer with writable, window-aligned memory states that expose the mid-range scale as a first-class representation.

By keeping all three levels accessible, we can ground them with scale-matched contrastive objectives to extract useful information at the exact granularity downstream tasks require. We first introduce **Progressive Memory Attention (PMA)** (Section 2.1), the architectural block that produces these mid-range memory states alongside refined tokens. Because these memory states are explicitly exposed, we supervise them directly with the **PMA Contrastive Loss (PCL)** (Section 2.2) to capture robust regional structure. This mid-range objective is complemented by token- and sequence-level contrastive objectives (Section 2.3), and the full pipeline is composed in Section 2.4.

Notation. Let $x \in \mathbb{R}^{L \times C}$ denote a single C-channel time-series of length L. The encoder f_θ produces a matrix $R \in \mathbb{R}^{K \times D}$ containing K *tokens*, each D-dimensional, plus a separate `[CLS]` vector $c \in \mathbb{R}^D$. Hence, the overall output is $\big[R;\, c\big] \in \mathbb{R}^{(K+1) \times D}$.

2.1 Progressive Memory Attention

The core engine of our architecture is the **Progressive Memory Attention (PMA)** operation, illustrated in Fig. 2. Instead of independently processing isolated windows or maintaining a strictly read-only cache, PMA equips each window with a small, fixed-size set of writable memory slots. For a given window w at block b, the PMA operation takes the window's tokens alongside n_m memory slots carried forward from the previous window, producing a refined token stream $W_{w,b}$ and an updated memory state $M_{w,b}$

$$W_{w,b},\, M_{w,b} \;=\; \mathrm{PMA}\big(M_{w-1,b},\, \bar{M}_{w,b-1},\, \bar{W}_{w,b-1}\big)\,. \qquad (1)$$

These inputs encode context along two axes: $M_{w-1,b}$ propagates memory **horizontally** from the previous window at the same block, while $\bar{M}_{w,b-1}$ and $\bar{W}_{w,b-1}$ carry the refined memory and tokens **vertically** from the previous block to the next level. (The bar denotes variables refined by adaptive gates.) Rather than forcing the model to re-attend to all past tokens, this carried memory acts as a compact, evolving summary of historical context.

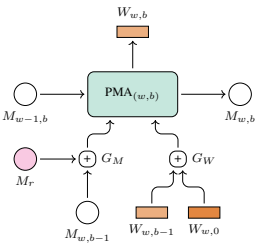

Figure 2: PMA information flow (1).

Concretely, for each window PMA first refines the carried memory by mixing it with a learned reset state, concatenates this refined memory with the previous-block memory and the current window tokens, applies an asymmetric attention mask that keeps memory updates strictly causal, and reads out the updated memory slots alongside the refined window tokens (Fig. B.1 gives the algorithmic form). A single multi-head attention layer is applied jointly to the concatenated sequence; because the slots serve as both queries and keys/values, the attention mechanism can read from and write to them in a single pass.

Two adaptive gates regulate this flow. The first softly mixes the carried memory with a learned reset state (analogous to the forget/reset gates in recurrent architectures), allowing the model to overwrite stale context when temporal regimes change. The second mixes the processed window tokens with their original signal representation, preserving fine-grained local detail while integrating the broader context. Both gates, the attention-mask construction, and the per-window overlap aggregation are detailed in Appendix A.1.

Crucially, these persistent, window-aligned memory slots serve as the **interface** through which the mid-range level becomes explicitly supervisable. Because they are updated in place and exposed at the window scale, we can apply an objective (PCL, Section 2.2) directly to them, rather than indirectly via the sequence summary.

By applying this operation to every window, a **PMA block** propagates memory horizontally across time. Stacking B such blocks into the full PMT backbone propagates information vertically, expanding a token's contextual span linearly in both depth and window index. This dual propagation allows deeper blocks to effectively fuse historical summaries (Appendix A.3 provides a formal receptive-field derivation).

2.2 PMA Contrastive Loss

To ground these memory representations, we apply the **PMA Contrastive Loss (PCL)** (an InfoNCE objective) directly to the memory slots emerging from the final PMA block. By demanding cross-view consistency at the window scale, PCL forces the model to distill salient mid-range motifs into these memory slots such that they are persistent over the stochastic augmentations. Intuitively, each memory slot m functions as a learnable motif detector for its respective window (an individual D-dimensional vector extracted from the final block's memory state M; see Appendix A.4 for its formal definition). Contrasting these individual slots directly ensures that they do not simply cache raw past tokens, but actively compress the window's temporal dynamics into distinct, robust semantic features.

Specifically, each anchor memory slot m_a from one view is paired with its corresponding positive slot m_p from the second view at the same window and slot index. Negatives $m_n \in \mathcal{N}_a$ are drawn

from the memory slots of other sequences in the mini-batch. The loss takes the standard InfoNCE [1] form

$$\mathcal{L}_{\text{PCL}} = -\mathbb{E}_{m_a} \log \frac{\exp(\langle m_a, m_p \rangle / \tau)}{\exp(\langle m_a, m_p \rangle / \tau) + \sum_{m_n \in \mathcal{N}_a} \exp(\langle m_a, m_n \rangle / \tau)}, \tag{2}$$

where $\langle \cdot, \cdot \rangle$ denotes cosine similarity (slots are ℓ_2-normalized) and τ is a temperature parameter. (Full notation, symmetric anchor/key averaging, and implementation details are provided in Appendix A.4.)

2.3 Token and Sequence Objectives

To complete our multi-scale hierarchy, the local (token) and global (sequence) representation levels are supervised by two complementary contrastive objectives. Both follow the standard protocol of projecting representations through a level-specific head prior to the loss.

Hierarchical Gaussian Contrastive Loss (HGCL). HGCL grounds the fine-grained token representations by enforcing local temporal continuity. Because tokens at neighboring positions within the same sequence are intrinsically correlated, HGCL treats nearby in-sample tokens as soft positives (graded by a Gaussian over temporal distance) while treating cross-sequence tokens as hard negatives. Crucially, these Gaussian widths are derived directly from the window size and stride, requiring no pre-computed data-space distances or alignments.

This objective operates hierarchically at two scales: a token-level component, $\mathcal{L}_t$, that softly aggregates neighbors within the same window, and a window-level component, $\mathcal{L}_w$, that aggregates overlapping windows across the sequence. For both, the anchor is matched with its aligned counterpart in the second view, augmented by the soft-positive aggregate, while negatives are drawn from other sequences in the batch. The total loss is their weighted sum

$$\mathcal{L}_{\text{HGCL}} = \alpha \, \mathcal{L}_t + \beta \, \mathcal{L}_w, \tag{3}$$

where α and β trade fine-scale temporal coherence against window-scale consistency. (The bucket-form InfoNCE structure and exact Gaussian formulations are detailed in Appendix A.5.)

Instance Contrastive Loss (ICL). Finally, ICL grounds the global sequence summary using a standard instance-level InfoNCE objective. For a mini-batch of $\mathcal{B}$ sequences with two stochastic views each, let c_i^v denote the `[CLS]` token of the i-th sequence under view v. Each anchor $g(c_i^1)$ pairs with its matched view-2 counterpart $g(c_i^2)$ as the positive, while the remaining $2(\mathcal{B}-1)$ projected `[CLS]` tokens in the batch serve as negatives

$$\mathcal{L}_{\text{ICL}} = -\frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \log \frac{\exp(\langle g(c_i^1), g(c_i^2) \rangle / \tau)}{\sum_{(i',v) \neq (i,1)} \exp(\langle g(c_i^1), g(c_{i'}^v) \rangle / \tau)}, \tag{4}$$

where $\langle \cdot, \cdot \rangle$ denotes cosine similarity and $g(\cdot)$ is the projection head. (The encoder that aggregates the PMA token stream into the sequence summary c is described in Section 2.4; symmetric view averaging and projection-head specifics are in Appendix A.6.)

2.4 End-to-End Learning Framework

The full PMT pipeline integrates the components described above into a single, end-to-end forward pass. (1) A 1-D convolution tokenizes the C-channel waveform into K patch tokens. (2) These tokens are unfolded into overlapping windows of length W and stride S, and processed by a stack of B PMA blocks where memory propagates both horizontally and vertically (Section 2.1). Because windows overlap ($S < W$), an *attentive overlap aggregator* (Appendix A.2) merges overlapping token outputs after each block to maintain a constant sequence length. (3) A short stack of encoder layers refines the resulting token stream using neighborhood-masked attention (restricting each token to a local context window) while a single `[CLS]` token attends globally to aggregate the sequence summary. (4) The network is trained by explicitly grounding the structural hierarchy at its three representation levels: HGCL on the fine-grained tokens, PCL on the mid-range memory slots, and ICL on the global sequence summary (Sections 2.2 and 2.3). These three exposed levels—local tokens, mid-range memory, and global summary—serve directly as the distinct representation interfaces for downstream tasks. The total loss is the weighted sum of these three objectives, i.e.,

$$\mathcal{L}_{\text{total}} = \lambda_{\text{ICL}} \mathcal{L}_{\text{ICL}} + \lambda_{\text{HGCL}} \mathcal{L}_{\text{HGCL}} + \lambda_{\text{PCL}} \mathcal{L}_{\text{PCL}}. \tag{5}$$

Two-view training. Following standard practice in time-series contrastive learning [4, 6, 7], each sample is presented to the weight-shared backbone as two stochastic views. A **weak** view preserves local morphology (via channel-wise z-scaling and low-variance Gaussian noise), while a **strong** view additionally breaks short-range correlations while preserving coarse semantics (via time-warping and magnitude-warping). The three contrastive objectives operate jointly on these paired views, denoted as view 1 and view 2.

Full implementation details (including dataset-specific normalization, attention masking specifics, and the per-block handling of overlapping window outputs) are deferred to Appendix B; Appendix C reports compute and memory cost relative to a FlashAttention baseline.

3 Experiments

We evaluate PMT by probing each of its three representation levels with a downstream task matched to that level's temporal granularity: low-label classification at the global level (Section 3.1), cue retention and qualitative analysis at the mid-range level (Section 3.2), and forecasting at the local level (Section 3.3). We then isolate the architectural contribution from the multi-scale objectives through a matched-architecture comparison (Section 3.4) and quantify loss complementarity in Section 3.5.

We use seven UCR/UEA/UCI benchmarks [8–10]: *HAR*, *Epilepsy*, *Wafer*, *FordA*, *FordB*, *PhalangesOutlinesCorrect (POC)*, and *ElectricDevices*. For classification we follow the linear-evaluation protocol of Eldele et al. [3]: the backbone is pretrained without labels, frozen, and a linear SVM probe matching Lee et al. [5] is trained on 1% and 5% labeled subsets; we report top-1 accuracy and macro-F1 (mF1) on the official test split. Forecasting and cue retention follow the protocols introduced in their respective subsections. We compare against time-series SSL baselines TS2Vec [4], TS-TCC [3], and SoftCLT [5], and against generic SSL baselines adapted to time-series: CPC [1], SimCLR [2], and SSL-ECG [11]. Dataset selection, per-dataset statistics, and additional protocol details are in Appendix D; related work appears in Section 4.

3.1 Global Level

Table 1 presents linear-probe classification results at 1% and 5% labels. On average across all seven datasets and both label fractions, PMT achieves the highest Top-1 accuracy (84.4 vs. 83.1 for TS-TCC+SoftCLT and 82.5 for TS2Vec+SoftCLT) and matches the best macro-F1 score, outperforming baselines in 11 out of 14 accuracy settings. A standout result is on the HAR dataset, where PMT trained on just 1% of the labels surpasses prior methods trained on 5% (92.9 vs. 92.1–92.6).

Table 1: Self-supervised learning results on 1% and 5% labeled subsets. Each cell shows accuracy / macro-F1.

	Self-supervised learning (1% labeled)					
Dataset	SSL-ECG	CPC	SimCLR	TS2Vec+SoftCLT	TS-TCC+SoftCLT	PMT
HAR	60.0 / 54.0	65.4 / 63.8	65.8 / 64.3	91.0 / 91.0	82.9 / 82.8	**92.9 / 93.2**
Epilepsy	89.3 / 86.0	88.9 / 85.8	88.3 / 84.0	96.3 / 94.1	95.6 / **95.6**	**96.6** / 94.7
Wafer	93.4 / 76.1	93.5 / 78.4	93.8 / 78.5	95.3 / 88.1	96.5 / 96.5	**98.7 / 96.5**
FordA	67.9 / 66.2	75.8 / 75.2	55.9 / 55.7	87.1 / 87.1	81.5 / 81.2	**88.2 / 88.2**
FordB	64.4 / 60.5	66.8 / 65.0	50.9 / 49.8	67.9 / 67.9	74.8 / 74.8	**76.3 / 76.2**
POC	62.5 / 41.2	64.8 / 48.2	61.5 / 38.4	63.6 / 62.8	65.4 / 64.6	**68.2 / 65.7**
ElectricDevices	60.1 / 50.0	59.3 / 48.9	62.5 / 51.2	62.0 / 53.0	**64.6 / 63.2**	64.1 / 58.1
	Self-supervised learning (5% labeled)					
Dataset	SSL-ECG	CPC	SimCLR	TS2Vec+SoftCLT	TS-TCC+SoftCLT	PMT
HAR	63.7 / 58.6	75.4 / 74.7	75.8 / 74.9	92.1 / 92.1	92.6 / 92.6	**95.5 / 95.9**
Epilepsy	92.8 / 89.0	92.8 / 90.2	91.3 / 89.2	96.7 / 94.9	96.2 / **96.1**	**96.7** / 94.9
Wafer	94.9 / 84.5	92.5 / 79.4	94.8 / 83.3	98.8 / 96.8	98.2 / **98.2**	**99.0** / 97.2
FordA	73.6 / 70.7	86.5 / 86.5	69.6 / 68.9	92.5 / 92.5	**93.2 / 93.2**	89.5 / 89.5
FordB	71.7 / 69.8	86.3 / 86.2	63.0 / 60.7	78.8 / 78.6	**88.0 / 88.0**	76.7 / 76.7
POC	62.9 / 43.3	66.9 / 44.3	62.7 / 42.4	70.9 / 69.7	69.4 / 66.3	**72.7 / 70.6**
ElectricDevices	63.7 / 56.1	62.4 / 58.1	63.9 / 58.6	62.4 / 54.4	65.1 / **63.8**	**65.9** / 60.8

The engine-vibration benchmarks (FordA/B) present an exception. While PMT leads at 1% labels on both, its performance stagnates at 5% labels. In contrast, autoregressive and predictive-contrastive baselines (CPC, TS-TCC+SoftCLT) gain 13–20pp, whereas PMT and the purely contrastive multi-resolution baseline TS2Vec+SoftCLT remain close to their 1% accuracy. We attribute this to two likely factors: first, the short-lived phase and frequency events characteristic of FordA/B are highly sensitive to the tokenizer's stride (Section 3.6); second, predictive pretraining naturally encodes phase-relative temporal positions, giving it an advantage over the agreement-based contrastive objectives used by PMT. Integrating a predictive component to bridge this gap remains outside of the scope of this work.

3.2 Mid Range

The mid-range probe tests whether the writable memory states function as a representation level rather than an internal optimization device. We design two evaluations: a quantitative cue-retention probe that isolates whether mid-range information is recoverable from the memory states, and a qualitative similarity analysis that inspects what the memory states encode.

Cue retention. We design a new cue-retention experiment to test whether a backbone with mid-range state actually *uses* that state to carry information through a sequence. Strong downstream accuracy does not by itself indicate that an architecture is propagating mid-range information—the task may simply be solvable from local features. To disentangle the confoundment, our protocol injects a synthetic perturbation (a Hann-tapered burst) upstream of the final window into a frozen backbone that was pretrained on FordA without ever seeing the cue. A logistic-regression probe then predicts cue presence strictly from end-of-sequence (EOS) features. Because the model was never trained to detect the cue, success requires that intermediate evidence was actively preserved through state propagation rather than discarded. The EOS feature is the final memory state for PMT, the corresponding recurrent or cache state for similar-architecture baselines (Section 3.4), and windowized embeddings for TS2Vec. Appendix D.3 details the full protocol.

Table 2: FordA cue detection. EOS-probe averages across seeds.

Method	EOS AUC	EOS Top1	EOS mF1
TS2Vec (+SoftCLT)	0.649	0.638	0.604
Transformer-XL	0.933	0.795	0.790
xLSTM	0.956	0.820	0.817
PMT-$\lambda_{\text{PCL}} = 0$	0.983	0.831	0.824
PMT	**0.994**	**0.921**	**0.920**

To apply this protocol to PMT specifically, we further isolate the writable-memory *interface* from the supervision that shapes it. We evaluated an unsupervised-memory variant (PMT-λ_{PCL}=0, identical architecture but PCL disabled during pretraining). The λ_{PCL}=0 row measures the interface alone, and the gap to full PMT recovers PCL's contribution.

TS2Vec, which relies on hierarchical pooling, discards the mid-range pattern and retains the cue poorly (0.649 EOS AUC in Table 2). State-propagating architectures fare much better, and PMT's writable memory outperforms both Transformer-XL and xLSTM even without PCL supervision (0.983 vs. 0.933 and 0.956 AUC). Adding PCL supervision yields a small AUC gain (+0.011) but substantial improvements in threshold-based Top-1 (+0.090) and macro-F1 (+0.096). The small AUC gain indicates that the unsupervised memory already *ranks* cue probabilities well; PCL's additional work is calibration. However, higher Top-1 and mF1 scores correlate with a model that has more refined representations (i.e., they are more discriminative).

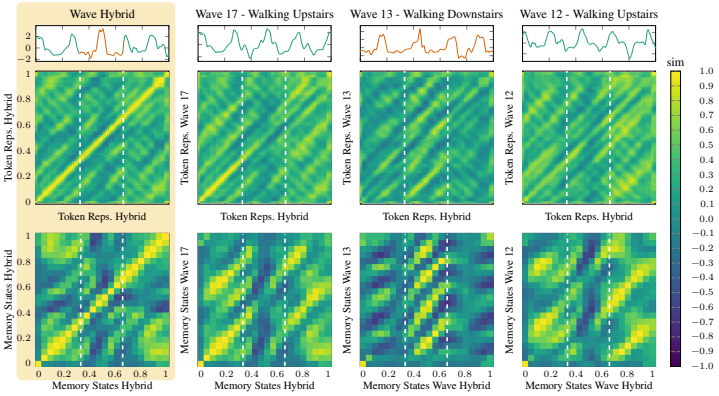

Figure 3: Cosine similarity matrices for the representations and states between pairwise signals from HAR. Higher similarity shows that the signals correlate as evidenced by the learned embeddings. The vertical bars denote the different sections of the hybrid wave. The wave number corresponds to the sample index in the test dataset.

Qualitative analysis. Fig. 3 shows cosine-similarity matrices on a HAR sequence spliced from three activities (walking upstairs, walking downstairs, walking upstairs). The block-diagonal pattern across same-class segments is sharper at the memory-state level than at the token level, with the two upstairs segments mutually bright while the downstairs segment is suppressed. Additional splices and Epilepsy comparisons are in Appendix E.7. Compared to TS2Vec+SoftCLT's intermediate convolutional embeddings, Fig. E.5, PMT's memory states show sharper class-block structure on the same splice.

3.3 Local Level

The local level is supervised by HGCL on per-token representations, with the writable memory propagating context to support predictions extending beyond the current window. We probe these representations through forecasting following the evaluation protocol of Lee et al. [5]: a frozen encoder produces per-timestep features and a per-horizon ridge regressor predicts future values. Table 3 reports MSE and MAE on ETTh1 and Electricity at horizons $\{24, 48, 168, 336, 720\}$ in univariate and multivariate modes, with SoftCLT re-trained under the windowed protocol PMT requires for in-batch negative mining (Appendix D.4). SoftCLT (on the TS2Vec backbone) and PMT both rely solely on contrastive objectives.

Table 3: Forecasting results following SoftCLT evaluation for different horizons (H).

		Univariate				Multivariate			
		SoftCLT		PMT		SoftCLT		PMT	
Dataset	H	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	**0.045**	**0.164**	0.103	0.257	**0.525**	**0.510**	0.546	0.516
	48	**0.075**	**0.212**	0.140	0.302	**0.580**	**0.545**	0.595	0.549
	168	**0.151**	**0.306**	0.184	0.345	0.730	0.634	**0.714**	**0.621**
	336	**0.171**	**0.328**	0.197	0.361	0.884	0.712	**0.834**	**0.688**
	720	0.367	0.535	**0.177**	**0.342**	0.989	0.781	**0.952**	**0.753**
	Avg.	0.162	**0.309**	**0.160**	0.322	0.742	0.636	**0.728**	**0.626**
Electricity	24	0.258	0.281	**0.250**	**0.279**	0.484	0.515	**0.378**	**0.444**
	48	0.311	0.316	**0.298**	**0.310**	0.497	0.521	**0.396**	**0.452**
	168	0.429	0.395	**0.410**	**0.386**	0.512	0.531	**0.411**	**0.460**
	336	0.559	0.472	**0.533**	**0.465**	0.513	0.533	**0.413**	**0.462**
	720	0.860	0.653	**0.849**	**0.646**	0.517	0.538	**0.421**	**0.468**
	Avg.	0.483	0.423	**0.468**	**0.417**	0.505	0.528	**0.404**	**0.457**

PMT achieves lower MSE on all four (dataset, mode) averages and lower MAE on three of four, with the largest margin on multivariate Electricity. The advantage concentrates at longer horizons: PMT wins every $H \geq 168$ comparison on ETTh1 multivariate and every horizon on Electricity, while SoftCLT remains stronger at short horizons on ETTh1 univariate. The horizon-dependent pattern reflects the role of writable, window-aligned memory in the architecture: predictions that span more than a single window can read off the propagated memory states, while short-horizon predictions are largely served by the local token features that both methods compute.

3.4 Architecture Comparison

The probes in Sections 3.1 to 3.3 establish that the multi-scale protocol shapes useful representations, but they leave open how much of that improvement requires the writable, window-aligned memory specifically. To probe this, we replace the PMA stack with two architectural alternatives that propagate context across windows without exposing a window-aligned writable memory interface: an xLSTM stack [12] (recurrent hidden state) and a Transformer-XL backbone [13] (read-only segment cache). Tokenizer, encoder, and SSL training are kept identical to PMT. Because neither alternative exposes a memory interface that PCL can supervise, both baselines are trained with the two protocol losses they can host (HGCL and ICL).

Table 4: Comparison with xLSTM and Transformer-XL (averaged across 1% and 5% labeled data).

	PMT		xLSTM		Transformer-XL	
Dataset	Top-1 Acc	mF1	Top-1 Acc	mF1	Top-1 Acc	mF1
HAR	**94.2**	**94.6**	93.5	93.8	93.0	93.3
Epilepsy	**96.6**	**94.7**	94.5	91.9	94.5	92.0
Wafer	**98.9**	**96.9**	93.4	81.1	95.7	92.0
FordA	**88.9**	**88.9**	88.2	88.2	88.3	88.3
FordB	**76.5**	**76.5**	75.5	75.4	75.6	75.5
POC	**70.5**	**68.2**	65.5	64.8	65.6	61.7
ElectricDevices	**65.0**	**59.5**	55.1	50.7	61.7	55.6
Average	**84.4**	**82.7**	80.8	78.0	82.1	79.8

As shown in Table 4, averaged over the seven benchmarks, PMT attains the strongest Top-1 accuracy (**84.4** vs. 82.1 for Transformer-XL and 80.8 for xLSTM) and macro-F1 (**82.7** vs. 79.8 and 78.0). The matched-architecture baselines remain within 1–3pp on FordA, FordB, and ElectricDevices, and

Table 5: Ablation of the individual loss components and evaluation of their weighting coefficients, averaged over 1% and 5% label splits. Higher is better.

(a) Loss-component ablation: FordA, HAR, POC and Wafer.

Setup	λ_{ICL}	λ_{HGCL}	λ_{PCL}	Top-1 ↑	mF1 ↑
All $\lambda = 1$	✓	✓	✓	87.49	86.06
Leave-one-out					
$\lambda_{\text{ICL}} = 0$	×	✓	✓	83.33	81.23
$\lambda_{\text{HGCL}} = 0$	✓	×	✓	84.69	82.37
$\lambda_{\text{PCL}} = 0$	✓	✓	×	87.33	86.01
Single-only					
ICL only	✓	×	×	81.37	79.59
HGCL only	×	✓	×	82.51	79.17
PCL only	×	×	✓	81.58	79.25

(b) Evaluation of λ coefficients on Wafer and POC.

	HGCL		PCL		ICL	
λ	Top-1 ↑	mF1 ↑	Top-1 ↑	mF1 ↑	Top-1 ↑	mF1 ↑
0	82.64	77.98	83.90	81.24	82.25	78.15
0.01	82.14	78.09	83.92	80.90	82.58	79.21
0.1	**83.81**	**81.43**	**84.08**	**81.27**	82.42	80.12
0.5	83.63	80.25	83.58	80.90	83.67	**81.17**
1	83.58	80.77	83.71	80.77	**83.77**	80.64

Table 6: Evaluation of patch sizes and stride on FordA/B, and ElectricDevices. Training 150 epochs, using the mean 1% and 5% results over 3 seeds.

(a) Keeping a constant stride (9), evaluating *patch sizes* as a fraction of the seq. length.

	FordA		FordB		ElectricDevices	
Patch size	Top-1	mF1	Top-1	mF1	Top-1	mF1
2.5%	82.2	79.0	70.7	66.5	61.2	53.4
5%	85.0	83.9	72.9	70.4	62.1	55.2
7.5%	86.4	85.4	75.5	75.0	63.0	**56.1**
10%	86.7	85.6	75.3	74.7	**63.7**	55.7
12.5%	**87.4**	**86.7**	**75.4**	**75.1**	63.5	55.6
15%	87.4	86.0	74.8	74.1	63.0	55.4

(b) With a constant patch size (62), evaluating *stride* as a fraction of the patch size.

	FordA		FordB		ElectricDevices	
Stride	Top-1	mF1	Top-1	mF1	Top-1	mF1
10%	**87.7**	**87.3**	**75.3**	**74.1**	**63.8**	**56.0**
25%	85.0	83.1	74.8	74.1	61.3	53.7
50%	72.9	65.1	67.2	64.2	60.8	53.6
75%	66.4	56.0	67.6	62.5	60.5	53.0
100%	62.5	57.8	59.5	55.4	58.5	50.5

lose larger margins on Wafer and POC. The pattern is the one the framework predicts: alternative architectures recover part of the protocol's benefit through the losses they can host, but the mid-range level supervised by PCL is not available to them by construction, and the resulting representations carry a measurable gap from PMT's. A short-run architectural ablation on HAR further confirms these structural advantages (cf. Appendix E.3).

3.5 Loss Ablation and Evaluation

Table 5 probes the three losses jointly (5) across 1% and 5% label splits. Removing any single loss from a uniform-weight configuration (λ=1) drops Top-1 accuracy (ICL by 4.2pp, HGCL by 2.8pp, PCL by 0.2pp), and single-loss setups trail by 5–6pp, confirming that all three structural levels are necessary—cf. Table 5(a). Although PCL's leave-one-out drop is small here—since the classification probe reads only the `[CLS]` token, filtering PCL's direct effect on the memory states—its substantial impact is evident when memory is probed directly in the cue-retention experiment (Section 3.2).

Sweeping individual coefficients, Table 5(b), reveals stable performance, with accuracy varying minimally ($<$ 1pp for λ_{PCL}, $<$ 1.7pp for λ_{HGCL}, $<$ 4.1pp for λ_{ICL} (expected: `[CLS]` used for classificaiton)). Overall, uniform weighting (λ=1) lands within $\sim$2pp of optimally tuned setups, establishing it as a strong, deployment-ready default that does not require per-dataset tuning (cf. Appendix E.6).

3.6 Tokenization

Patch length and stride control how the raw waveform is presented to PMT, and we find them to be the most consequential per-dataset choices. We sweep each independently on FordA, FordB, and ElectricDevices, training each configuration for 150 epochs and reporting the mean over 1% and 5% label splits across 3 seeds. As seen in Table 6(a) patch length is well-behaved: across all three datasets, accuracy varies by 1–5pp over a 6× range of patch sizes, with a wide plateau once patches are long enough to cover the relevant motif. Table 6(b) shows stride matters more: moving from heavily overlapping (10% of patch length) to non-overlapping (100%) patches costs 25pp Top-1 on

FordA, 16pp on FordB, and 5pp on ElectricDevices. Small strides are a safe default; patch length is selected within the plateau per dataset. A window-size ablation in Appendix E.5 shows similarly flat behavior for $W \in [0.1K, 0.3K]$, with degradation only as $W \to K$. We discuss limitations regarding tokenizer sensitivity and batch-size requirements in Appendix F.

4 Related Work

Self-supervised learning for time-series. Early contrastive methods such as CPC [1] and SimCLR [2] learn instance-level representations: CPC predicts future latents along the timeline with a probabilistic contrastive loss, while SimCLR treats the entire sequence as a single view and ignores within-series locality. Later work injects explicit temporal structure. TS-TCC [3] couples cross-view future prediction with contextual instance discrimination, and CA-TCC [6] extends it with a class-aware term for semi-supervised settings. TNC [14] contrasts points inside vs. outside a fixed temporal neighborhood; SoftCLT [5] relaxes this with soft positive assignments weighted by a Gaussian over time and a soft-DTW similarity over instances, so phase-shifted segments contribute graded positive signal. TS2Vec [4] enforces contrastive agreement at multiple temporal resolutions, but the resolutions are produced by repeated max-pooling, so each scale is supervised on a compressed token stream. Hierarchical self-supervision has also been pursued in the masked-modeling family: HiMTM [15] uses multi-scale masked reconstruction with self-distillation for forecasting, supervising scales through reconstruction targets rather than contrastive objectives.

These methods either supervise representation at a single scale or shape it indirectly—through hierarchical pooling that aggregates intermediate detail away, or through reconstruction targets in the masked-modeling family. Our framework supervises three scales contrastively with the token stream preserved through the backbone: HGCL at the token level, PCL at the memory level, and ICL at the sequence level. Unlike SoftCLT-style soft positives, HGCL derives graded positives from window geometry alone, without data-space distance or DTW alignment, and keeps the negative pool disjoint from the soft-positive pool. The intermediate scale is shaped by a contrastive signal applied to first-class memory representations, rather than to surrogates produced by pooling or masked reconstruction.

Memory-augmented transformers. Vanilla transformers are stateless: once a window is processed, the past must be re-read. Transformer-XL [13] extends context by caching the previous segment's hidden activations and concatenating them to the next, providing a longer horizontal context that is strictly read-only. Compressive Transformer [16] extends this lineage by additionally compressing older cached activations into a coarser memory, retaining the read-only character. Set Transformer [17] and Perceiver [18] introduce learnable latent tokens that travel vertically through layers as a fixed-size bottleneck and reset for every sequence, never carrying sample-specific state across windows. Sequence Complementor [19] appends a small number of learnable complementary sequences to a patchified encoder; these tokens are shared across samples and discarded after the encoder, not maintaining sequence-specific state. Titans [20] introduces writable persistent slots that survive across segments, but exposes them as a single global bank decoupled from any sliding-window alignment, so the memory cannot be addressed at any specific local temporal scale.

None of these architectures provide a state interface that is simultaneously sample-specific, window-aligned, writable, and exposed at a temporally meaningful scale that a learning signal can address directly. PMT fills this gap: each window carries a writable, sample-specific memory state aligned to a sliding window, supervised directly by PCL as a first-class representation.

Hierarchical and long-context transformers for time-series. A separate line of work tackles long-context modeling for time-series through architectural choices on the encoder rather than through memory. PatchTST [21] discards memory entirely, using patch tokens and vanilla self-attention so distant dependencies are supplied through an increasingly long look-back window. Pyraformer [22] introduces pyramidal attention with multi-scale temporal nodes connected by inter- and intra-scale edges, enabling long-range modeling at low complexity. Rough Transformers [23] compress sequences via fixed signature-based tokenization, providing continuous-time inductive bias with a lightweight token sequence.

These architectures expose temporal structure at multiple scales but do so through fixed compression or pyramidal aggregation that produces read-only intermediate nodes, not representations that a contrastive learning signal can supervise independently. PMT instead exposes the mid-range scale as

a writable memory state that is supervised contrastively at the same scale, complementing rather than replacing the token and sequence-level outputs already provided by conventional transformers.

5 Conclusion

We introduced a multi-scale contrastive framework supervising time-series representations at three distinct scales: token, mid-range, and sequence-level. To realize this, we proposed the **Progressive Memory Transformer (PMT)**, augmenting conventional transformers with a sample-specific, window-aligned memory that exposes the mid-range scale as a first-class representation. PMT probes well at every targeted level: it achieves state-of-the-art low-label classification (global), the cleanest cue retention among state-propagating architectures with highly calibrated features (mid-range), and lower forecasting error than matched contrastive baselines (local). Finally, ablations isolate the unique architectural contribution of the writable mid-range interface and confirm the complementarity of the three objectives, establishing uniform loss weighting as a stable, deployment-ready default.

Acknowledgments

This work was funded by the Research Council of Norway through Visual Intelligence, Centre for Research-based Innovation (309439), and by AURORA project (359216). The computations were performed on resources provided by Sigma2 (NN8104K)—the National Infrastructure for High-Performance Computing and Data Storage in Norway. We acknowledge Sigma2 for access to the LUMI supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CSC (Finland) and the LUMI consortium through Sigma2, Norway.

References

[1] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Inter. Conf. Mach. Learn. (ICML)*, pages 1597–1607. PMLR, 2020.

[3] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. Time-series representation learning via temporal and contextual contrasting. In *Inter. Joint Conf. Artif. Intell. (IJCAI)*, pages 2352–2359, 2021.

[4] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. TS2Vec: Towards universal representation of time series. In *AAAI Conf. Artif. Intell. (AAAI)*, 2022. URL https://arxiv.org/abs/2106.10466.

[5] Seunghan Lee, Taeyoung Park, and Kibok Lee. Soft contrastive learning for time series. In *Inter. Conf. Learn. Represent. (ICLR)*, 2024. URL https://openreview.net/forum?id=pAsQSWlDUf.

[6] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee-Keong Kwoh, Xiaoli Li, and Cuntai Guan. Self-supervised contrastive representation learning for semi-supervised time-series classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(12):15604–15618, December 2023. ISSN 1939-3539. doi: 10.1109/tpami.2023.3308189. URL http://dx.doi.org/10.1109/TPAMI.2023.3308189.

[7] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee-Keong Kwoh, and Xiaoli Li. Label-efficient time series representation learning: A review. *IEEE Trans. Artif. Intell.*, 5(12):6027–6042, December 2024. ISSN 2691-4581. doi: 10.1109/tai.2024.3430236. URL http://dx.doi.org/10.1109/TAI.2024.3430236.

[8] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. The UCR time series classification archive, October 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[9] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The UEA multivariate time series classification archive, 2018, 2018. URL https://arxiv.org/abs/1811.00075.

[10] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. A public domain dataset for human activity recognition using smartphones. In *ESANN Eur. Symp. Artif. Neural Netw.*, pages 3–4, 2013.

[11] Pritam Sarkar and Ali Etemad. Self-supervised learning for ecg-based emotion recognition. In *IEEE Inter. Conf. Acoust., Speech, Signal Process. (ICASSP)*. IEEE, May 2020. doi: 10.1109/icassp40776.2020.9053985. URL http://dx.doi.org/10.1109/ICASSP40776.2020.9053985.

[12] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended long short-term memory. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, 2024.

[13] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Conf. Assoc. Comput. Ling. (ACL)*, 2019.

[14] Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. Unsupervised representation learning for time series with temporal neighborhood coding. In *Inter. Conf. Learn. Represent. (ICLR)*, 2021. URL https://openreview.net/forum?id=8qDwejCuCN.

[15] Shubao Zhao, Ming Jin, Zhaoxiang Hou, Chengyi Yang, Zengxiang Li, Qingsong Wen, and Yi Wang. HiMTM: Hierarchical multi-scale masked time series modeling with self-distillation for long-term forecasting. In *ACM Inter. Conf. Inf. Knowl. Manag. (CIKM)*, 2024. doi: 10.1145/3627673.3679741.

[16] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *Inter. Conf. Learn. Represent. (ICLR)*, 2020. URL https://openreview.net/forum?id=SylKikSYDH.

[17] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Inter. Conf. Mach. Learn. (ICML)*, pages 3744–3753, 2019.

[18] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *Inter. Conf. Learn. Represent. (ICLR)*, pages 4651–4664. PMLR, 2021.

[19] Xiwen Chen, Peijie Qiu, Wenhui Zhu, Huayu Li, Hao Wang, Aristeidis Sotiras, Yalin Wang, and Abolfazl Razi. Sequence complementor: Complementing transformers for time series forecasting with learnable sequences. In *AAAI Conf. Artif. Intell. (AAAI)*, volume 39, pages 15913–15921, Apr. 2025. doi: 10.1609/aaai.v39i15.33747. URL https://ojs.aaai.org/index.php/AAAI/article/view/33747.

[20] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, 2025. URL https://openreview.net/forum?id=8GjSf9Rh7Z.

[21] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *Inter. Conf. Learn. Represent. (ICLR)*, 2023.

[22] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X. Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *Inter. Conf. Learn. Represent. (ICLR)*, 2022. URL https://openreview.net/forum?id=0EXmFzUn5I.

[23] Fernando Moreno-Pino, Álvaro Arroyo, Harrison Waldon, Xiaowen Dong, and Álvaro Cartea. Rough transformers: Lightweight and continuous time series modelling through signature patching. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, 2024. URL https://openreview.net/forum?id=gXWmhzeVmh.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9: 1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735.

[25] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *Inter. Conf. Learn. Represent. (ICLR)*, 2024.

[26] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Inter. Conf. Learn. Represent. (ICLR)*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[27] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Inter. Conf. Learn. Represent. (ICLR)*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

[28] Ralph G Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Physical Review E*, 64(6):061907, 2001.

[29] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2020.

[30] Zhirong Wu, Yuanjun Xiong, Stella Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2018.

A PMT Details

A.1 PMA Details

This appendix gives the operational details of the PMA block summarized in Section 2.1: the gating functions, attention masking, overlap aggregation, and receptive-field growth.

The memory state propagates *horizontally* to the next window and *vertically* up the stack, so the receptive field widens by the stride S as the window advances while deeper blocks fuse these compact summaries without re-encoding all tokens, see Figs. A.1 and 2 and Eq. (1).

For a given window w at a block b, we compute window $W_{w,b}$ and memory state $M_{w,b}$ representations through our proposed PMA block, such that

$$W_{w,b}, M_{w,b} = \text{PMA}(M_{w-1,b}, \bar{M}_{w,b-1}, \bar{W}_{w,b-1}), \tag{A.1}$$

where $M_{w-1,b}$ provides the *temporal* context (memory state from the previous window at the same block), and $\bar{M}_{w,b-1}$ and $\bar{W}_{w,b-1}$ provide *hierarchical* context (refined memory state and window tokens from the previous block). The initial memory state for block b is a *reset state* used for initialization and adaptive reset, i.e., $M_{0,b} = M_r$. We refine the memory state at a given window and block with a learnable gate that mixes the carry-over memory state and the reset state (implemented as a pre-update mix that forms $\bar{M}_{w,b-1}$)

$$\bar{M}_{w,b-1} = G_M(M_{w,b-1}, M_r), \tag{A.2}$$

that adaptively mixes the memory state with the reset state M_r, enabling the model to overwrite stale context and prevent drift when regimes change. This is reminiscent of forget/reset gates in LSTMs [24], but applied to window-level memory slots updated via attention rather than to per-time-step hidden states. Similarly, we refine the representations with an adaptive gating function,

$$\bar{W}_{w,b-1} = G_W(W_{w,b-1}, W_{w,0}), \tag{A.3}$$

that mixes the original signal representation, $W_{w,0} = f_\theta(x; w)$, with the processed one $W_{w,b}$ to preserve local detail while integrating context. We illustrate this block in Fig. 2. Within each PMA block we apply one multi-head FlashAttention-2 [25] layer to the concatenated input sequence (1). Memory slot queries are unmasked; window queries see every key in $M_{w-1,b}$ and the strictly-causal slice of their own window but are blocked from $\bar{M}_{w,b-1}$ preventing leakage from deeper (potentially future-aware) summaries and keeping updates strictly autoregressive. We perform this computation (1) for every window and block in a forward manner.

In our implementation, we normalize the representations layer-wise. Moreover, we enforce these local and causal constraints by applying custom block-diagonal attention masks via FlashAttention-2 [25].

Attentive overlap aggregation & residual path. Because windows overlap, the same position appears O times per block; an overlap aggregator merges these rows so the token length stays constant across blocks. The aggregator is a single-query cross-attention per position over its O overlapped patches (keys/values from the overlaps) with a learned head$\times$overlap bias. Its output is scaled by $1/\sqrt{O}$, post-normalized, and multiplied by a learnable γ, then fused with the masked-mean skip via a SkipGate before re-entering the residual stream. Standard pre-norm skips wrap the chunk-processing and feed-forward sub-layers. Additional gates control state reuse across blocks.

A.2 Attentive Overlap Aggregator

For global position t with overlapped embeddings $\{W_t^{(r)}\}_{r=1}^{O_t}$, the aggregator uses a *single query per position* to attend over overlaps and produce

$$A_t = \gamma \, \text{LN}\Big(O^{-1/2} \, \text{Concat}_{h=1}^{H} \sum_{r=1}^{O_t} \alpha_{h,r} \, V_{h,r}\Big), \tag{A.4}$$

$$\alpha_{h,r} = \text{softmax}_r\Big(\tfrac{\langle q_h, K_{h,r}\rangle}{\sqrt{d_h}} + b_{h,r}\Big). \tag{A.5}$$

Here $K_{h,r}, V_{h,r}$ are per-head projections of the overlapped $W_t^{(r)}$, q_h is formed by depthwise Conv1D across overlaps, mean, then a 2-layer MLP, $b_{h,r}$ is a learned head $\times$ overlap bias, and there is *no*

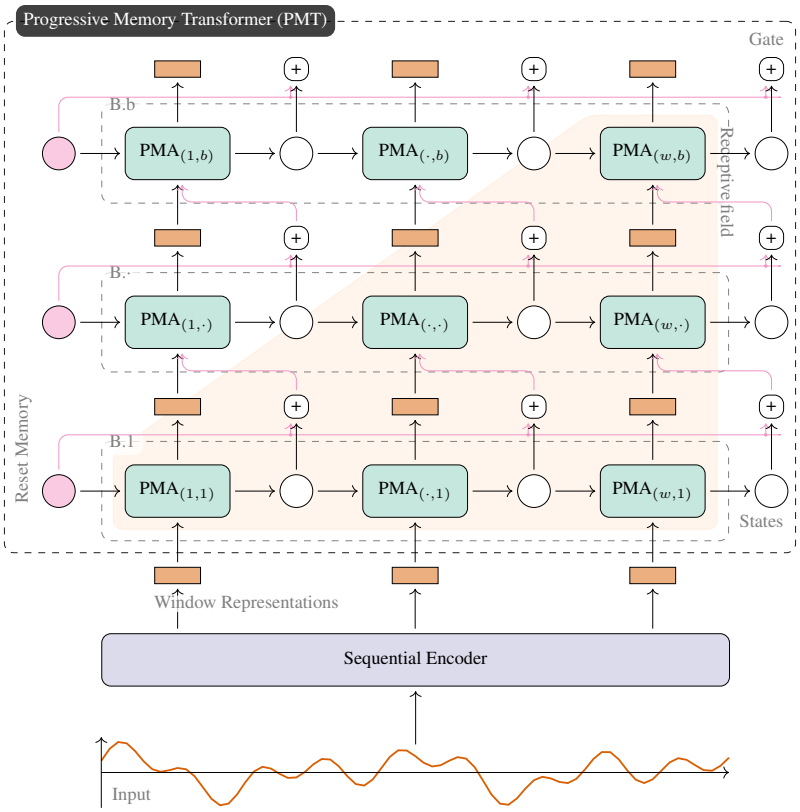

Figure A.1: An illustration of PMT unrolling several PMA blocks through time (left to right) and hierarchy levels (bottom to top). The sequence is tokenized and encoded through an encoder. Then, each PMA block uses these representations and previous states to produce a new version of both of these inputs. At each level, a given PMA block uses a gating mechanism to control how much of the previous level's memory state is passed forward or reset (i.e., uses a reset memory state instead).

output projection after head concatenation. The skip path is the masked mean

$$S_t = \tfrac{1}{O_t} \sum_{r=1}^{O_t} W_t^{(r)}, \tag{A.6}$$

and the block output mixes them via a learnable SkipGate

$$\widehat{W}_t = G_{\text{skip}}(S_t,\, A_t). \tag{A.7}$$

(Streaming computes the same weighted sum without materializing $[\mathcal{B}, K, O, D]$ and uses a tile-constant O for the $O^{-1/2}$ scaling.)

A.3 Receptive-field growth in Progressive Memory Attention

Notation. Let W be the *window length* (tokens per window), S the *stride* ($1 \le S \le W$), B the number of stacked PMA blocks, K the total number of input tokens, and $w \in \{1, 2, \ldots, N\}$ the window index, which covers tokens $x^{\text{token}}_{(w-1)S:(w-1)S+W-1}$. Define

$$\mathcal{R}_w^{(b)} \;=\; \text{all input tokens that can influence } \textit{any} \text{ token in window } w \text{ after block } b,$$

so that $|\mathcal{R}_w^{(b)}| \le K$ by construction. The bounds below report the *nominal receptive-field span* induced by horizontal memory and overlap-based propagation; the actual cardinality is upper-bounded by these expressions and capped by K. These bounds describe the forward history available under the PMA masking used in our experiments; if non-causal overlap aggregation is enabled, the spans should be interpreted as one-sided history bounds rather than full bidirectional receptive fields.

Claim 1 (base case: a window in isolation)**.** *A single window processed in isolation—without horizontal memory propagated from earlier windows—has receptive field*

$$\left|\mathcal{R}_w^{(1)}\right|_{\textit{isolated}} = W, \quad \mathcal{R}_w^{(1)}\big|_{\textit{isolated}} = \big[(w-1)S, (w-1)S + W - 1\big]. \tag{A.8}$$

This is the receptive field consumed by the window encoder before any cross-window propagation. The two regimes below extend this base case differently.

Claim 2 (without horizontal memory)**.** *If the horizontal memory bank is* disabled*, only overlap-based propagation across blocks expands the receptive field; each extra block contributes at most $W - S$ additional tokens, capped by available preceding tokens,*

$$\boxed{\left|\mathcal{R}_w^{(b)}\right|_{\textit{no mem}} \;\le\; \min\big\{K,\; W + \min\big((b-1)(W-S),\; (w-1)S\big)\big\}} \tag{A.9}$$

which is consistent with Claim 1 at $b = 1$.

Claim 3 (with horizontal memory)**.** *With the memory bank active, the horizontal carry $M_{w-1,b}$ allows information from preceding windows to influence window w already in the first block; subsequent blocks contribute at most $W - S$ additional tokens of nominal span via overlap propagation,*

$$\boxed{\left|\mathcal{R}_w^{(b)}\right|_{\textit{mem}} \;\le\; \min\big\{K,\; W + (w-1)S + (b-1)(W-S)\big\}} \tag{A.10}$$

At $b = 1$ the bound becomes $\min\{K, W + (w-1)S\}$, reflecting the predecessor coverage carried in via memory; Claim 1 is recovered only at $w = 1$.

FordA-sized illustration. Consider a token-sequence of length $K = 500$ with $W = 100$, $S = 25$, and $B = 4$. The number of valid (non-padded) windows is $N = \lfloor (K - W)/S \rfloor + 1 = 17$. For the last window ($w = 17$), the nominal span at block 1 from (A.10) is already

$$W + (w-1)S \;=\; 100 + 16 \times 25 \;=\; 500,$$

so the final memory state can in principle be influenced by the whole sequence through horizontal propagation alone. With $B = 4$, the nominal span becomes

$$W + (w-1)S + (B-1)(W-S) \;=\; 100 + 16 \times 25 + 3 \times 75 \;=\; 725,$$

which is capped by K:

$$|\mathcal{R}_{17}^{(4)}|_{\text{mem}} \;\le\; \min\{500,\, 725\} \;=\; 500.$$

For the EOS window the additional blocks therefore refine and recompose full-sequence information rather than enlarging the receptive set; depth contributes new uncovered tokens only for windows where the block-1 span has not yet saturated.

A.4 Details on PCL

PCL is applied to the memory slots produced by the final PMA block only. Let $\mathcal{B}$ be the batch size, $v \in \{1, 2\}$ the view index, and $w = 1, \ldots, N$ the window index. The memory state of the i-th sample at window w under view v is $M_{i,w}^{(v)} \in \mathbb{R}^{n_m \times D}$, with n_m slots per window. For an anchor $m_a = M_{i,w}^{(1)}[k]$, the positive is $m_p = M_{i,w}^{(2)}[k]$, and the loss averages over all anchor slots with normalization $1/(\mathcal{B}N n_m)$. The negative pool $\mathcal{N}_a$ excludes all slots from the same source sequence as the anchor; we sample $n_{\text{neg}} = 512$ negatives per anchor.

Symmetry is enforced by computing the loss in both anchor/key directions and averaging: $\mathcal{L}_{\text{PCL}} = \frac{1}{2}(\mathcal{L}_{\text{PCL}}^{1\to 2} + \mathcal{L}_{\text{PCL}}^{2\to 1})$. The slots are ℓ_2-normalized and used directly as contrastive embeddings; no projection head is applied. The temperature τ follows a linear schedule from a starting value to an ending value over the training run; specific start/end temperatures are dataset-dependent and reported with the per-dataset hyperparameters in our released code. The loss is computed in chunks over anchors to bound activation memory, with a custom autograd path that stores compact backward statistics rather than retaining all sampled negatives.

A.5 Details on HGCL

HGCL applies a bucket-form InfoNCE objective at two scales—token-within-window and window-across-sequence. Both scales operate on the same projected representations $z^{(v)} \in \mathbb{R}^{\mathcal{B}\times K\times D}$ (view $v \in \{1, 2\}$, K tokens after patchification), unfolded into windows of length W and stride S.

Bucket objective. For an anchor a with aligned positive p, kernel-weighted soft-positive aggregate $T_a = \sum_c q_{a,c}\, s_{a,c}$ over local candidates c, and negative bank $\mathcal{N}_a$

$$\mathcal{L}_a = -\log \frac{\exp(s_{a,p}/\tau)}{\exp(s_{a,p}/\tau) + \exp((T_a - s_{a,p})/\tau) + \sum_{n\in\mathcal{N}_a} \exp(s_{a,n}/\tau)}, \tag{A.11}$$

where $s_{a,c} = \langle z_a, z_c\rangle$ is cosine similarity and τ is a level-specific temperature.

Conventions. The kernel weights $q_{a,c}$ are unnormalized Gaussian values; $q_{a,p} = 1$ at the aligned candidate, and T_a sums over all local candidates including p, so $T_a - s_{a,p} = \sum_{c\neq p} q_{a,c}\, s_{a,c}$. Subtracting $s_{a,p}$ keeps the aligned positive in the numerator without double-counting in the denominator.

Token level. For an anchor token at position k in window w of view 1, the aligned positive is the matched token at the same position in view 2. Local candidates are the W tokens of the same window in view 2, with Gaussian weights

$$q^t_{k,j} = \exp\left(-\frac{(k-j)^2}{2\sigma_p^2}\right), \qquad \sigma_p \propto W. \tag{A.12}$$

Negatives are sampled from tokens of other sequences in the global batch.

Window level. Window representations are the ℓ_2-normalized mean of their tokens. For an anchor window at position u in view 1, the aligned positive is the matched window in view 2. Local candidates are the windows of the same sequence in view 2, with Gaussian weights

$$q^w_{u,u'} = \exp\left(-\frac{((u-u')\cdot S)^2}{2\sigma_w^2}\right), \qquad \sigma_w \propto W \cdot \frac{W-S}{W}. \tag{A.13}$$

Negatives are sampled from windows of other sequences in the global batch.

Combined loss. The total HGCL loss combines the two scales

$$\mathcal{L}_{\text{HGCL}} = \alpha\, \overline{\mathcal{L}_t} + \beta\, \overline{\mathcal{L}_w}, \tag{A.14}$$

where $\overline{\cdot}$ denotes mean over valid anchors.

A.6 Details on ICL

The loss in Eq. (4) is computed symmetrically over view assignments

$$\mathcal{L}_{\text{ICL}} = \tfrac{1}{2}\left(\mathcal{L}^{1\to 2}_{\text{ICL}} + \mathcal{L}^{2\to 1}_{\text{ICL}}\right). \tag{A.15}$$

The projection head $g(\cdot)$ is a two-layer MLP with GELU activation, applied per view.

B Implementation Details

We train the PMT using an AdamW [26] optimizer with a cosine annealing learning rate [27] scheduler, with a warmup period of 5% to a peak of $1e-4$ and a minimum learning rate of $1e-6$. Models were trained with a batch size of 256. The models were trained on Nvidia GH200 GPUs. For dataset specific hyperparameters used for Table 1, we refer the reader to the attached repository (link on page 1).

The cosine similarity figures were all created using per-dataset specific frozen backbone. For both the HAR and Epilepsy figures, the backbone consisted of 6 PMA blocks with a window size of 6 and stride 3, using 2 memory states. Commonly for all visualizations, we do a simple forward pass through the model and extract the output token representations and the final PMA block's memory states. We use the mean per window memory state for the visualizations. Both the averaged memory state and the output tokens are ℓ_2-normalized.

Algorithm B.1: Forward pass through a B-block PMA stack

Input: patch tokens $W_{1:N,0}$, random memory M_r
Output: token stream $W_{1:N,B}$, memories $M_{1:N,B}$

for $b \leftarrow 1$ **to** B **do**
  $M_{0,b} \leftarrow M_r$;
  for $w \leftarrow 1$ **to** N **do**
    $\bar{W}_{w,b-1} \leftarrow G_W\big(W_{w,b-1}, W_{w,0}\big)$; $\bar{M}_{w,b-1} \leftarrow G_M\big(\mathrm{LN}(M_{w,b-1}), \mathrm{LN}(M_r)\big)$;
    $\big[M_{w,b}, W^{\mathrm{out}}_{w,b}\big] \leftarrow \mathrm{SelfAttn}\big([M_{w-1,b} \| \bar{M}_{w,b-1} \| \bar{W}_{w,b-1}]\big)$;
  $W_{w,b} \leftarrow \mathrm{OverlapPool}\big(\{W^{\mathrm{out}}_{w',b}\}_{w'=1}^{N}\big)$ $(\forall w)$

- G_W — mixes new patch evidence with prior tokens.
- G_M — gate blending inherited memory with M_r.
- $\mathrm{SelfAttn}$ — masked attention over $[M_{w-1,b} \| \bar{M}_{w,b-1} \| \bar{W}_{w,b-1}]$.
- $\mathrm{OverlapPool}$ — attentive overlap aggregator producing $W_{w,b}$.

Figure B.1: Forward pass through a stack of B PMA blocks: per-window, per-block memory and token updates with adaptive gates and asymmetric attention masking.

The scatter plots, such as Fig. E.7 use PCA for dimensionality reduction per token and per memory state. Heat-maps, such as Figs. 3 and E.2 use the cosine-similarity matrix for both the token and memory state visualizations.

Patchified input. Our sequence encoder is a 1-D convolution with kernel k and stride k first tokenizes the waveform into fixed-length patch embeddings; these tokens—not the raw samples—form the input to every PMA block. Although we restrict ourselves to this patchified view in the present work, extending PMA to operate directly on raw time steps is a promising avenue for future research.

Overlap-aware processing. We unfold the patch stream into windows of length W and stride S (overlap $O = \lceil W/S \rceil$). Each window passes through a single-layer Transformer with the asymmetric mask described in Appendix A.1; FlashAttention-2 reduces its space requirement to $\mathcal{O}\big((|M| + S)D\big)$ per window. After all $N = \lceil K/S \rceil$ windows are processed, the attentive overlap aggregator merges the O overlapping rows at every position. Without this overlap aggregator each subsequent PMA block would receive a growing number of tokens due to $W > S$. The overlap aggregator ensures the input and output shape of any PMA block remains equal.

Global aggregation. Finally, we append the `[CLS]` token to the output from the PMA blocks, and pass the tokens through a series of encoder layers with neighborhood masking. Specifically, a patch token at position t attends only to its local context window (positions $t - n, \ldots, t$), whereas the `[CLS]` token attends to every position to aggregate the global representation.

C Computational Analysis

We report minimal, reproducible compute measurements for completeness. These results detail compute and resource notes already in Appendix B and use the same backbone as the experiments. The goal is to characterize the overhead of PMA relative to a vanilla transformer with FlashAttention in the *same* implementation, rather than to claim PMT is more efficient than long-context architectures such as Transformer-XL [13] or patch-based forecasters like PatchTST [21]. We distinguish between a *streaming* PMA microbenchmark (single-window kernel; no re-encoding the past) and a *vectorized* non-streaming configuration (overlapped windows materialized for speed during training).

C.1 Hardware and Setup

The main paper's experiments were trained on NVIDIA GH200 (∼95 GB) GPUs—typically 2 GPUs via DDP, with 8 dataloader workers per GPU. A full 250-epoch experiment under this configuration takes roughly 1.5 hours. The streaming and vectorized benchmarks reported in Appendices C.2 to C.4 were run separately on a single NVIDIA A100 (80 GB), FP16, PyTorch with FlashAttention-2 for attention kernels; we did not re-run them on more recent hardware, since these benchmarks characterize PMA's intrinsic cost relative to FlashAttention rather than absolute throughput on current hardware. Peak GPU memory is measured via `torch.cuda.max_memory_allocated()` after `cudaDeviceSynchronize()`; on shorter sequences, PyTorch's caching allocator may report

Table C.1: Streaming PMA: single-window kernel timed; FA run on the full sequence. For long horizons, PMA reduces both FLOPs and peak memory while sustaining 96 kHz real-time with a 20% head-room.

Sequence (samples → tokens)	PMA window ms (seq s)	FA full seq s	Max SR [Hz] ↑	Δ FLOPs (PMA/FA)		Δ mem (PMA/FA)	
512 → 63	0.8 (0.01)	0.000	48,000	+95%	(0.16/0.08 G)	0%	(26/26 MB)
1,024 → 127	0.8 (0.01)	0.000	48,000	+82%	(0.30/0.17 G)	0%	(26/26 MB)
2,048 → 255	0.8 (0.01)	0.000	96,000	+70%	(0.60/0.35 G)	0%	(26/26 MB)
4,096 → 511	0.8 (0.01)	0.000	96,000	+53%	(1.21/0.80 G)	0%	(28/28 MB)
8,192 → 1,023	0.8 (0.01)	0.000	96,000	+31%	(2.52/1.93 G)	−36%	(28/44 MB)
16,384 → 2,047	0.8 (0.01)	0.001	96,000	+6%	(5.52/5.20 G)	−68%	(30/94 MB)
32,768 → 4,095	0.8 (0.01)	0.003	96,000	−18%	(13.0/15.8 G)	−83%	(52/308 MB)
65,536 → 8,191	0.8 (0.01)	0.008	96,000	−37%	(33.6/53.0 G)	−95%	(52/1,092 MB)
131,072 → 16,383	1.1 (0.01)	0.030	96,000	−49%	(98.1/191.9 G)	−97%	(122/4,262 MB)

Table C.2: Non-streaming inference (vectorized). Vectorization duplicates overlapped windows for speed, inflating peak memory; the attentive overlap aggregator removes duplicates post-block.

Model	Params [M]	GFLOPs	Peak mem [MB]	Latency [ms] ↓	Tokens/s ↑
Vanilla Transformer	7.4	692.34	8,485	4,619.13	86,590
FlashAttention (FA)	7.4	692.34	8,336	3,476.55	115,047
PMA (vectorized)	16.3	329.95	12,292	2,962.21	135,022
PMT (full)	20.1	329.99	12,337	5,381.27	74,326

the same peak for different models because blocks are reserved in advance, and we report the measured peak in all cases.

C.2 Streaming PMA microbenchmarks

We time a single PMA block with $D = 320$ (as in the main experiments), a 16 sample-convolutional patchifier (8× compression), window stride $S = 0.5\,W$, and one memory slot. For comparability, FlashAttention (FA) is run on the full sequence (global receptive field). We report per-window latency (ms), total sequence latency (s) implied by sliding over the sequence, maximum sustained sampling rate (Hz) with a 20% head-room, and relative FLOPs/memory versus FA (rounded).

Table C.1 shows that for short sequences the writable memory trades extra compute for fixed per-step latency, but as the sequence grows the FA baseline's global attention dominates. Streaming PMA keeps the per-step memory bounded by $O((|M| + S)D)$ (Appendix B), which avoids the growth of full-sequence attention.

C.3 Vectorized non-streaming inference

For training speed, we also report a non-streaming vectorized configuration that materializes all overlapped windows before the attentive overlap aggregator. All models use $D = 320$, 6 blocks (PMT includes two neighborhood encoders for `[CLS]`), 100,000 time steps (8× tokenization), $W = 0.1K$, $S = 0.5W$, one memory slot. Latency is end-to-end.

Notes and caveats. (i) FA sees the full sequence at once whereas streaming PMA strictly limits the receptive field to the current window plus memory slots; this explains FA's lower latency on short sequences and PMA's memory advantage on long horizons. (ii) The vectorized PMA/PMT inflate peak memory due to overlapped materialization; the streaming kernel avoids this by construction. (iii) Reported Δ values use rounded base numbers; minor rounding mismatch may occur.

C.4 Attentive Overlap Aggregator Cost

To understand the computational distribution within PMA blocks, we isolate and measure the *attentive overlap aggregator* component in both streaming and vectorized configurations.

Methodology. We measure the aggregator in isolation by providing pre-computed window embeddings, thus excluding the window encoder (PMA attention) costs. Setup matches Appendix C.2, with one memory slot.

Results. Tables C.3 to C.5 show that the overlap aggregator accounts for only 1–3% of per-window processing time in streaming mode and less than 1% in vectorized mode. The aggregator's computa-

Table C.3: Streaming overlap aggregator: isolated component analysis showing minimal overhead.

W (tokens)	S/W	O	d	GFLOPs/step	Time share [%] ↓
1024	0.5	2	320	0.24	3.1
2048	0.5	2	320	0.48	1.7
4096	0.5	2	320	0.96	1.1

Table C.4: Vectorized overlap aggregator: computational cost scales linearly with sequence length.

L (tokens)	W	S/W	K	d	GFLOPs	MFLOPs/token
25,000	2,500	0.5	2	320	11.7	0.47
50,000	5,000	0.5	2	320	23.4	0.47
100,000	10,000	0.5	2	320	46.8	0.47

tional cost is dominated by key-value projections (87% of aggregator FLOPs), while the attention mechanism itself requires minimal computation due to single-token queries over $O{=}2$ positions.

Implementation notes. Our analysis isolates the aggregator component to measure its inherent computational cost. In production streaming systems, windows would be processed individually with aggregation happening asynchronously, avoiding the simulation overhead present in our experimental framework. The vectorized implementation materializes overlapped windows for training speed but increases peak memory; the aggregator itself contributes negligibly to this memory overhead.

Table C.5: Aggregator attribution: fraction of end-to-end time attributable to overlap aggregation.

Configuration	Aggregator time share [%] ↓
Streaming ($W{=}2048$, $S/W{=}0.5$, $d{=}320$)	1.7
Vectorized ($L{=}100$k, $W{=}0.1L$, $S/W{=}0.5$)	<1

D Experimental Protocol and Datasets

D.1 Datasets

In our experiments, we rely on a set of well-established time-series benchmarks drawn from the UCR, UEA, and UCI repositories [8, 9]. The Human Activity Recognition (**HAR**) dataset [10] contains triaxial accelerometer and gyroscope streams recorded at 50 Hz from 30 volunteers who carried a smartphone on their waist while performing six everyday actions (walking, climbing or descending stairs, sitting, standing, and lying) [4]. For epileptic-seizure detection we adopt the version of the **Epilepsy** dataset simplified by TS-TCC: the original EEG collection—23.6-second segments from 500 subjects and five classes [28]—is reduced to a binary seizure/non-seizure task. The remaining benchmarks—Wafer, FordA, FordB, PhalangesOutlinesCorrect (POC), and ElectricDevices—are sourced from the UCR archive [8]. **Wafer** contains inline process-control sensor traces from silicon-wafer fabrication and is strongly imbalanced: defective wafers constitute 10.7% of the training set and 12.1% of the test set. **FordA** and **FordB** each comprise 500-sample engine-vibration sequences used to decide whether a specific subsystem fault is present; FordA was recorded under controlled laboratory noise, whereas FordB reflects normal operating conditions. The **POC** dataset merges three tasks derived from more than 1,300 radiographs employed for bone-age estimation, with labels indicating whether the automatically extracted phalange outlines are correct. Finally, the **ElectricDevices** dataset contains electricity-consumption profiles from 251 UK households, gathered to study residential usage patterns and help lower carbon emissions. Detailed statistics for all datasets appear in Table D.1.

Selection. The seven datasets span five domains, sequence lengths from 80 to 500, and channel counts from 1 to 9, providing diversity along the axes (length, channelization, class cardinality) most relevant to contrastive pretraining. Their training splits jointly comprise $\sim$13.7M time-steps, sufficient for in-batch InfoNCE training without external memory banks or queues [29, 30], which lets us use a single comparable training recipe across all datasets. Micro-datasets with only tens of sequences (e.g., *GunPoint*, *CBF*) cannot accommodate this batch-size requirement and are excluded.

D.2 Linear-Evaluation Protocol

For classification (Section 3.1) we follow the linear-evaluation protocol of Eldele et al. [3]: the backbone is pretrained on each training set without labels using the multi-scale contrastive objective of Section 2, then frozen. A linear SVM probe matching Lee et al. [5] is trained on 1% and 5% labeled subsets, and top-1 accuracy and macro-F1 are reported on the official test split. Per-dataset hyperparameters are documented in the released code.

D.3 Cue Retention Protocol

Why this experiment. Memory-augmented and state-propagating architectures expose intermediate states that are claimed to carry information across windows, but downstream task accuracy alone leaves two factors entangled: the architecture's capacity to carry information through its state, and the training objective's role in shaping what gets stored there. Downstream accuracy is also a poor instrument for either: many classification benchmarks are decidable from local features alone, so a model with an inert intermediate state can score competitively without ever propagating mid-range information, and a model that propagates well can underperform on tasks that don't reward it. The cue-retention probe is designed to bypass this conflation by introducing a controlled, out-of-distribution perturbation into the input and asking only whether evidence of that perturbation survives propagation

Table D.1: Datasets used for self-supervised classification

Dataset	# Train	# Test	Length	# Channel	# Class
HAR	7 352	2 947	128	9	6
Epilepsy	9 200	2 300	178	1	2
Wafer	1 000	6 174	152	1	2
FordA	1 320	3 601	500	1	2
FordB	3 636	810	500	1	2
POC	1 800	858	80	1	2
ElectricDevices	8 926	7 711	96	1	7

through the model's state. The probe is OOD by construction—the model is never trained on the cue—so success requires retention as an architectural and supervisory property, not as a task-specific shortcut.

Application to PMT. We use the protocol to disentangle two contributions specific to our design: the writable, window-aligned memory *interface* (architecture) and the PCL objective that supervises it (training). The architectural contribution is measured directly by the $\lambda_{\text{PCL}} = 0$ row, in which PMT retains the writable memory mechanism but receives no contrastive signal at the memory level. The supervisory contribution is recovered by the gap between $\lambda_{\text{PCL}} = 0$ and full PMT. Read this way, Table 2 reports two distinct quantities for our model in the same table: how much mid-range capacity the architecture provides on its own, and how much PCL adds on top.

Cue construction. A Hann-tapered burst is added to a single input channel, with amplitude $\alpha\sigma$ where σ is the channel's standard deviation computed *excluding* the cue span (preventing leakage through normalization), and $\alpha = 0.5$. The burst width is fixed at 20% of the sequence length across all runs and architectures. The burst is placed at a window offset chosen from a configured set of gap ratios, controlling the distance from cue to end of sequence. Cue retention is run with a non-overlapping-window variant of PMA for compute reasons; the variant shares the writable, PCL-supervised memory interface evaluated elsewhere in the paper. The models are trained on FordA *without any exposure to the cue*; the cue is introduced only at evaluation time on the test set; and the backbone remains frozen throughout.

Probe. We use the **end-of-sequence (EOS)** probe: features are extracted only from the final window per sequence, and a logistic-regression classifier (with ℓ_2-standardized features) predicts whether the sequence contained a cue at all. Restricting to a single fixed-position feature per sequence prevents the probe from exploiting positional correlations introduced by the gap-ratio sampling, and because the cue is upstream of the final window, the probe can succeed only if its evidence has been retained through state propagation.

Metrics. Let $g \geq 0$ index the gap (in windows) between the cue and the end of sequence; sequences without a cue contribute the negative class.

- **EOS AUC.** Standard binary ROC-AUC over the EOS probe's predicted cue-presence probabilities. Threshold-free; primary headline metric.
- **EOS Top-1.** Top-1 accuracy of the EOS probe at the natural decision threshold ($p = 0.5$).
- **EOS mF1.** Macro-F1 at the same decision threshold, balancing performance across present-vs-absent classes.

All three are averaged across seeds; the probe and feature extractors are otherwise identical across architectures, so AUC differences reflect representation quality, and the threshold-based gap between AUC and Top-1/mF1 reflects how well the representation places the decision boundary at a usable threshold.

Feature extraction. Per architecture, the EOS feature is the final-window state representation each architecture exposes natively: for PMT, the final PMA block's memory states at the last window (mean-pooled across slots when more than one is configured); for Transformer-XL, the segment cache at the final position; for xLSTM, the recurrent hidden state at the final time step. TS2Vec+SoftCLT has no native window-aligned state, so we unfold its intermediate convolutional embeddings into windows matching PMT's window length and stride and mean-pool within each window.

Supervisory signal. For PMT the EOS feature is the final memory state; for the matched-architecture baselines (Section 3.4) the corresponding state representation; for TS2Vec+SoftCLT, the windowized intermediate embeddings (the closest analogue in a backbone without explicit window-aligned state). To separate the writable-memory *interface* (architecture) from the effect of supervising it directly, we also report PMT with $\lambda_{\text{PCL}} = 0$.

D.4 Forecasting Protocol

PMT follows the forecasting evaluation protocol of Lee et al. [5]: a frozen encoder produces per-timestep representations, and a per-horizon ridge regressor predicts future values on ETTh1 and Electricity at horizons $\{24, 48, 168, 336, 720\}$, using the same data splits as SoftCLT. The protocols differ in one respect: PMT relies on windowed training to support in-batch negative mining, which requires preparing the forecasting data as fixed windows rather than full series with random temporal

crops. To ensure comparability, we retrain SoftCLT under this adapted windowed protocol while keeping their architecture and forecasting hyperparameters fixed; results in Table 3 use these retrained SoftCLT numbers.

Causal feature extraction. Per-timestep features are produced via the sliding-encode convention of Yue et al. [4], Lee et al. [5]: the feature at original time t is the backbone output at the final position of a separate forward pass over the left-padded input window $[\max(0,\ t-P),\ t]$, with $P = 200$. By construction, no input at position $> t$ participates in the computation of the feature at t, regardless of any non-causal attention internal to the backbone. For forecasting we configure the tokenizer with patch size 1 and stride 1 so that token positions align one-to-one with timesteps; this differs from the patch-based tokenizer used for classification (Section 3.6). Left padding uses NaN values, which the backbone treats as masked positions.

E Additional Experimental Results

E.1 Supervised Results and Additional Ablations

To verify that the architecture is not limited to SSL, we train PMT end-to-end with *full supervision* (cross-entropy on the label) using the same tokenizer and encoder backbone as in the main text. The `[CLS]` token is passed to a linear classification head. Unless noted, optimization and scheduling follow Appendix B (AdamW + cosine decay). For the fully supervised experiments in this section, no self-supervised losses are used.

E.2 Full-Supervision (Cross-Entropy) Results

Table E.1 reports Top-1 accuracy and macro-F1 on four representative datasets. Results show that PMT matches or exceeds a vanilla Transformer trained with the same supervised protocol on three of the four datasets (notably +6.0pp on FordA), with a small drop on FordB. These results confirm the *task-agnostic* nature of the backbone: while the main paper focuses on low-label SSL, the same architecture trains effectively in a purely supervised regime.

E.3 Short-Run Architectural Ablation (HAR)

To make the architectural comparison visible in the main body while keeping training time modest, we include a 50-epoch ablation on HAR that contrasts PMT with a vanilla Transformer and two windowed variants without the full PMA mechanism. For completeness we also include the 5% SSL condition (same data, identical backbone depth/width; SSL losses only in that column)—see Table E.2.

Takeaways. (i) In the short-run supervised setting, PMT is on par (within noise) with a vanilla Transformer. (ii) Under SSL, PMT yields consistently stronger features at 5% labels, suggesting the writable memory and progressive aggregation surface useful mid-range cues even when labels are scarce.

E.4 Robustness to Reset-State Initialization

The memory reset token M_r initializes the first-window state and can be mixed in by the learned reset gate when regimes change. We tested robustness to the random initialization of M_r by repeating HAR training five times with different seeds; Table E.3 shows negligible variance.

Table E.1: Fully supervised results (cross-entropy). Top-1 / macro-F1 (%).

	PMT (supervised)		Vanilla Transformer (supervised)	
Dataset	Top-1	mF1	Top-1	mF1
HAR	97.8	98.0	97.7	97.9
FordA	91.5	91.5	85.5	85.5
FordB	79.3	79.2	80.5	80.5
Wafer	99.8	99.5	99.5	98.7

Table E.2: HAR (50 epochs). Supervised vs. 5% SSL. Top-1 / macro-F1 (%).

	Supervised (CE)		SSL (5% labels, linear probe)	
Model	Top-1	mF1	Top-1	mF1
PMT	97.1	97.4	93.6	93.9
Vanilla Transformer	97.3	97.6	91.9	92.0
Windowed Transformer (with temporal state pass)	96.7	97.0	93.1	93.3
Windowed Transformer (no state pass)	96.8	97.1	93.0	93.2

Table E.3: Reset-state robustness (HAR). Supervised training with different random seeds for M_r; Top-1 accuracy (%).

Seed 42	Seed 123	Seed 456	Seed 789	Seed 2024	Mean $\pm$ Std
96.13	96.19	96.06	95.99	95.99	96.07 ± 0.08

In addition to the small numeric spread, predictions are identical for $\sim$99.7% of samples across seeds, indicating that PMT is insensitive to the particular initialization of the reset memory.

E.5 PMA Window-Size Ablation

Setup. We study the sensitivity of PMT to the *window length* W in Progressive Memory Attention (PMA). We parameterize W as a fraction ρ of the token length K (after patchification), i.e., $W = \lfloor \rho K \rfloor$ with $\rho \in \{0.10, 0.20, 0.30, 0.50, 1.00\}$. Unless noted, we keep the stride $S = \lfloor 0.5\, W \rfloor$, use a single memory slot per window (n_m=1), and leave every non-geometric hyperparameter unchanged (optimizer, temperatures for ICL/PCL/HGCL, Gaussian widths, augmentations). We pretrain on HAR with the same recipe as in the main experiments but in a short-run setting (reduced training budget); we then train linear SVM probes on the 1% and 5% label splits and report the average Top-1 accuracy and macro-F1. Results are averaged across the same random seeds used for our ablations.

(1) **Wide plateau at small/mid windows.** Performance is essentially constant for $\rho \in [0.1, 0.3]$, indicating that PMA's writable memory compensates for smaller windows by accumulating context progressively across windows and depth; this aligns with the receptive-field growth in Appendix A.3, Equation (A.10). (2) **Very large windows are unnecessary.** As $\rho \to 1.0$, accuracy and macro-F1 decline slightly. With few very large windows, overlap reduces and the progressive mechanism has fewer horizontal updates, dampening the benefits of memory refresh. (3) **Practical choice.** Any ρ in $[0.1, 0.3]$ is a safe default. We use ρ=0.2 in our configs to balance throughput (smaller attention tiles) and robust downstream accuracy without additional tuning. For this ablation we did *not* retune contrastive temperatures or Gaussian widths; the flat response in $[0.1, 0.3]$ therefore represents a conservative estimate.

E.6 Losses Evaluation and Ablation

Per-dataset weights in the main experiments. The loss weights λ_{ICL}, λ_{HGCL}, and λ_{PCL} are dataset-level hyperparameters in our main classification, forecasting, and cue-retention results. They are tuned per dataset using the same sweep ranges reported in the released configurations and are not held at a uniform value across datasets. The λ=1 uniform configuration reported in Table 5(a) is a methodological reference point used to characterize the loss combination, not the configuration used to produce the headline numbers in Table 1.

What the evaluations vary. Each evaluation in Table 5 sweeps a single weight while holding the other two at their per-dataset tuned values. Specifically, Table 5(b) varies each λ of the specific

Table E.4: PMA window-size ablation on HAR (short-run pretraining). $W = \rho K$, $S = 0.5W$, n_m=1. We report the average over 1% and 5% label splits. Performance is flat for $\rho \in [0.1, 0.3]$ and degrades as W approaches full context.

Window fraction ρ	Avg. Top-1 (%)	Avg. mF1 (%)
0.10	92.8	93.0
0.20	92.7	93.0
0.30	92.7	92.9
0.50	92.5	92.6
1.00	92.3	92.5

loss (i.e., from PCL, HGCL, and ICL) over $\{0, 0.01, 0.1, 0.5, 1\}$ on POC and Wafer with the other two fixed to the per-dataset values used elsewhere; Table 5(a) reports leave-one-out and single-only configurations on top of the λ=1 reference point. We sweep against tuned remainders rather than a uniform reference because the sweeps are intended to characterize the marginal effect of each loss in deployment, not its effect against an arbitrary baseline.

Why the $\sim$1pp sweep stability statement holds. The conclusion that uniform weighting is a strong default (accuracy varies by under 1pp on $\lambda_{\text{PCL}} \in [0, 1]$ and under 1.7pp on $\lambda_{\text{HGCL}} \in [0, 1]$) is read off the sweep tables, Table 5(b), where the remainders are held at tuned values. The corresponding span when starting from the λ=1 reference—i.e., the leave-one-out rows in Table 5(a)—is larger (4.2pp ICL, 2.8pp HGCL, 0.2pp PCL on Top-1), which reflects the dataset mix in that table differing from the POC/Wafer focus of the sweep tables. Practically, this means: if a practitioner deploys the protocol with all three losses at λ=1, the result should be within $\sim$ 2pp of a tuned configuration; per-dataset tuning improves over uniform weighting but is not required.

Dataset selection for evaluation and ablation. The combined-loss ablation in Table 5(a) covers FordA, HAR, POC, and Wafer; the per-weight sweeps in Table 5(b) are restricted to POC and Wafer. POC and Wafer are the two smallest datasets in our benchmark mix, which makes them tractable for sweeping a single λ across five values—a full per-dataset sweep across all seven benchmarks would multiply the run count beyond what is justified for a confirmatory evaluation. Restricting the sweep to two datasets is sufficient because the body claim is about *stability* of the loss combination across λ values, not about per-dataset optimal λ selection: showing that two independent datasets exhibit a similar flat profile (under 1pp on λ_{PCL}, under 1.7pp on λ_{HGCL}) supports the conclusion that uniform weighting is a strong default. Per-dataset tuning, where used in main results, follows the standard configuration sweep over a wider hyperparameter set and is not specific to the ablation tables. This has a practical implication for adopting the protocol on new datasets: the sweeps suggest that per-dataset λ tuning produces measurable but small gains, and that the dominant choice—whether to deploy the multi-scale objective at all—is unlikely to depend on it. Across the four datasets we evaluated, uniform λ=1 landed within $\sim$ 2pp of tuned configurations, suggesting that practitioners adopting the protocol on new datasets can reasonably start from uniform weighting, with the larger gains coming from the protocol itself rather than from weight selection.

E.7 Additional Results for Learned Representations

Fig. E.1 shows two spliced waveforms of the same class (walking upstairs). It shows four off-center diagonal lines, together signifying the foot strikes of the subject. We included this supplementary figure for completeness, here plotting all 9 channels, rather than the single channel we plot for the other figures. Additionally, this same-class comparison highlights similar patterns to what we see across all walking-related classes: the foot-strike pattern (diagonal lines).

Fig. E.2 shows three sets of cosine similarity matrices from the Epilepsy dataset. Each pair consists of a token representation comparison on the left and a memory state comparison on the right. The first column of each pair compares *token embeddings*; the right column compares the corresponding *progressive-memory states*. **Same-class pairs (columns 1–4):** Diagonal stripes are faint at token level but become sharply defined after memory integration, indicating that the progressive-memory layer consolidates class-specific cues while damping phase noise. **Cross-class pair (columns 5–6):** Residual token-level correlations disappear almost entirely in the memory states, suggesting that sequences from different classes are mapped to near-orthogonal regions of latent space. Overall, class separability is mainly realized *after* recurrent aggregation; token embeddings alone retain limited spectral overlap.

Fig. E.3 shows a double-spliced waveform in the class pattern: Standing, Laying, Standing. Here we see a clear checkerboard pattern in the hybrid-to-hybrid comparison. This demonstrates that the model produces similar features for the two unique standing waveforms, and distinct features for the laying class.

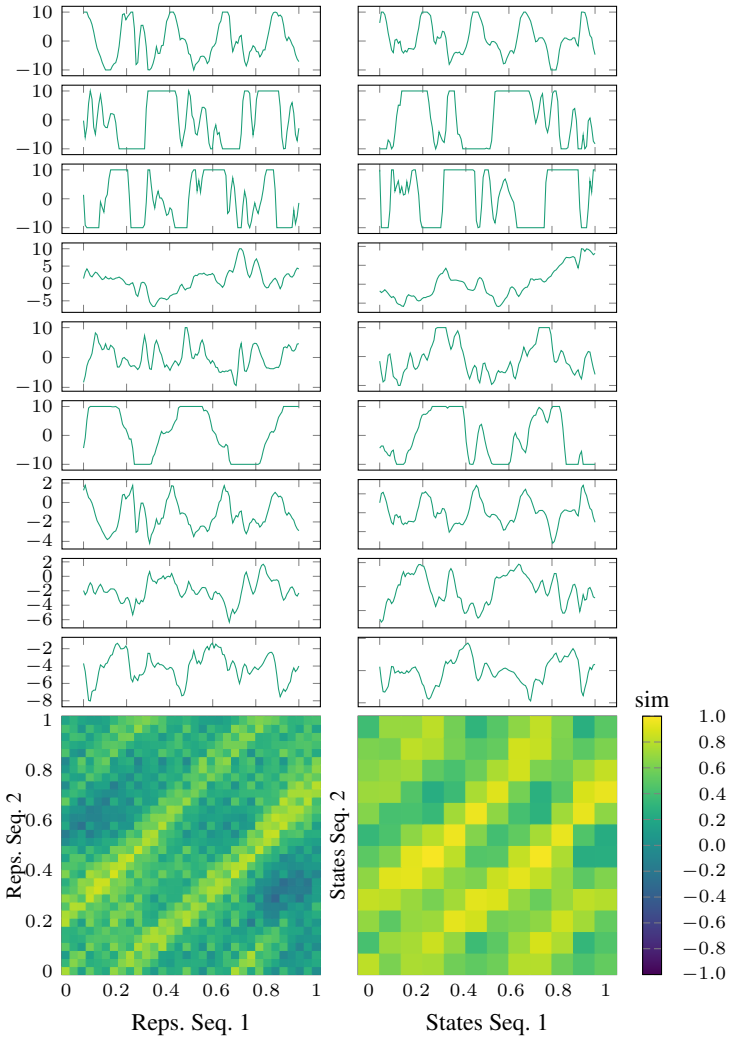

Figure E.1: The cosine similarity matrix between the token (bottom left) and the memory states (bottom right) representations of two signals (number 6 and 12, respectively, from the HAR validation set) of the same class (walking upstairs). The nine channels of the signals are plotted independently (top). The similarities shows that the representations and states correlate to each other between the temporal domain.

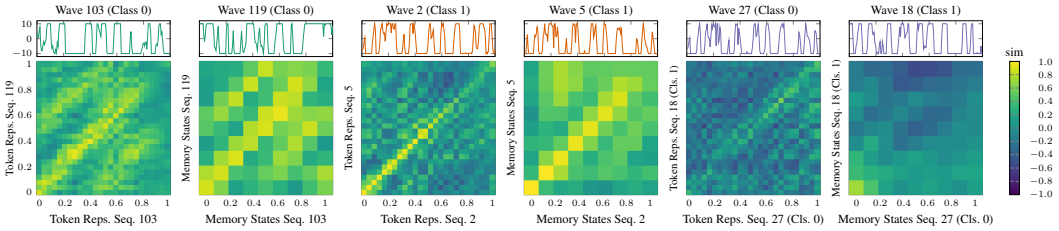

Figure E.2: Cosine similarity matrices of the representations and states for the Epilepsy dataset. Comparison of two sequences from the same class 0 (left) and class 1 (middle), and between two sequences from different classes (right).

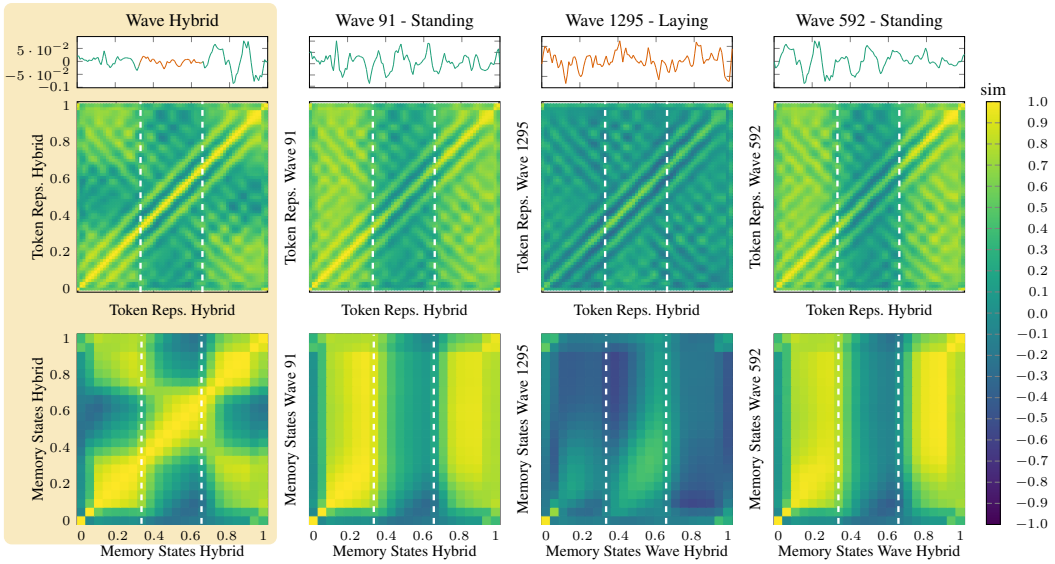

Figure E.3: Cosine similarity matrices for the representations and states between pair-wise signals from HAR. Higher similarity shows that the signals correlate as evidenced by the learned embeddings. The vertical bars denote the different sections of the hybrid wave. The wave number corresponds to the sample index in the validation dataset.

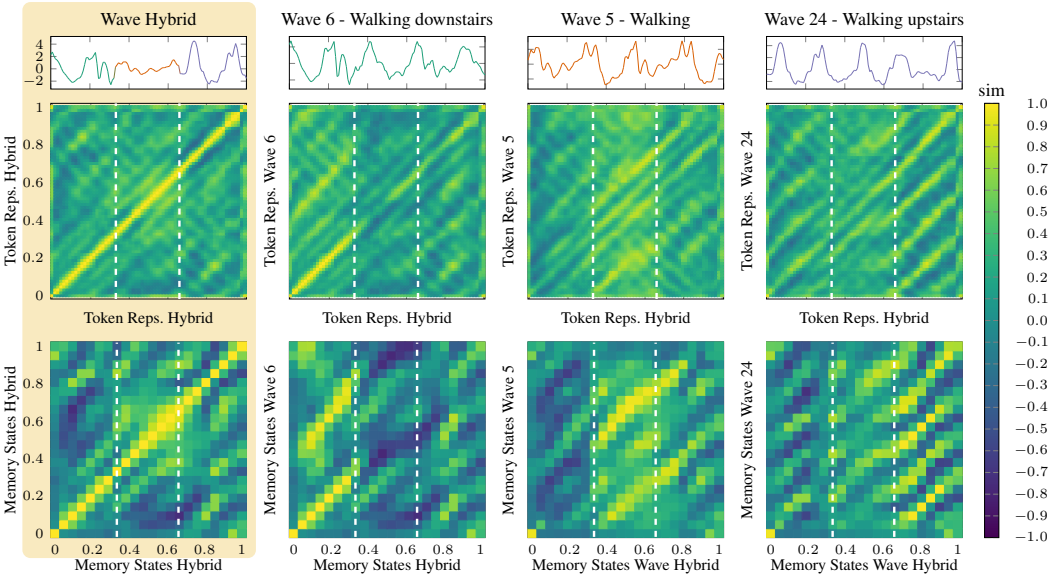

Figure E.4: Cosine similarity matrices for the representations and states between pair-wise signals from HAR. Higher similarity shows that the signals correlate as evidenced by the learned embeddings. The vertical bars denote the different sections of the hybrid wave. The wave number corresponds to the sample index in the validation dataset.

Fig. E.4 shows a 3-way splice of semantically close but unique class waveforms (walking downstairs, walking, and walking upstairs). What we want to see in this figure is no correlation across splice borders. The figure shows that both the memory-state and token-level representations are distinct, meaning the model has learned both local-level and mid-range *class-separable* and *temporal compositional* representations for these semantically proximal activities.

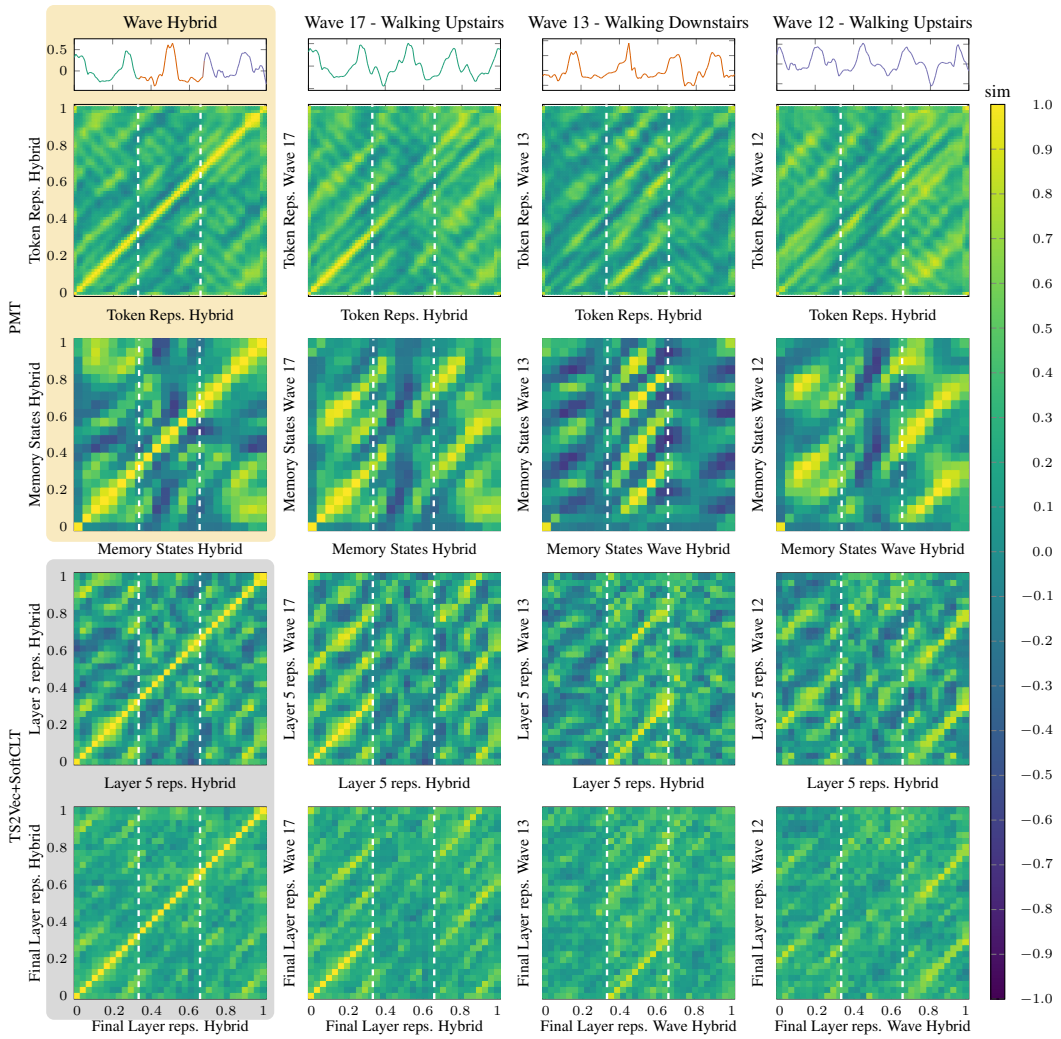

Figure E.5: Cosine similarity matrices for the embeddings of PMT and TS2Vec + SoftCLT, from layer 5 and 10 (final), between pair-wise signals from HAR. Higher similarity shows that the signals correlate as evidenced by the learned embeddings. The vertical bars denote the different sections of the hybrid wave. The wave number corresponds to the sample index in the validation dataset. Same samples as Fig. 3.

Fig. E.5 shows TS2Vec+SoftCLT's embeddings at layer 5 and the final layer, as trained in the SoftCLT paper [5]. This figure uses the same data splice as in Fig. 3. TS2Vec's stacked dilated convolution layers extract the "foot-strike" pattern we see in Fig. 3, but lack the clear "checkerboard" pattern in the hybrid-to-hybrid comparison. We include this figure for completeness; although extracting representations from different layers may not be perfectly analogous to our token vs. memory-state comparison, it illustrates the difference in mid-range motif capture.

To further showcase the behavior of the representations at different structural levels, we visualize t-SNE and PCA projections on additional HAR sequences. While these sequences differ from the specific splices used in the similarity-matrix experiments, they clearly illustrate how the representation space evolves from fine-grained tokens to compressed memory states.

First, Fig. E.6 projects the representations of a Standing-Laying-Standing splice using t-SNE, confirming that the memory states for the two distinct standing segments cluster tightly together and remain cleanly separable from the laying segment. Similarly, Fig. E.7 shows a PCA projection over another unique HAR splice alternating between walking and walking downstairs. Because the PMA unrolls over sliding windows with a stride greater than one, the model produces a larger number of token representations (capturing *local nuance*) compared to the more compressed memory states (capturing

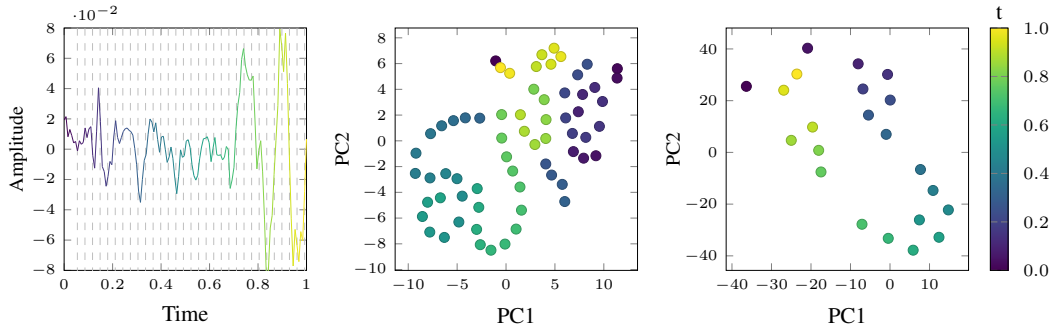

Figure E.6: (Left) Double spliced HAR waveform. (Middle) t-SNE of representations (tokens) of the signal. (Right) t-SNE of memory-states. The splicing is sourced from two unique waveforms from separate activities (standing and laying)

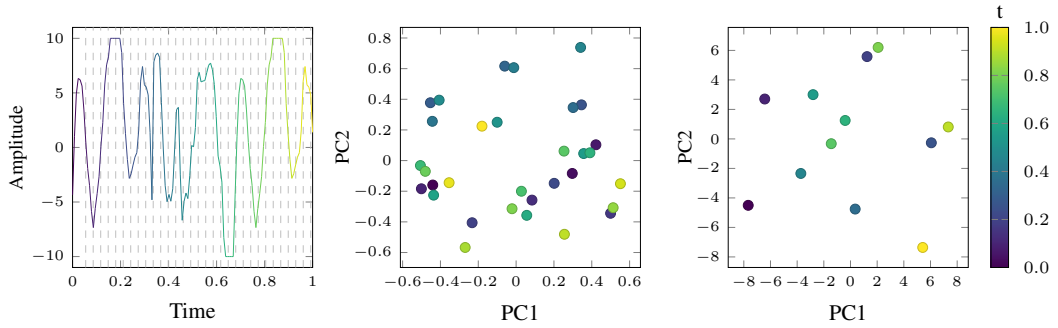

Figure E.7: (Left) Double spliced HAR waveform. (Middle) PCA of representations (tokens) of the signal. (Right) PCA of memory-states.

mid-range motifs). The center panel highlights PMT's ability to extract token representations that are semantically consistent with the underlying signal, as demonstrated by their linear separability based on the source class. This strong linear separability extends to the memory states shown in the right panel (however, we note that the first memory state in the sequence is typically less semantically distinct due to its small initial receptive field, as derived in Appendix A.3).

F Limitations

PMT's framework is independent of the input stem we use, but in this work we tokenize via a fixed-stride 1D convolution, the standard choice for patch-based time-series architectures. As Section 3.6 shows, patch length is well-behaved across a wide plateau but stride must be small enough to capture short-lived events, making tokenization the main per-dataset adjustment in our recipe. Replacing the patch stem with a learned or signal-adaptive tokenizer is a natural avenue for removing this tuning step.

The three contrastive objectives, particularly ICL at the sequence level, depend on large in-batch negative sets. This is a standard characteristic of InfoNCE-based self-supervised frameworks but constrains application to extremely small datasets or resource-limited hardware without falling back on momentum queues, which we did not explore.

G LLM Disclosure

We used a large language model as a writing assistant to improve clarity and grammar, and to help surface potentially relevant related work during scoping. All technical claims, modeling choices, experiments, analysis were the work of the authors. No citations were included without verification. No empirical results were generated with LLMs.