Perseus: Interactive Time Series Segmentation with Sparse Supervision via Stateful Memory

Ching Chang*‡, Ming-Chih Lo*, Chiao-Tung Chan†, Wen-Chih Peng*, and Tien-Fu Chen*
*Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan
†Department of Electrical and Control Engineering, National Yang Ming Chiao Tung University, Hsinchu, Taiwan
‡Corresponding author: blacksnail789521@gmail.com

arXiv:2510.09930v2 [cs.LG] 21 Aug 2026

***Abstract*—Real-world systems, ranging from industrial manufacturing to wearable healthcare, generate multivariate time series with hierarchical states ranging from coarse regimes to fine-grained events. Unlike zero- or few-shot segmentation, our setting uses dense state labels for model training. Sparse expert prompts provide inference-time corrections that resolve sequence-specific ambiguities without retraining. In practice, this feedback is grouped around selected events or transitions, leaving large portions of the timeline unprompted. The prompt-based sliding-window baselines evaluated here are *stateless* with respect to user interaction history: they use guidance only within the current window and cannot retain it across these gaps. To address this, we propose Perseus (Persistent Segmentation with User Supervision), a framework that transitions from synchronous processing to *asynchronous state management*. Perseus decouples supervision from inference via a distinct Write-Read architecture: grouped user cues are asynchronously encoded into a persistent memory bank (Write), which is then actively queried by the inference engine (Read) to service unprompted windows. This mechanism bridges supervision gaps by conditioning predictions on a global history of interactions rather than solely on local inputs. Extensive experiments on six datasets demonstrate that while evaluated stateless prompting baselines degrade significantly under grouped supervision, Perseus maintains robustness and achieves up to 85% accuracy improvement in multi-granularity settings. Code and preprocessing instructions are available at https://github.com/blacksnail789521/Perseus.**

***Index Terms*—Time Series Segmentation, Interactive Segmentation, Persistent Memory, Prompting, Multiple Granularities**

I. Introduction

Modern web platforms, mobile apps, and manufacturing systems generate rich multivariate time series capturing evolving user and device states. Examples include smart home activity recognition [1]–[3], financial market monitoring [4]–[6], and sports performance tracking [7]–[9]. These states often exhibit multiple levels of granularity, from coarse regimes to fine-grained events [10]–[13]. For instance, long-term market cycles overlap with short-lived volatility spikes [4], [14], while applications like anomaly detection rely on understanding these interactions [15], [16]. To analyze such behaviors, domain experts increasingly use interactive tools to inject sparse feedback [17]–[19]. However, unlike current model assumptions, real-world interaction is highly *bursty and grouped*. Practitioners rarely place prompts[1] uniformly. Instead, they typically annotate specific event clusters, leaving long timeline segments completely untouched. The challenge lies in ensuring this localized feedback effectively guides segmentation across the inevitable supervision gaps. This is a fully supervised setting rather than zero- or few-shot segmentation: models are trained with dense per-timestep state labels, while sparse prompts let experts correct sequence-specific errors and select the desired granularity at inference without retraining.

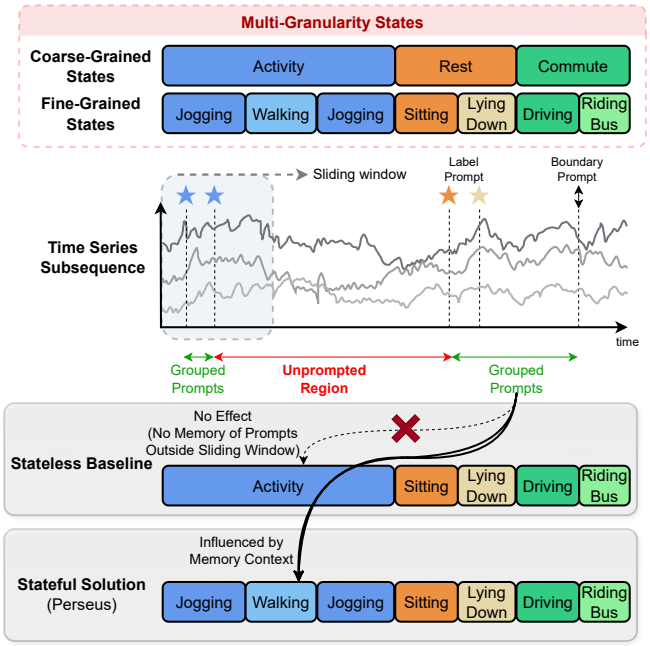


Fig. 1. The unprompted region failure mode. Grouped supervision creates gaps where stateless prompting models cannot access prior interactions. Perseus uses stateful memory to retrieve that guidance in unprompted windows.

Prompt-based sliding-window methods such as PromptTSS [20] are *stateless* with respect to user interaction: they apply guidance locally and do not retain it across windows. Conventional supervised segmenters [21]–[23] are not natively interactive and, when equipped with a stateless prompting interface, likewise cannot persist feedback across windows. These stateless prompting mechanisms therefore operate under an implicit *Gapless Supervision Assumption*, expecting every sliding window to contain at least one prompt. In realistic grouped scenarios, vast stretches become *unprompted windows*

[1]In this work, we use the term "prompting" to refer to the injection of sparse signal cues (e.g., clicks, labels) to guide segmentation, distinct from natural language instructions used in Large Language Models (LLMs) [18], [20].

lacking immediate signals, as shown in Figure 1. Because interaction state is local to each window, these methods suffer from *contextual amnesia*: once the window moves past an annotation, guidance is inaccessible [24]. The trained model still produces predictions in these gaps, but it cannot use the expert's earlier corrections. Consequently, most of the sequence is segmented without leveraging user guidance, limiting interactive labeling efficiency.

Our analysis reveals this limitation is catastrophic under strictly grouped sparse supervision. When supervision density drops below 5% and is concentrated in bursts, the evaluated stateless prompting baselines degrade sharply—dropping from near-perfect accuracy to approximately 60% in multi-granularity settings. Without a mechanism to propagate user intent, these models cannot bridge the gap between sparse interaction and global segmentation, limiting their usefulness in grouped-feedback deployments.

To bridge these gaps, we propose **Perseus** (**Per**sistent **Se**gmentation with **U**ser **S**upervision), a framework transitioning from synchronous stateless prompting to *asynchronous state management*. Unlike stateless prompting approaches, Perseus decouples supervision from inference via a distinct *Write-Read architecture*. In the asynchronous write phase, sparse cues are encoded into a persistent memory bank. During the sequential read phase, the model actively queries this state to service unprompted windows, sustaining intent across unannotated gaps. By maintaining persistent state, Perseus ensures global consistency regardless of feedback distribution.

In summary, our contributions are:

- **Asynchronous Stateful Framework.** We propose Perseus, a framework that decouples prompt encoding from inference via an asynchronous *Write-Read* mechanism. By fusing sparse prompts with local context into a persistent memory bank, Perseus bridges contextual amnesia in stateless prompting.
- **Robustness to Grouped Supervision.** We identify the unprompted window phenomenon as a critical failure mode. Experiments show that while evaluated stateless baselines degrade significantly when feedback is grouped, Perseus maintains robustness, achieving up to 85% accuracy improvement in multi-granularity tasks.
- **Comprehensive Evaluation.** Evaluating on six wearable and industrial datasets, Perseus demonstrates superior refinement efficiency in iterative settings, achieving an average per-iteration gain of 2.66 percentage points compared to 1.19 for the stateless PromptTSS baseline.

II. Related Work

A. Stateless Prompting in Time Series Segmentation

Time-series segmentation encompasses several complementary problem formulations. Change-point detection localizes transitions in a stream [1], while unsupervised latent-state inference and subsequence clustering discover recurring regimes and partition sequences without interactive input [11], [12]. Interactive machine learning instead places user corrections inside an iterative modeling loop [25]. Perseus connects these lines by retaining sparse user guidance for prompt-based segmentation across inference windows, rather than proposing a new offline discovery or clustering objective.

Within supervised segmentation, neural architectures capture complex spatial and temporal dependencies. Standard architectures like DeepConvLSTM [26] and U-Time [23] model local contexts, while methods like MS-TCN++ [22] and PrecTime [21] improve boundary precision using multi-stage temporal convolutions.

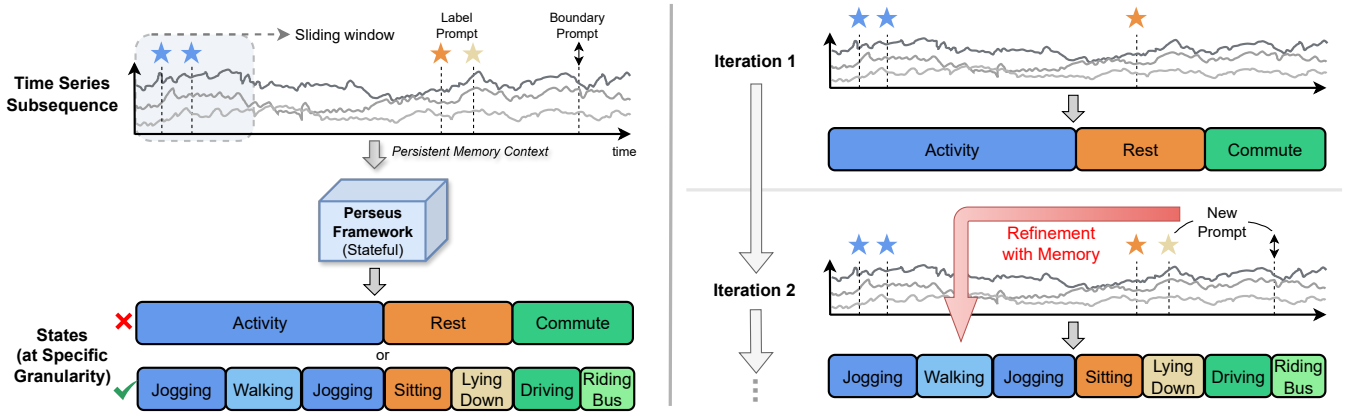
These architectures can model temporal dependencies within their inputs. Here, *stateless* refers specifically to interaction history across inference windows, not to general sequence-modeling capacity. Under this definition, PromptTSS [20], which introduced sparse prompting for time series segmentation, is stateless: it applies each prompt within its local sliding window and does not retain it after that window is processed. Under grouped sparse prompting, this stateless interface creates *contextual amnesia*: predictions outside the prompted window cannot access earlier user guidance.

Beyond segmentation, related domains offer complementary solutions. Hierarchical classification frameworks [27]–[31] target static data rather than dynamic evolution. Domain adaptation techniques [32]–[34] address distribution shifts but cannot incorporate specific user feedback during inference. Similarly, active learning [35]–[37] reduces annotation effort but typically requires retraining, which disrupts real-time interactive sessions. Perseus targets the narrower problem of persisting sparse inference-time guidance across windows for multi-granularity segmentation.

B. Interactive Mining & Prompting

Interactive machine learning aims to integrate human feedback directly into the inference process. In Natural Language Processing (NLP), Large Language Models (LLMs) adapt to new tasks via textual instructions [38], [39]. Similarly, in Computer Vision, systems like Segment Anything (SAM) [18], [24] allow users to steer segmentation via spatial clicks or boxes. It is important to note that in the context of time series segmentation, prompting refers to the injection of sparse signal cues (e.g., label clicks or boundary markers) rather than natural language instructions used in LLMs [20].

For time series, PromptTSS applies prompt signals only within the immediate local region [20]. Consequently, most of the sequence remains unprompted and segmented without leveraging user guidance. This locality restricts utility in high-stakes domains like industrial monitoring or healthcare, where the goal is for a small number of expert annotations to propagate consistently across long, complex sequences. Our work addresses this gap by introducing a stateful memory mechanism that persists encoded prompts, ensuring that interactive guidance extends beyond local neighborhoods to achieve global coherence.



(a) Subsequence-based Time Series Segmentation

(b) Iterative Refinement

Fig. 2. Problem setup and iterative refinement in Perseus. (a) Segmentation is performed over subsequences rather than a single sliding window, with prompts provided as grouped label or boundary cues. (b) In the iterative process, additional prompts are encoded into the persistent memory bank to update the global state. This enables the model to refine predictions across the entire timeline, transforming coarse regimes (e.g., Activity, Rest) into fine-grained actions (e.g., Jogging, Sitting), even in regions that received no new local feedback.

III. Methodology

A. Problem Formulation

Figure 2(a) illustrates the overall problem formulation. Let $\mathbf{X} = (x_1, \ldots, x_L) \in \mathbb{R}^{L \times C}$ denote a complete and evenly-sampled multivariate time series of length L with C channels. The corresponding ground-truth state sequence is $\mathbf{S} = (s_1, \ldots, s_L) \in \mathbb{Z}^L$, where each s_t belongs to one of K discrete states. To make training and inference efficient, we partition $\mathbf{X}$ into M non-overlapping *subsequences*, each of length L_s:

$$\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M)}], \quad \mathbf{x}^{(m)} \in \mathbb{R}^{L_s \times C}.$$

Here, $M = \lfloor L/L_s \rfloor$ denotes the total number of subsequences. Each subsequence $\mathbf{x}^{(m)}$ is associated with its state labels $\mathbf{s}^{(m)} \in \mathbb{Z}^{L_s}$. Within each subsequence, we further extract W sliding windows of length T and stride S:

$$\mathbf{x}^{(m)} \mapsto \{\mathbf{w}_1^{(m)}, \ldots, \mathbf{w}_W^{(m)}\}, \quad \mathbf{w}_j^{(m)} \in \mathbb{R}^{T \times C}.$$

*a) **Prompt Definition**:* In addition to the time series data, we allow sparse prompts to guide segmentation. A *label prompt* is represented as a vector $p_l \in \{0,1\}^{2K}$, where the first K dimensions encode the correct class in a one-hot manner and the second K dimensions encode incorrect classes using a multi-hot representation. A *boundary prompt* is represented as a binary value $p_b \in \{0,1\}$, indicating whether a state transition occurs at a given timestamp. Each prompt corresponds to exactly one timestamp within the subsequence. We denote the set of all prompts provided for a subsequence m as $\mathcal{P}^{(m)}$. The goal of time series segmentation is to learn a mapping that, given a subsequence $\mathbf{x}^{(m)}$ together with a set of prompts $\mathcal{P}^{(m)}$, predicts its corresponding state sequence $\mathbf{s}^{(m)}$.

*b) **Grouped Supervision & The Gapless Assumption**:* Stateless prompt-based sliding-window models implicitly assume *Gapless Supervision*, where every sliding window is expected to contain at least one prompt ($\forall j, \mathcal{P}^{(m)} \cap \mathbf{w}_j^{(m)} \neq \emptyset$). However, real-world interaction is grouped and sparse, creating *unprompted windows* where local supervision is absent ($\mathcal{P}^{(m)} \cap \mathbf{w}_j^{(m)} = \emptyset$). In these gaps, the trained model can still predict from its input, but cannot use guidance provided elsewhere in the subsequence. Perseus addresses this by learning a stateful mapping $\hat{y} = F(\mathbf{w}_j^{(m)}, \mathbf{M}_{\text{all}})$, where predictions are conditioned on the global history of prompts stored in memory rather than local signals alone.

B. Perseus Architecture

To address the *contextual amnesia* caused by stateless prompting, Perseus introduces *Asynchronous State Management*. Unlike stateless prompt-based approaches that pair prompts with individual windows, Perseus decouples the lifespan of user feedback from the processing window via a distinct *Write-Read* mechanism. The framework consists of five main components: the time series encoder, the prompt encoder, the memory encoder, the memory bank, and the state decoder. Figure 3 illustrates the overall architecture, distinguishing between the asynchronous memory write phase (Phase 1) and the sequential memory read phase (Phase 2).

*a) **Time Series Encoder**:* Given a window $\mathbf{w} \in \mathbb{R}^{T \times C}$, the time series encoder f_x maps it into a sequence of patch-level tokens. Directly applying a Transformer to long time series windows is computationally expensive. To address this, we employ *patching* [24], [40], where adjacent time steps are grouped into patch tokens, reducing sequence length while preserving local temporal patterns. Formally, the patching operation produces

$$\mathbf{w}_{\text{patched}} = \text{Patching}(\mathbf{w}), \tag{1}$$

where $\mathbf{w}_{\text{patched}} \in \mathbb{R}^{T_p \times (C \cdot P)}$ with patch length P and T_p denoting the number of patches. The Transformer encoder is then applied:

$$\mathbf{z}_x = f_x(\mathbf{w}_{\text{patched}}), \tag{2}$$

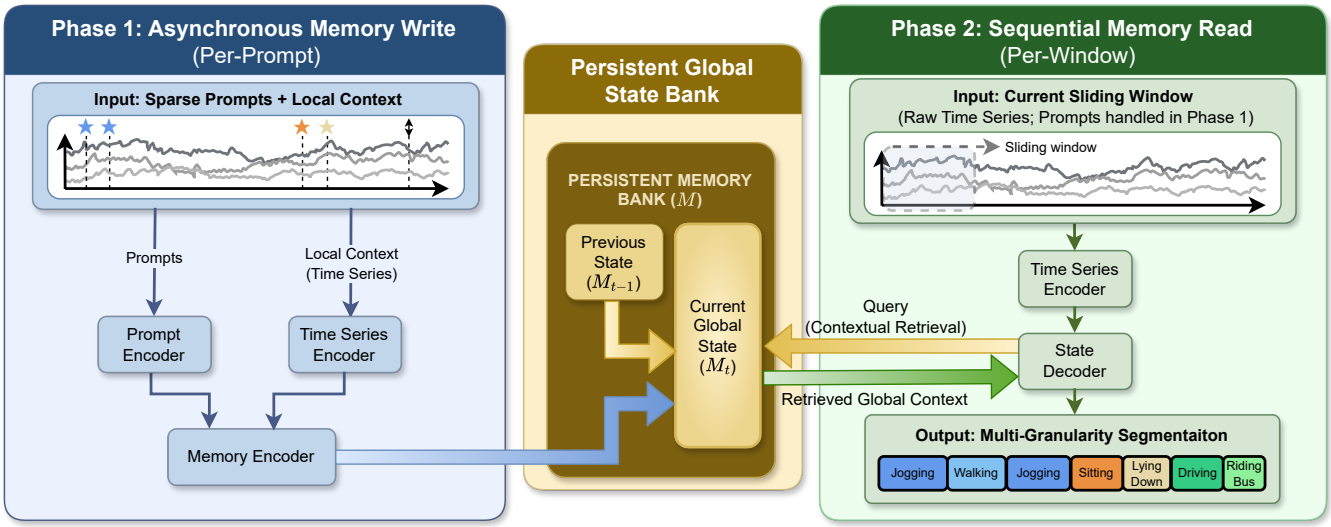


Fig. 3. Overview of the **Perseus** framework featuring asynchronous state management. **Phase 1 (Memory Write):** Sparse prompts and their local context are asynchronously encoded into a persistent global state bank to update the memory tokens. **Phase 2 (Memory Read):** During sequential inference, the state decoder actively queries this memory bank to retrieve context for every sliding window, including unprompted ones. This decouples supervision from inference, allowing the model to progressively refine multi-granularity segmentation as the global state evolves.

yielding $\mathbf{z}_x \in \mathbb{R}^{T_p \times D}$, where D is the embedding dimension.

*b) **Prompt Encoder**:* The prompt encoder f_p maps each user-provided prompt into a D-dimensional embedding. Each prompt corresponds to exactly one timestamp within a subsequence. We consider two types of prompts: label prompts and boundary prompts.

A label prompt is represented as a vector $p_l \in \{0,1\}^{2K}$. The first K dimensions form a one-hot vector indicating the correct class for the chosen timestamp, while the second K dimensions form a multi-hot vector indicating classes that should not occur at that timestamp. This dual encoding allows the prompt to convey both positive and negative supervision simultaneously. A boundary prompt is represented as a binary value $p_b \in \{0,1\}$. Here, $p_b = 1$ specifies that a state transition should occur at the given timestamp, while $p_b = 0$ specifies that the state should remain unchanged. Thus, boundary prompts directly guide the model in refining segmentation boundaries. Both types of prompts are projected into the shared embedding space $\mathbb{R}^D$. Label prompts are transformed by a linear projection, whereas boundary prompts are mapped through a learned embedding table. In addition, every prompt embedding is augmented with a type embedding (distinguishing label vs. boundary) and an aspect embedding (distinguishing correct vs. incorrect). The type embedding identifies whether the payload is a label or boundary cue, while the aspect embedding identifies whether that cue is declared correct or incorrect. After contextual fusion, these distinctions remain encoded in the memory token that the state decoder later queries. The final prompt embedding is expressed as:

$$\mathbf{z}_p = \begin{cases} \text{Linear}(p_l) + \mathbf{e}_{\text{type}} + \mathbf{e}_{\text{aspect}} & \text{if prompt is label} \\ \text{Embed}(p_b) + \mathbf{e}_{\text{type}} + \mathbf{e}_{\text{aspect}} & \text{if prompt is boundary} \end{cases} \tag{3}$$

*c) **Memory Encoder (Write Mechanism)**:* The memory encoder f_m drives the *asynchronous memory write* phase (Phase 1). Operating independently of the sliding window inference, it transforms each sparse prompt into a *memory token* that can be persisted in the memory bank. Its purpose is to fuse the supervision carried by the prompt with local temporal evidence from the subsequence, ensuring that the stored token reflects both the user guidance and the surrounding signal dynamics. For a prompt anchored at timestamp t_c, we extract a local context window of length T_{ctx} centered at t_c. This window is encoded by the time series encoder f_x to produce context tokens $\mathbf{z}_{ctx} \in \mathbb{R}^{T_{ctx} \times D}$. Given the prompt embedding $\mathbf{z}_p \in \mathbb{R}^D$ and the context tokens $\mathbf{z}_{ctx}$, the memory encoder applies cross-attention with the prompt as query and the context as key and value:

$$\mathbf{m} = f_m(\mathbf{z}_p, \mathbf{z}_{ctx}), \tag{4}$$

where $\mathbf{m} \in \mathbb{R}^D$ is the resulting memory token. This token captures both the user's intent at t_c and the semantic context of the event, preparing it for global retrieval.

*d) **Memory Bank**:* The memory bank persists memory tokens across iterations, enabling the model to service unprompted windows. At the beginning of an iteration, it contains previously written tokens $\mathbf{M}_{\text{old}} \in \mathbb{R}^{N_{\text{old}} \times D}$. New tokens $\mathbf{M}_{\text{cur}} \in \mathbb{R}^{N_p \times D}$, generated by the memory encoder from N_p prompts in the current iteration, are appended to form the updated bank:

$$\mathbf{M}_{\text{all}} = [\mathbf{M}_{\text{old}}; \mathbf{M}_{\text{cur}}], \quad N_{\text{all}} = N_{\text{old}} + N_p, \tag{5}$$

where $[\cdot;\cdot]$ denotes concatenation. By persisting across iterations, the memory bank accumulates knowledge from multiple prompts, supporting long-term refinement of segmentation predictions.

*e) **State Decoder (Read Mechanism)**:* The state decoder g_s drives the *sequential memory read* phase (Phase 2). It processes raw sliding windows during inference and actively queries the persistent memory bank to retrieve context, even when the local window contains no prompts. We employ a *Two-Way Transformer* that maintains a time-series stream $\mathbf{X}$ and a memory stream $\mathbf{M}$. First, self-attention lets the time-series tokens exchange context within the current window. Second, the time-series tokens query the memory tokens ($Q = \mathbf{X}$, $K = V = \mathbf{M}$), injecting retrieved supervision into the current-window representation. Third, self-attention among memory tokens integrates evidence across stored prompts. Fourth, the memory tokens query the updated time-series tokens ($Q = \mathbf{M}$, $K = V = \mathbf{X}$), aligning the memory stream with the current window. Fifth, separate feed-forward networks update both streams. The prediction head reads only the updated time-series stream after the blocks are stacked. Updates to the memory stream are local to the decoder call and do not overwrite the persistent memory bank. Formally, given window embeddings $\mathbf{z}_x \in \mathbb{R}^{T_p \times D}$ and memory tokens $\mathbf{M}_{\text{all}} \in \mathbb{R}^{N_{\text{all}} \times D}$ from the memory bank, the decoder produces

$$\hat{\mathbf{y}} = g_s(\mathbf{z}_x, \mathbf{M}_{\text{all}}) \in \mathbb{R}^{T \times K}. \tag{6}$$

Here $\hat{\mathbf{y}}$ denotes the predicted per-timestep state probabilities after de-patchification. Since the encoder operates on patch tokens, de-patchification expands patch-level logits back to the original timestamp resolution by distributing each patch prediction to its covered time steps and averaging in regions of overlap. This architecture ensures that predictions are informed both by the current input and by the accumulated supervision stored in the memory bank.

*f) **Complexity & Scalability Analysis**:* Each prompt contributes one D-dimensional memory token, so storage is $O(N_{\text{all}}D)$ and grows linearly with the cumulative prompt count unless the bank is capped. For each window, cross-attention between T_p patch tokens and N_{all} memory tokens costs $O(T_p N_{\text{all}} D)$, while self-attention among memory tokens contributes $O(N_{\text{all}}^2 D)$. Treating the embedding dimension as fixed, the additional read cost is therefore $O(T_p N_{\text{all}} + N_{\text{all}}^2)$, not $O(1)$ with respect to the number of stored prompts. Under a fixed sparse prompt budget, N_{all} is bounded and this overhead remains modest for a given window length. A capacity-limited deployment could additionally use an eviction policy, whose accuracy tradeoff we leave for future study.

C. Iterative Training and Evaluation

Perseus is fully supervised rather than zero- or few-shot: dense ground-truth state sequences supervise the loss at every training timestamp, while sparse prompts simulated from those labels train the interaction pathway. At inference, model weights remain fixed and users provide only sparse label or boundary prompts to correct sequence-specific errors and express the desired granularity without retraining. The iterative procedure progressively incorporates these prompts and accumulates them in memory, modeling how segmentation improves as feedback is added (Figure 2(b)). At each iteration r, the model performs two key procedures:

*a) **Step 1: Prompt Sampling & State Update**:* At the start of iteration r, we simulate interactive feedback by sampling N_p new prompts from the ground-truth labels for each subsequence. Unlike the window-based inference, this sampling occurs at the *subsequence level*. For a subsequence m, the sampled prompts are encoded (f_p) and fused with local context (f_m) to generate new memory tokens:

$$\mathbf{M}_{\text{cur}}^{(m)} = f_m(f_p(p^{(m)}), \mathbf{z}_{ctx}^{(m)}). \tag{7}$$

These new tokens $\mathbf{M}_{\text{cur}}^{(m)}$ are immediately appended to the memory bank, updating the global state available for the subsequent inference step.

*b) **Step 2: Optimization via Memory Read**:* Once the state is updated, we perform inference on every sliding window $\mathbf{w}_j^{(m)}$ within the subsequence. All prompts collected in iteration r are written before any window is read, so our experiments model offline, non-causal refinement: a prompt later on the timeline can guide an earlier window within the same subsequence. The state decoder g_s utilizes the updated memory bank $\mathbf{M}_{\text{all}}^{(m)}$ to generate predictions for unprompted windows:

$$\hat{\mathbf{y}}_j^{(m)} = g_s\big(\mathbf{z}_{x,j}^{(m)}, \mathbf{M}_{\text{all}}^{(m)}\big). \tag{8}$$

The model is optimized using cross-entropy loss averaged over all windows and timesteps in the iteration:

$$\mathcal{L}^{(r)} = \frac{1}{W \cdot T} \sum_{j=1}^{W} \sum_{t=1}^{T} \mathcal{L}_{CE}\big(s_{j,t}^{(m)},\, \hat{y}_{j,t}^{(m)}\big), \tag{9}$$

where $s_{j,t}^{(m)}$ is the ground-truth state. After the backpropagation step, the new tokens $\mathbf{M}_{\text{cur}}^{(m)}$ are detached from the computation graph and stored in the memory bank for the next iteration.

TABLE I
STATISTICAL OVERVIEW OF DATASETS USED IN PERSEUS EXPERIMENTS.

Datasets	# Features	# Timesteps	# Time Series	# States	State Duration	Avg State Duration
Pump V35	9	27,770 ∼ 40,810	40	41 ∼ 43	1 ∼ 3,290	543
Pump V36	9	26,185 ∼ 38,027	40	41 ∼ 43	1 ∼ 1,960	517
Pump V38	9	20,365 ∼ 30,300	40	42 ∼ 43	1 ∼ 1,820	407
USC-HAD	6	25,356 ∼ 56,251	70	12	600 ∼ 13,500	3,347
PAMAP2	9	8,477 ∼ 447,000	9	2 ∼ 13	1 ∼ 42,995	14,434
IndustryMG (Fine-Grained)	3	13,629 ∼ 45,010	17	4	10 ∼ 2,236	583
Pump V35 (2x Coarser)	———	Same as PumpV35	———	22	1 ∼ 3,570	771
Pump V36 (2x Coarser)	———	Same as PumpV36	———	21 ∼ 22	1 ∼ 2,085	719
Pump V38 (2x Coarser)	———	Same as PumpV38	———	22	1 ∼ 3,305	625
USC-HAD (2x Coarser)	———	Same as USC-HAD	———	6	1,700 ∼ 19,100	6,694
PAMAP2 (2x Coarser)	———	Same as PAMAP2	———	6	1 ∼ 50,198	16,997
IndustryMG (Coarse-Grained)	———	Same as IndustryMG (Fine-Grained)	———	2	18 ∼ 3,415	883
Pump V35 (4x Coarser)	———	Same as PumpV35	———	11	1 ∼ 3,855	1,171
Pump V36 (4x Coarser)	———	Same as PumpV36	———	11	1 ∼ 4,874	1,019
Pump V38 (4x Coarser)	———	Same as PumpV38	———	11	1 ∼ 4,154	933

IV. EXPERIMENTS

*a) **Datasets**:* We evaluate Perseus on six datasets, with their statistics summarized in Table I. The first group consists of publicly available benchmarks including **USC-HAD** [41], **PAMAP2** [42], and the **Pump** datasets (V35, V36, V38) [21]. USC-HAD records human daily activities using wearable accelerometer and gyroscope sensors, while PAMAP2 contains

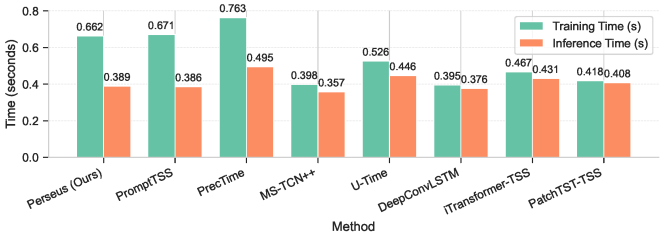


Fig. 4. Comparison of training and inference times per batch on the PAMAP2 dataset.

multimodal sensor data collected from subjects performing diverse physical activities. The Pump datasets are collected from hydraulic pump end-of-line (EoL) testing, each annotated with more than 40 operational states. These datasets were originally annotated with fine-grained states only, and we create multiple levels of granularity by systematically merging *neighboring states in temporal order*. For USC-HAD (12 states) and PAMAP2 (24 states), the relatively small number of classes allows only a $2\times$ coarser version. For Pump V35, V36, and V38, we construct both $2\times$ and $4\times$ coarser levels to capture broader operating phases. In contrast, **IndustryMG** is a proprietary industrial dataset with *naturally annotated* multi-granularity states, capturing both high-level production phases and fine-grained machine operations. Unlike the synthetic coarsening applied to the open datasets, IndustryMG directly provides real-world hierarchical annotations, making it an important benchmark for validating adaptive segmentation in practice. IndustryMG cannot be redistributed because the measurements were supplied under third-party confidentiality restrictions. No proprietary records are included in the public release. Code, preprocessing instructions, and the open-data pipeline are available at https://github.com/blacksnail789521/Perseus.

*b) **Metrics**:* To comprehensively evaluate segmentation performance, we adopt three commonly used metrics: Accuracy (ACC), Macro F1-score (MF1), and the Adjusted Rand Index (ARI). Accuracy measures the overall proportion of correctly predicted states across the sequence. Macro F1-score balances performance across classes by averaging the F1-score of each state, which is particularly important in time series with imbalanced state distributions. Adjusted Rand Index evaluates clustering consistency by comparing predicted and ground-truth state assignments while adjusting for chance, providing a robust measure of segmentation quality. Metrics are computed over the windowed timestamp instances produced by the evaluation loader: if a timestamp occurs in several overlapping windows, each occurrence is counted once per window rather than aggregated into one subsequence-level prediction. This overlap-weighted protocol is applied identically to every method and matches the training unit, but it is not a one-vote-per-unique-timestamp evaluation. We state this distinction to avoid implying otherwise. Boundary errors are therefore reflected through timestamp labels rather than a relaxed boundary tolerance.

*c) **Baselines**:* We compare Perseus against a broad range of state-of-the-art segmentation models. Throughout this evaluation, *stateless* refers specifically to interaction history: none of the baselines retains prompted feedback across inference windows. **PromptTSS** [20] represents the state-of-the-art in prompting-based segmentation. It incorporates label and boundary cues but operates under a local sliding-window assumption. While highly effective under uniform supervision, it lacks subsequence-level persistence, making it vulnerable to grouped supervision gaps. **PrecTime** [21] is a sequence-to-sequence architecture combining sliding windows with dense labeling, designed for precise industrial segmentation but limited to local context. **MS-TCN++** [22] applies multi-stage temporal convolutional layers with dilations to refine predictions and mitigate over-segmentation. **U-Time** [23] adapts the U-Net architecture for 1D time series, capturing both local and global temporal context within a fixed window. **DeepConvLSTM** [26] combines convolutional feature extraction with recurrent modeling for wearable activity recognition. Finally, we include *forecasting adapters* (**iTransformer-TSS** [43], **PatchTST-TSS** [40]), which are segmentation variants of strong forecasting models. While they capture long-range dependencies, they remain stateless with respect to user interaction history across inference passes.

Among all baselines, only Perseus and PromptTSS support prompting natively. For the remaining baselines, we use a post-hoc prompting wrapper: separate models are trained for each granularity level, each model predicts the full subsequence, and the candidate with the lowest mismatch on the available prompted timestamps is selected. No ground-truth labels outside the prompted positions are used during inference, and all methods receive the same prompt budget and locations. We also include a simple **LabelProp** sanity baseline: under the same hotspot prompt protocol, it copies the nearest declared-correct label prompt through the subsequence. Boundary prompts and incorrect-label prompts are counted in the budget but are not converted into oracle labels.

*d) **Experimental Setup: Grouped Sparse Supervision**:* A critical distinction of our evaluation is the supervision setting. Unlike prior works that evaluate under dense or uniform prompting assumptions, we use a *grouped sparse budget* and distinguish three protocols below. For the single-step experiment, the number of prompt cues is $\lfloor 0.05 L_s \rfloor$, i.e., a budget equal to 5% of the subsequence length rather than necessarily 5% distinct prompted timestamps. Prompts are not uniformly distributed: given $W = 8$ windows per subsequence, their timestamps are constrained to two *bursty* windows, leaving the remaining six windows entirely *unprompted*. This setup mimics realistic user behavior where annotators mark clusters of events and leave large gaps. Under this setting, we test whether models suffer from contextual amnesia in unprompted windows or can successfully retrieve supervision provided elsewhere in the subsequence. Prompt simulation draws label and boundary cues with equal probability and correct versus incorrect aspects with equal probability. For a declared-incorrect label cue, the simulator uniformly samples

TABLE II

SEGMENTATION PERFORMANCE UNDER THE SINGLE ITERATION INFERENCE SETTING. A BUDGET OF $\lfloor 0.05 L_s \rfloor$ PROMPT CUES IS PROVIDED ONCE PER SUBSEQUENCE. BEST RESULTS ARE IN **BOLD**, AND SECOND-BEST ARE UNDERLINED.

Dataset		USC-HAD		PAMAP2		IndustryMG		Pump V35		Pump V36		Pump V38	
Granularity		Single	Multiple	Single	Multiple	Single	Multiple	Single	Multiple	Single	Multiple	Single	Multiple
Perseus (Ours)	ACC	**92.70**	**95.56**	**94.94**	**96.96**	96.32	**89.68**	**88.96**	**92.34**	**91.61**	**91.83**	**85.55**	**88.95**
	MF1	**91.18**	**93.92**	**93.51**	**96.16**	79.50	**71.58**	**82.35**	**88.33**	**86.38**	**85.94**	**79.61**	**83.08**
	ARI	**87.65**	**91.75**	**89.32**	**92.35**	92.58	**79.23**	**86.03**	**86.99**	**89.69**	**87.14**	**78.53**	**80.97**
PromptTSS	ACC	80.53	60.91	51.90	53.54	94.79	71.60	70.93	41.47	77.01	41.69	72.33	41.98
	MF1	77.30	52.01	47.22	41.09	72.05	40.27	54.20	32.26	63.61	32.53	55.45	34.14
	ARI	68.23	39.60	31.09	11.55	89.24	55.24	62.06	20.74	68.55	20.96	64.30	21.06
PrecTime	ACC	72.48	51.21	44.19	44.00	96.46	77.10	62.05	31.38	60.46	31.71	63.93	33.02
	MF1	68.15	46.03	31.23	27.96	72.49	66.12	56.25	29.66	57.78	29.23	57.13	30.44
	ARI	58.35	30.80	13.23	13.55	92.57	67.37	59.81	17.65	50.26	16.36	55.96	17.69
MS-TCN++	ACC	31.41	26.29	43.00	43.82	96.75	76.97	54.18	28.39	41.71	23.08	37.99	21.98
	MF1	22.72	16.76	18.28	19.50	70.25	63.36	20.11	14.42	15.18	14.62	10.65	10.64
	ARI	19.06	14.28	9.10	19.83	**95.49**	68.46	59.19	18.75	38.94	13.28	28.91	10.55
U-Time	ACC	48.24	32.11	45.32	41.07	**97.69**	77.53	43.01	16.85	25.03	12.58	49.06	27.23
	MF1	47.21	25.99	31.38	22.43	90.07	63.99	23.03	11.99	12.17	7.06	23.14	18.01
	ARI	26.62	15.73	26.27	21.78	**95.49**	68.97	27.55	6.71	15.56	3.75	42.30	15.27
DeepConvLSTM	ACC	32.81	30.65	58.15	48.65	**97.69**	77.84	63.78	35.06	61.32	34.31	56.01	30.72
	MF1	34.23	27.39	47.86	29.54	**90.34**	66.17	48.13	28.78	46.52	31.02	31.94	24.39
	ARI	18.77	12.73	32.35	31.19	95.43	69.16	60.53	21.90	50.75	18.44	50.02	17.70
iTransformer-TSS	ACC	29.28	30.05	36.32	36.50	67.25	55.99	27.44	16.72	40.96	21.73	32.17	18.24
	MF1	16.46	15.21	8.12	7.89	29.60	32.31	11.80	9.39	15.72	12.80	11.09	10.19
	ARI	14.04	10.51	9.58	11.49	30.26	18.84	19.32	6.76	31.32	10.01	19.73	6.41
PatchTST-TSS	ACC	60.74	45.77	38.86	38.21	50.32	31.52	61.61	33.09	65.73	34.03	63.05	31.97
	MF1	51.35	38.33	23.95	11.92	27.11	19.83	42.70	26.01	43.39	26.74	40.12	23.56
	ARI	45.50	27.74	13.31	4.81	16.74	6.29	55.74	17.95	55.75	17.54	55.06	16.43

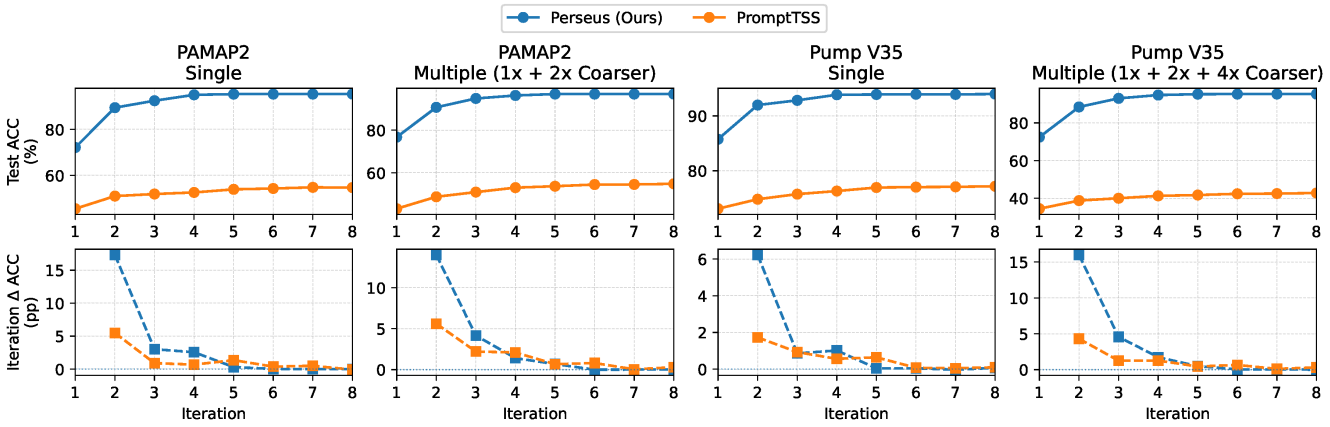


Fig. 5. Segmentation performance under the multiple-iteration inference setting, evaluated on two datasets (PAMAP2 and Pump V35), each with both single and multiple granularities of states. **Top row:** Test accuracy (ACC, %). **Bottom row:** Iteration Δ ACC (percentage points, pp).

one of the $K-1$ classes different from the ground-truth class. For an incorrect boundary cue, it flips the ground-truth boundary bit.

*e) **Implementation Details**:* We split each dataset chronologically into 70% training, 15% validation, and 15% testing. For subsequence construction, we use non-overlapping segments of length L_s, each further divided into $W = 8$ sliding windows of length T with stride S. We set $T = 256$ (or $T = 512$ for USC-HAD) with stride $S = T/4$. Perseus employs iterative prompting at the subsequence level. During training, each of $N_r = 8$ iterations adds $N_p = 4$ prompts, for 32 cumulative prompts. This training budget is distinct from the single-step 5% test budget. All models are trained with the AdamW optimizer (learning rate 1×10^{-4}).

A. Single Iteration Inference

In this setting, user feedback is provided once using the grouped budget of $\lfloor 0.05 L_s \rfloor$ prompt cues. This setup serves as a stress test for *state persistence*: can the model retain user intent across the six of eight windows (75%) that receive no prompt? The results are summarized in Table II.

*a) **Overcoming Contextual Amnesia**:* Under this grouped supervision, Perseus achieves an average accuracy

TABLE III
MATCHED LABEL-PROPAGATION SANITY CHECK ON INDUSTRIAL DATASETS. Δ ACC (PP) IS PERSEUS (OURS) ACC MINUS LABELPROP ACC. HIGHER IS BETTER FOR ALL METRICS.

Dataset	Gran.	LabelProp			Perseus (ours)			Δ ACC (pp)
		ACC	MF1	ARI	ACC	MF1	ARI	
Pump V36	Single	65.79	45.75	59.83	91.61	86.38	89.69	**+25.83**
Pump V36	Multi.	72.76	58.61	58.64	91.83	85.94	87.14	**+19.07**
Pump V38	Single	61.29	36.57	49.73	85.55	79.61	78.53	**+24.26**
Pump V38	Multi.	69.63	54.32	53.23	88.95	83.08	80.97	**+19.32**

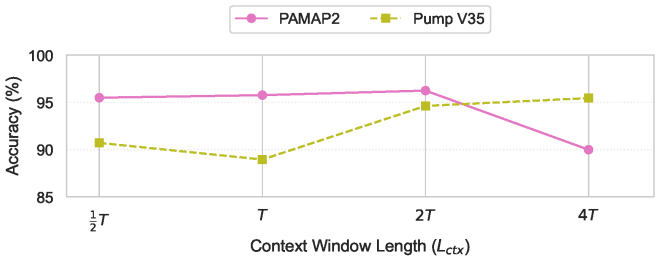


Fig. 6. Ablation study on the impact of context window length T_{ctx} on the segmentation accuracy of Perseus (PAMAP2 and PumpV35).

improvement of 54% (23% on single-granularity and 85% on multi-granularity datasets) compared to the best-performing neural baseline. Crucially, the performance of the stateless PromptTSS baseline drops significantly compared to its theoretical ceiling. This degradation highlights *contextual amnesia* in local prompt-based models: in unprompted windows (inter-burst gaps), PromptTSS cannot access supervision provided in neighboring windows. In contrast, Perseus utilizes its memory bank to bridge these gaps. By encoding the "burst" of prompts into persistent memory tokens, it allows unprompted windows to attend to the supervised history. The massive gain in multi-granularity tasks (117% improvement over neural baselines excluding PromptTSS) further validates that hierarchical state definitions require global context that only a stateful memory can provide.

*b) **Beyond Label Propagation**:* Table III evaluates whether a simple persistent-label rule is sufficient. LabelProp is intentionally strong for a state-machine baseline: it receives the same grouped prompt budget and copies the nearest usable correct-label prompt across unprompted regions. On Pump V36 and V38, Perseus improves ACC by 19.07–25.83 percentage points and also yields large MF1/ARI gains, showing that industrial multi-state transitions cannot be recovered by copying a single local label. This result should be interpreted with dataset structure in mind: on activity datasets such as PAMAP2 and USC-HAD, many test subsequences contain one long state, making label propagation competitive. For example, the median state run spans the full subsequence on PAMAP2 and USC-HAD, with 90.9% and 67.1% single-state subsequences in their single-granularity test splits. The Pump datasets contain shorter state runs and more transitions, making them a stricter test of whether memory can bridge unprompted state changes.

*c) **Efficiency**:* We further analyze training and inference time for all methods, as shown in Figure 4. Despite the addition of the memory bank, Perseus maintains a runtime profile highly similar to PromptTSS. This indicates that the cost of statefulness is modest under the evaluated sparse prompt budget. The Write/Read operations use compact token representations rather than high-dimensional feature maps, allowing Perseus to address contextual amnesia while retaining a runtime profile close to PromptTSS in this setting.

B. Iterative Inference

We next evaluate the dynamic capability of Perseus in an interactive setting using PAMAP2 and Pump V35. We limit the comparison to Perseus and PromptTSS, the only evaluated methods with native iterative prompt input. This is a separate fixed-count protocol: each of $N_r = 8$ iterations adds four new prompts, yielding 32 cumulative prompts rather than stopping at the single-step 5% budget. The results are illustrated in Figure 5.

*a) **State Accumulation vs. Stateless Re-computation**:* While both models improve with more feedback, the mechanism of improvement differs fundamentally. PromptTSS treats each iteration as a largely independent inference pass. It benefits from new prompts only because the local sliding windows happen to capture them. Perseus, however, treats iterations as *state accumulation*. Results show that Perseus achieves an average per-iteration accuracy gain (Δ ACC) of 2.66 percentage points, compared to just 1.19 for PromptTSS. The disparity is most critical in the early phase (Iterations 1–4), where Perseus achieves a massive 6.07 pp gain compared to 2.25 pp for the baseline. This confirms that Perseus is a more *data-efficient learner*: by retrieving previously stored tokens from the memory bank, it compounds the value of every new prompt, whereas the stateless baseline struggles to propagate sparse signals beyond their immediate context.

C. Ablation Study

*a) **Effect of Context Window Length**:* We first examine the effect of the context window length T_{ctx} used when encoding prompts into memory tokens. Experiments are conducted on PAMAP2 and Pump V35, where we vary T_{ctx} across four values: $\frac{1}{2}T$, T, $2T$, and $4T$. The results are shown in Figure 6.

We observe that increasing T_{ctx} generally improves segmentation performance, confirming that the model benefits from leveraging more surrounding time series context when generating memory tokens. However, the improvement is not permanent. On PAMAP2, performance begins to decrease when T_{ctx} reaches $4T$, suggesting that excessively long context windows may introduce noise or dilute the local temporal cues that are most relevant for prompt alignment. This indicates that while richer context is useful for constructing memory, it should remain within a reasonable range to balance local precision and global coverage.

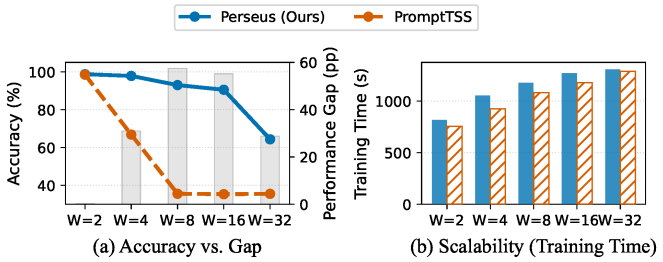


Fig. 7. Impact of subsequence length L_s on PAMAP2 (fixed budget). (a) Accuracy and the widening performance delta (Δ) highlight Perseus's superior robustness to signal dilution. (b) Training time shows favorable empirical scaling over the evaluated range.

*b) **Effect of Subsequence Length & Scalability**:* We next evaluate the model's robustness and scalability by varying the subsequence length L_s (containing $W \in \{2, 4, 8, 16, 32\}$ sliding windows). To simulate realistic interaction constraints, we adopt a *Fixed User Budget* protocol: we keep the total supervision budget fixed at 32 prompts (accumulated via 8 iterations with $N_p = 4$), concentrated within exactly two sliding windows regardless of the total sequence length. This creates a challenging scenario where supervision density dilutes as L_s increases: coverage drops from 100% (2 out of 2 windows) at $W = 2$ to merely 6.25% (2 out of 32 windows) at $W = 32$. The results are presented in Figure 7.

Robustness: As shown in Figure 7(a), PromptTSS, the strongest stateless baseline, collapses to random guessing ($\sim$35%) as early as $W = 8$ due to the overwhelming number of unprompted windows. In contrast, Perseus exhibits superior robustness, maintaining high accuracy far longer and yielding a massive performance gap (+57.5pp at $W = 8$). Even at the extreme horizon of $W = 32$, Perseus retains useful signal (64.33%) where the baseline fails completely, demonstrating that the persistent memory bank successfully bridges extended temporal gaps to retrieve user intent.

Scalability: Figure 7(b) reports the empirical runtime trend as subsequence length increases. Because the total dataset size is fixed, the number of processed timesteps remains approximately constant as the number of subsequences decreases ($M \times L_s \approx$ const). A model linear in L_s would therefore have roughly constant epoch time, aside from batching and memory interactions. As L_s increases by $16\times$ (from $W = 2$ to $W = 32$), training time per epoch increases from 819s to 1310s, which is sublinear in L_s over the evaluated range. This trend is consistent with avoiding quadratic growth, although it does not establish an asymptotic complexity bound.

V. Conclusion

In this work, we introduced Perseus, a framework that transitions interactive time series segmentation from synchronous window-level processing to *asynchronous state management*. We identified that stateless prompt-based models suffer from *contextual amnesia* when grouped supervision leaves large temporal gaps between user inputs. Perseus addresses this by decoupling supervision from inference via an asynchronous *Write-Read memory mechanism*. Specifically, grouped bursts of feedback are asynchronously encoded into a persistent memory bank (Write), which is then actively queried to guide predictions in unprompted windows (Read). This design creates a semantic bridge that preserves user intent across the entire timeline, ensuring consistent segmentation even when supervision is sparse and distant.

Extensive experiments on six wearable and industrial datasets confirm the value of statefulness under grouped supervision. Under a strict grouped budget (5% density), Perseus demonstrates superior robustness, achieving an 85% accuracy improvement over the evaluated stateless baselines in multi-granularity settings. Furthermore, in interactive sessions, Perseus exhibits higher data efficiency, yielding an average per-iteration gain of 2.66 percentage points compared to 1.19 for the stateless PromptTSS baseline. Matched label-propagation analysis further shows that simple state persistence can be competitive on long-duration activity sequences, but fails on industrial datasets with more frequent unprompted transitions. These results establish that contextual memory, rather than label copying alone, is important for handling realistic, bursty human-in-the-loop interactions.

This study has limitations. Prompts are simulated from clean annotations, and robustness to noisy or conflicting expert feedback deserves further study. The current memory bank uses simple accumulation. Future work can introduce explicit forgetting or conflict resolution. We also plan to integrate *active learning* to identify informative unprompted windows and reduce the burden on domain experts.

References

[1] U. Bermejo, A. Almeida, A. Bilbao-Jayo, and G. Azkune, "Embedding-based real-time change point detection with application to activity segmentation in smart home time series data," *Expert Systems with Applications*, vol. 185, p. 115641, 2021.

[2] Y. Fu, Z. Zhu, L. Liu, W. Zhan, T. He, H. Shen, J. Zhao, Y. Liu, H. Zhang, Z. Liu *et al.*, "Remote sensing time series analysis: A review of data and applications," *Journal of Remote Sensing*, vol. 4, p. 0285, 2024.

[3] C. Chang, C. Chiao-Tung, W.-Y. Wang, W.-C. Peng, and T.-F. Chen, "Self-supervised learning of disentangled representations for multivariate time-series," in *NeurIPS 2024 Workshop: Self-Supervised Learning-Theory and Practice*, 2024.

[4] L. C. de Jesus Jr, F. Fernández-Navarro, and M. Carbonero-Ruz, "Enhancing financial time series forecasting through topological data analysis," *Neural Computing and Applications*, vol. 37, no. 9, pp. 6527–6545, 2025.

[5] Z. Dong, X. Fan, and Z. Peng, "Fnspid: A comprehensive financial news dataset in time series," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 4918–4927.

[6] C. Chang, W.-Y. Wang, W.-C. Peng, T.-F. Chen, and S. Samtani, "Align and fine-tune: Enhancing LLMs for time-series forecasting," in *NeurIPS Workshop on Time Series in the Age of Large Models*, 2024. [Online]. Available: https://openreview.net/forum?id=AaRCmJieG4

[7] R. Komitova, D. Raabe, R. Rein, and D. Memmert, "Time series data mining for sport data: A review," *Journal homepage: http://iacss. org/index. php? id*, vol. 21, no. 2, 2022.

[8] A. T. Alhasani, "Unsupervised clustering of multivariate sports activity data using k-means: A study on the sport data multivariate time series dataset," *Journal of Transactions in Systems Engineering*, vol. 3, no. 2, pp. 367–381, 2025.

[9] C. Chang, C.-T. Chan, W.-Y. Wang, W.-C. Peng, and T.-F. Chen, "Timedrl: Disentangled representation learning for multivariate time-series," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 2024, pp. 625–638.

[10] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM SigKDD Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, 2011.

[11] C. Wang, K. Wu, T. Zhou, and Z. Cai, "Time2state: An unsupervised framework for inferring the latent states in time series data," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–18, 2023.

[12] D. Hallac, S. Vare, S. Boyd, and J. Leskovec, "Toeplitz inverse covariance-based clustering of multivariate time series data," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 215–223.

[13] C. Chang, Y. Shi, D. Cao, W. Yang, J. Hwang, H. Wang, J. Pang, W. Wang, Y. Liu, W.-C. Peng *et al.*, "A survey of reasoning and agentic systems in time series with large language models," *arXiv preprint arXiv:2509.11575*, 2025.

[14] M.-C. Lo, C. Chang, and W.-C. Peng, "Text2freq: Learning series patterns from text via frequency domain," in *NeurIPS Workshop on Time Series in the Age of Large Models*, 2024. [Online]. Available: https://openreview.net/forum?id=Pi6sA1MSSr

[15] T.-Y. Ou, C. Chang, and W.-C. Peng, "Coke: Causal discovery with chronological order and expert knowledge in high proportion of missing manufacturing data," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, ser. CIKM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 4803–4810. [Online]. Available: https://doi.org/10.1145/3627673.3680083

[16] C. Chang, J. Hwang, Y. Shi, H. Wang, W.-C. Peng, T.-F. Chen, and W. Wang, "Time-imm: A dataset and benchmark for irregular multimodal multivariate time series," *arXiv preprint arXiv:2506.10412*, 2025.

[17] C. Chang, W.-Y. Wang, W.-C. Peng, and T.-F. Chen, "Llm4ts: Aligning pre-trained llms as data-efficient time-series forecasters," *ACM Trans. Intell. Syst. Technol.*, vol. 16, no. 3, Apr. 2025. [Online]. Available: https://doi.org/10.1145/3719207

[18] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, "Segment anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4015–4026.

[19] C.-M. Lin, C. Chang, W.-Y. Wang, K.-D. Wang, and W.-C. Peng, "Root cause analysis in microservice using neural granger causal discovery," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 1, pp. 206–213, Mar. 2024. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/27772

[20] C. Chang, M.-C. Lo, W.-C. Peng, and T.-F. Chen, "Promptts: A prompting-based approach for interactive multi-granularity time series segmentation," in *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, ser. CIKM '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 209–220. [Online]. Available: https://doi.org/10.1145/3746252.3761142

[21] S. Gaugel and M. Reichert, "Prectime: A deep learning architecture for precise time series segmentation in industrial manufacturing operations," *Engineering Applications of Artificial Intelligence*, vol. 122, p. 106078, 2023.

[22] S. Li, Y. A. Farha, Y. Liu, M.-M. Cheng, and J. Gall, "Ms-tcn++: Multi-stage temporal convolutional network for action segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 6, pp. 6647–6658, 2020.

[23] M. Perslev, M. Jensen, S. Darkner, P. J. Jennum, and C. Igel, "U-time: A fully convolutional network for time series segmentation applied to sleep staging," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[24] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson *et al.*, "Sam 2: Segment anything in images and videos," *arXiv preprint arXiv:2408.00714*, 2024.

[25] J. A. Fails and D. R. Olsen, Jr., "Interactive machine learning," in *Proceedings of the 8th International Conference on Intelligent User Interfaces*. New York, NY, USA: Association for Computing Machinery, 2003, pp. 39–45. [Online]. Available: https://doi.org/10.1145/604045.604056

[26] F. J. Ordóñez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016.

[27] Y. Sun, Z. Xie, D. Chen, E. Eldele, and Q. Hu, "Hierarchical classification auxiliary network for time series forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 19, 2025, pp. 20 743–20 751.

[28] A. Narayan, F. A. Reyes, M. Ren, and Y. Haoyong, "Real-time hierarchical classification of time series data for locomotion mode detection," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 4, pp. 1749–1760, 2021.

[29] D. Gkatzia, H. Hastie, and O. Lemon, "Comparing multi-label classification with reinforcement learning for summarisation of time-series data," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014, pp. 1231–1240.

[30] C. Nalmpantis and D. Vrakas, "On time series representations for multi-label nilm," *Neural Computing and Applications*, vol. 32, pp. 17 275–17 290, 2020.

[31] W. Liu, H. Wang, X. Shen, and I. W. Tsang, "The emerging trends of multi-label learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 11, pp. 7955–7974, 2021.

[32] M. Chen, L. Shen, H. Fu, Z. Li, J. Sun, and C. Liu, "Calibration of time-series forecasting: Detecting and adapting context-driven distribution shift," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 341–352.

[33] L. Meegahapola, H. Hassoune, and D. Gatica-Perez, "M3bat: Unsupervised domain adaptation for multimodal mobile sensing with multi-branch adversarial training," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 8, no. 2, pp. 1–30, 2024.

[34] H. He, O. Queen, T. Koker, C. Cuevas, T. Tsiligkaridis, and M. Zitnik, "Domain adaptation for time series under feature and label shifts," in *International conference on machine learning*. PMLR, 2023, pp. 12 746–12 774.

[35] B. Settles, "Active learning literature survey," Department of Computer Sciences, University of Wisconsin–Madison, Tech. Rep. TR 1648, 2009.

[36] F. Peng, Q. Luo, and L. M. Ni, "Acts: an active learning method for time series classification," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 175–178.

[37] E. Eldele, M. Ragab, Z. Chen, M. Wu, C.-K. Kwoh, and X. Li, "Label-efficient time series representation learning: A review," *IEEE Transactions on Artificial Intelligence*, 2024.

[38] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[39] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[40] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, "A time series is worth 64 words: Long-term forecasting with transformers," in *ICLR*. OpenReview.net, 2023.

[41] M. Zhang and A. A. Sawchuk, "Usc-had: A daily activity dataset for ubiquitous activity recognition using wearable sensors," in *Proceedings of the 2012 ACM conference on ubiquitous computing*, 2012, pp. 1036–1043.

[42] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *2012 16th International Symposium on Wearable Computers*. IEEE, 2012, pp. 108–109.

[43] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, "itransformer: Inverted transformers are effective for time series forecasting," in *International Conference on Learning Representations (ICLR)*, 2024.