MemPromptTSS: Persistent Prompt Memory for Iterative Multi-Granularity Time Series State Segmentation

Ching Chang*†
Computer Science
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Ming-Chih Lo
Computer Science
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Chiao-Tung Chan
Electrical and Control Engineering
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Wen-Chih Peng
Computer Science
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Tien-Fu Chen
Computer Science
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

Abstract

Web platforms, mobile applications, and connected sensing systems generate multivariate time series with states at multiple levels of granularity, from coarse regimes to fine-grained events. Effective segmentation in these settings requires integrating across granularities while supporting iterative refinement through sparse prompt signals, which provide a compact mechanism for injecting domain knowledge. Yet existing prompting approaches for time series segmentation operate only within local contexts, so the effect of a prompt quickly fades and cannot guide predictions across the entire sequence. To overcome this limitation, we propose MemPromptTSS, a framework for iterative multi-granularity segmentation that introduces persistent prompt memory. A memory encoder transforms prompts and their surrounding subsequences into memory tokens stored in a bank. This persistent memory enables each new prediction to condition not only on local cues but also on all prompts accumulated across iterations, ensuring their influence persists across the entire sequence. Experiments on six datasets covering wearable sensing and industrial monitoring show that MemPromptTSS achieves 23% and 85% accuracy improvements over the best baseline in single- and multi-granularity segmentation under single iteration inference, and provides stronger refinement in iterative inference with average per-iteration gains of 2.66 percentage points compared to 1.19 for PromptTSS. These results highlight the importance of persistent memory for prompt-guided segmentation, establishing MemPromptTSS as a practical and effective framework for real-world applications.

CCS Concepts

• **Mathematics of computing → Time series analysis**; • **Human-centered computing → Interactive systems and tools**; • **Computing methodologies → Supervised learning**.

Keywords

Time Series Segmentation, Interactive Segmentation, Persistent Memory, Prompting, Multiple Granularities

*Correspondence to: Ching Chang <blacksnail789521.cs10@nycu.edu.tw>
†Also with GoEdge.ai.

1 Introduction

Web platforms, mobile applications, and IoT services continuously generate rich multivariate time series, capturing evolving states of users, devices, and systems. Examples include activity recognition in smart homes [2, 4, 14], market monitoring in financial platforms [9, 11, 12], and real-time performance tracking in sports analytics [1, 3, 20]. These applications increasingly rely on interactive labeling tools where practitioners can provide sparse feedback, such as marking a few states or specifying limited boundaries [8, 19, 23]. The central challenge is to make this small amount of user input propagate effectively, so that minimal manual effort can guide segmentation across long and complex sequences.

Time series states often appear at multiple levels of granularity, ranging from coarse system regimes to fine-grained events [7, 17, 21, 39]. For example, in smart home monitoring, coarse states such as occupant presence or absence coexist with fine-grained activities such as cooking or exercising [2, 14]. In financial platforms, long-term market cycles such as bull and bear trends overlap with short-lived volatility spikes [11, 26]. Similarly, in sports analytics, one may track overall match phases while also capturing detailed player movements [20]. Accurately capturing both coarse and fine patterns is crucial, since downstream Web applications such as personalized services, anomaly detection, and risk analysis depend on understanding how these levels interact [5, 32]. However, achieving reliable segmentation across multiple granularities requires that user guidance extend beyond local regions and remain consistent throughout the sequence.

The first limitation is that prompt-guided segmentation approaches typically apply user input only within the immediate region where it is provided [6, 19]. When a user marks a small set of states or boundaries, the effect remains confined to the local context and quickly fades outside that area. Consequently, most of the sequence is segmented without leveraging user guidance, limiting the efficiency of interactive labeling where sparse prompts should drive large-scale improvements [36].

The second limitation is that predictions across different regions of a sequence are made independently, without mechanisms for global consistency [6]. This independence produces fragmented and sometimes contradictory state assignments when outputs are

assembled across the full sequence [35]. Such incoherence is particularly problematic in interactive Web applications, where users expect that providing a small number of corrections will yield reliable segmentation across the entire dataset.

To address these limitations, we propose **MemPromptTSS**, a new framework that introduces *persistent prompt memory* for interactive multi-granularity segmentation. For the first limitation, MemPromptTSS encodes each prompt together with its surrounding subsequence into a memory token stored in a dedicated memory bank, ensuring that the effect of a prompt persists across iterations rather than vanishing locally. For the second limitation, all subsequent predictions are conditioned on the entire bank of accumulated prompts, so user input provided at any point influences the whole sequence, enforcing global consistency. MemPromptTSS supports both label prompts, which provide contextual state annotations, and boundary prompts, which indicate transitions between states. By combining persistent memory with iterative refinement, the model propagates sparse feedback throughout long sequences, making it effective for mining and analyzing complex Web time series.

In summary, our contributions are as follows:

- **Persistent Prompt Memory.** We introduce MemPromptTSS, the first framework that preserves user prompts across iterations, directly addressing the problem of locally fading guidance.
- **Global Consistency with Iterative Refinement.** By conditioning predictions on all accumulated prompts in memory, MemPromptTSS resolves fragmented, inconsistent outputs and ensures coherence across entire sequences.
- **Context-Enriched Prompt Encoding.** We design a memory encoder that fuses each prompt with its local subsequence, producing memory tokens that carry both label and boundary information for long-horizon influence.
- **Comprehensive Evaluation.** On six datasets from wearable sensing and industrial monitoring, MemPromptTSS achieves 23% and 85% accuracy improvements over the best baseline in single- and multi-granularity segmentation under single iteration inference, respectively. In iterative inference, MemPromptTSS provides stronger refinement capability, with average per-iteration gains of 2.66 percentage points compared to 1.19 for PromptTSS.

2 Related Work

2.1 Time Series Segmentation

Time series segmentation has been studied extensively, with methods ranging from classical heuristics to modern deep learning architectures. Early unsupervised approaches relied on clustering and statistical models, which provided simple segmentations but lacked accuracy and robustness for complex temporal structures. Supervised neural methods have since become dominant because they deliver higher accuracy, capture both local and long-range dependencies, and can be trained end-to-end for segmentation objectives as labeled datasets and computational resources have become more widely available. DeepConvLSTM [31] integrates convolutional and recurrent layers to capture spatial and temporal dependencies, U-Time [34] adapts a U-Net-like structure to enable both local and global context modeling, MS-TCN++ [22] refines predictions through multi-stage temporal convolutions to mitigate over-segmentation, and PrecTime [15] combines sliding windows with dense labeling for precise industrial segmentation. More recently, PromptTSS [6] extended this line of work by introducing prompting into segmentation, showing that sparse user input can guide predictions and enable multi-granularity modeling. Despite these advances, current models remain limited in their ability to maintain consistency across entire sequences, which is essential for interactive applications such as mining long user activity logs or Web-of-Things sensor streams.

Beyond segmentation-specific methods, several related paradigms explore partial aspects of adaptability. Hierarchical and multi-label classification frameworks encourage consistency across label levels, but they are designed for static datasets rather than dynamic time series [16, 24, 28, 29, 38]. Domain adaptation techniques aim to improve robustness under distribution shift, yet they do not provide mechanisms for incorporating user feedback during inference [10, 18, 27]. Active learning reduces annotation cost by selecting informative samples, but it usually requires retraining, which is impractical in interactive settings [13, 33, 37]. Taken together, these directions highlight the gap: existing approaches offer either granularity, robustness, or efficiency in isolation, but none provide a unified way to incorporate sparse user input, propagate it across long sequences, and maintain consistency at multiple granularities.

2.2 Prompting and Interactive Segmentation

Prompting has become a powerful mechanism for guiding models with lightweight user input. In natural language processing, large language models adapt to new tasks via textual prompts [40, 41], while in computer vision, interactive segmentation systems such as Segment Anything [19, 35] show how simple cues like clicks or boxes can steer predictions in real time. These successes illustrate the broader promise of prompting as a way to integrate human feedback directly into the inference process.

In time series segmentation, prompting is still in its early stages. Existing approaches have begun to accept sparse cues such as label or boundary prompts during inference, enabling limited user guidance without retraining [6]. However, these signals are applied only within local regions of the sequence, and their effect disappears once the model moves further along the timeline. This locality restricts their usefulness in interactive labeling tools, where the practical goal is for a small number of prompts to propagate broadly and maintain consistency across the full sequence, especially in Web-facing applications that analyze mobile, IoT, or financial platform streams.

Our work builds on these insights by introducing a persistent memory mechanism that stores encoded prompts together with their local context and makes them available to all subsequent predictions. Through this design, interactive guidance extends beyond local neighborhoods, achieving global coherence and supporting iterative multi-granularity segmentation of long time series.

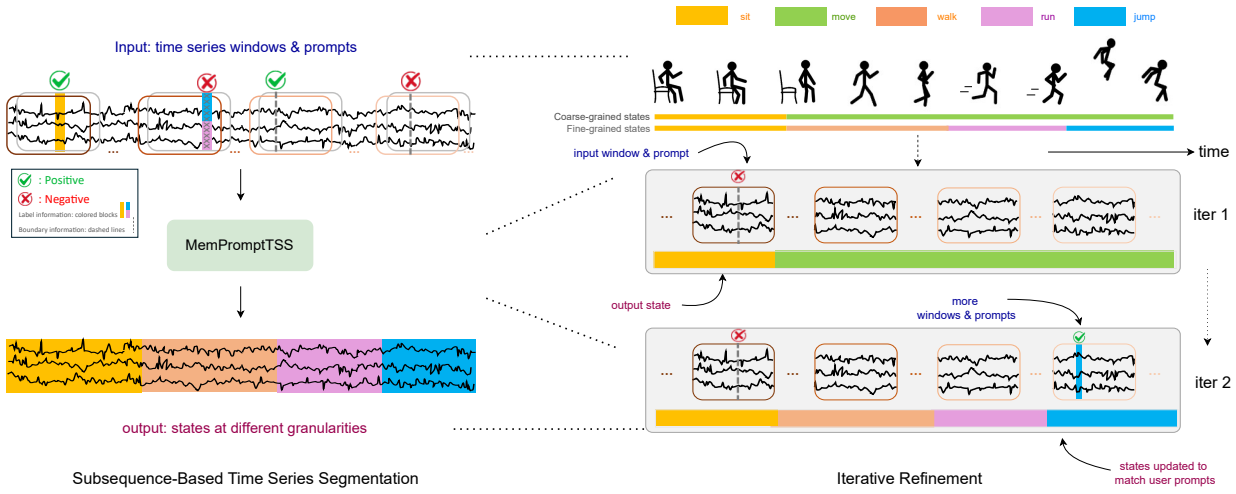

Figure 1: Problem setup and iterative refinement in MemPromptTSS. On the left, segmentation is performed over subsequences rather than a single sliding window, with prompts provided as label or boundary cues. On the right, iterative refinement illustrates how additional prompts guide the model to update predictions across multiple granularities, from coarse categories (e.g., move, sit) to fine actions (e.g., walk, run, jump).

3 Methodology

4 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:

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

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 and provides localized supervision.

The goal of time series segmentation is to learn a mapping that, given a subsequence $\mathbf{x}^{(m)}$ together with a set of prompts, predicts its corresponding state sequence $\mathbf{s}^{(m)}$. Figure 1 (left) illustrates this problem setup, where segmentation is performed on subsequences rather than individual sliding windows, with sparse prompts serving as supervision.

4.1 Model Architecture

Our framework consists of five main components: the time series encoder, the prompt encoder, the memory encoder, the memory bank, and the state decoder. Together, these modules enable the model to integrate raw time series data with sparse prompts, persist prompt information across iterations, and produce accurate state predictions. Figure 2 illustrates the overall architecture.

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* [30, 35], 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:

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

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

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.

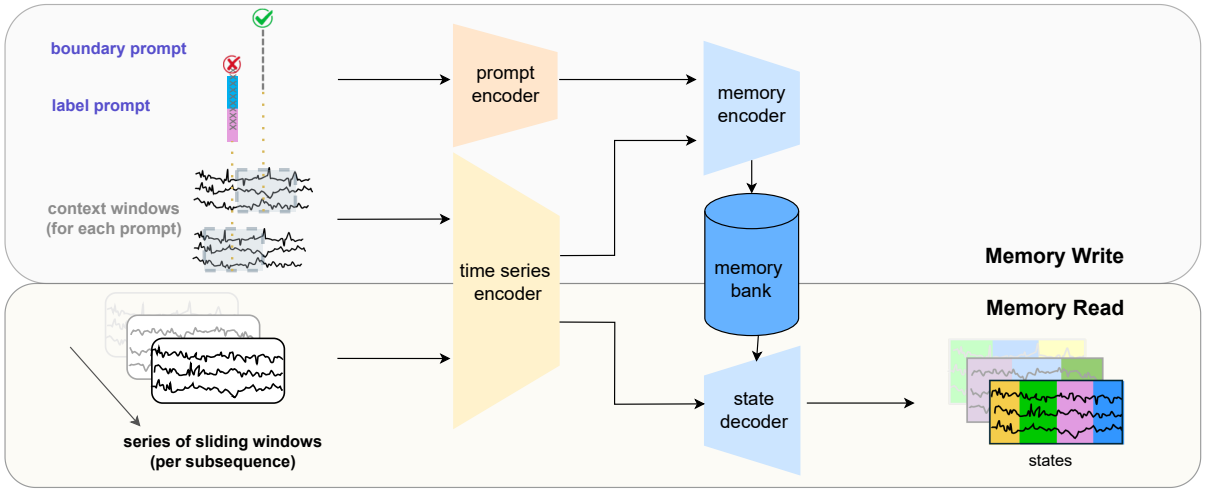

Figure 2: Overview of the MemPromptTSS framework. At each iteration, prompts are encoded and written into a per-subsequence memory bank (Memory Write). For every window, the state decoder integrates time series embeddings with memory tokens to produce predictions (Memory Read). Segmentation quality improves progressively as more prompts are provided across iterations.

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). This disambiguation ensures that the model can effectively interpret heterogeneous supervision signals. The final prompt embedding is expressed as

$$\mathbf{z}_p = f_p(p) \in \mathbb{R}^D. \tag{3}$$

Memory Encoder. The memory encoder f_m transforms each prompt into a *memory token* that can be stored in the memory bank and reused across iterations. Its purpose is to combine 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 memory token captures both the prompt supervision at timestamp t_c and the nearby temporal evidence provided by its context window. It is later written into the memory bank, enabling the model to accumulate prompt knowledge progressively over multiple iterations.

Memory Bank. The memory bank stores memory tokens accumulated across iterations. 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. Here, N_p is the number of prompts sampled in the current iteration, N_{old} is the number of tokens accumulated from all previous iterations, and N_{all} is the total number of tokens available for use in the current iteration.

By persisting across iterations, the memory bank enables the model to accumulate knowledge from multiple prompts, supporting long-term refinement of segmentation predictions. If an upper capacity limit is imposed, the bank follows a first-in-first-out (FIFO) replacement policy.

State Decoder. The state decoder g_s integrates the encoded time series tokens and the memory tokens retrieved from the memory bank to produce per-timestep predictions. We employ a *Two-Way Transformer* design consisting of the following steps: (1) self-attention over time series tokens, (2) cross-attention from time series to memory tokens, (3) self-attention among memory tokens, (4) cross-attention from memory tokens back to time series tokens, and (5) feed-forward networks applied to both streams.

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.

4.2 Iterative Training and Evaluation

MemPromptTSS is designed as an interactive framework where user-provided prompts guide segmentation across multiple iterations. This iterative strategy has two main benefits: (1) it allows the model to progressively incorporate increasing levels of user supervision, and (2) it enables long-term accumulation of prompt knowledge within the memory bank, which provides consistency across windows and iterations. By unifying training and evaluation under the same iterative procedure, MemPromptTSS directly models real-world usage, where segmentation quality improves as additional prompts are introduced (see Figure 1, right).

At each iteration r, the model performs two key operations in sequence: *Memory Write* followed by *Memory Read*. In the memory write step, the prompts sampled in the current iteration are stored into the memory bank at the subsequence level, allowing supervision to accumulate across iterations. In the memory read step, both the time series embeddings and the accumulated memory tokens (from previous and current iterations) are used to generate patch-level logits for every sliding window in the subsequence, which are then de-patchified (overlap-averaged) back to per-timestep predictions. This write–then–read process is repeated over multiple iterations, enabling the model to progressively refine segmentation as more prompts are provided.

Memory Write. At the beginning of iteration r, each subsequence in the batch samples N_p prompts from its ground-truth labels. Unlike sliding windows, which are processed individually during the read step, prompt sampling and memory writing are performed once per subsequence. For subsequence m, the sampled prompts are first encoded by the prompt encoder f_p, then fused with local context by the memory encoder f_m, resulting in new memory tokens:

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

where $\mathbf{M}_{\text{cur}}^{(m)} \in \mathbb{R}^{N_p \times D}$ denotes the new memory tokens for the subsequence. After encoding all prompts of the current iteration, the memory bank is updated by concatenating the new memory tokens with those from previous iterations:

$$\mathbf{M}_{\text{all}}^{(m)} = [\mathbf{M}_{\text{old}}^{(m)}; \mathbf{M}_{\text{cur}}^{(m)}], \quad N_{\text{all}} = N_{\text{old}} + N_p. \tag{8}$$

Memory Read. In contrast to the subsequence-level memory write, the read step is performed for every sliding window within the subsequence. For each window $\mathbf{w}_j^{(m)}$, the time series encoder produces embeddings $\mathbf{z}_{x,j}^{(m)}$, which are combined with the subsequence's memory bank through the state decoder g_s:

$$\hat{\mathbf{y}}_j^{(m)} = g_s(\mathbf{z}_{x,j}^{(m)}, \mathbf{M}_{\text{all}}^{(m)}), \tag{9}$$

Table 1: Statistical overview of datasets used in MemPromptTSS 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

where $\hat{\mathbf{y}}_j^{(m)} \in \mathbb{R}^{T \times K}$ are the predicted per-timestep state probabilities after de-patchification. The model is optimized using cross-entropy loss averaged over all windows and timesteps in the iteration:

$$\mathcal{L}^{(r)} = \frac{1}{W \cdot T} \sum_{j=1}^{W} \sum_{t=1}^{T} \mathcal{L}_{CE}(s_{j,t}^{(m)}, \hat{y}_{j,t}^{(m)}), \tag{10}$$

where $s_{j,t}^{(m)}$ is the ground-truth state at timestamp t in window j of subsequence m. After loss accumulation, a single optimizer step is applied, and the new tokens $\mathbf{M}_{\text{cur}}^{(m)}$ are detached and stored in the memory bank for subsequent iterations.

5 Experiments

Datasets. We evaluate MemPromptTSS on six datasets, with their statistics summarized in Table 1. The first group consists of publicly available benchmarks including **USC-HAD** [42], **PAMAP2** [36], and the **Pump** datasets (V35, V36, V38) [15]. USC-HAD records human daily activities using wearable accelerometer and gyroscope sensors, while PAMAP2 contains 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× coarser version; for Pump V35, V36, and V38, we construct both 2× and 4× 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.

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

Table 2: Segmentation performance under the single iteration inference setting. Prompts covering 5% of timestamps are 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
MemPromptTSS	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

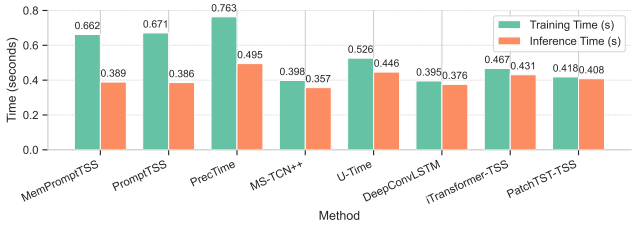

Figure 3: Comparison of training and inference times per batch on the PAMAP2 dataset.

Index evaluates clustering consistency by comparing predicted and ground-truth state assignments while adjusting for chance, providing a robust measure of segmentation quality. Together, these metrics capture complementary aspects of segmentation, ensuring fair evaluation across datasets with varying state counts and granularity levels.

Baselines. We compare MemPromptTSS against a broad range of state-of-the-art segmentation models. **PromptTSS** [6] is a prompting-based segmentation framework that enforces global consistency across sliding windows by incorporating label and boundary cues. However, it does not maintain subsequence-level memory across iterations, which limits its ability to refine predictions in an interactive setting. **PrecTime** [15] is a sequence-to-sequence architecture designed for precise segmentation in industrial settings. **MS-TCN++** [22] applies multi-stage temporal convolutional layers with dilations to refine predictions and mitigate over-segmentation. **U-Time** [34] adapts the U-Net architecture for 1D time series, capturing both local and global temporal context. **DeepConvLSTM** [31] combines convolutional feature extraction with recurrent modeling to handle multimodal wearable activity recognition. We also adapt strong forecasting models, **iTransformer** [25] and **PatchTST** [30], into segmentation variants denoted as iTransformer-TSS and PatchTST-TSS. Although originally built for forecasting, their ability to capture long-range temporal dependencies makes them suitable for segmentation after replacing the forecasting head with a classification layer.

Among all baselines, only MemPromptTSS and PromptTSS support prompting natively. For fair comparison, we equip all other baselines with a post-hoc prompting wrapper: we train separate models at different levels of state granularity and select predictions that best align with the provided prompts during inference. While this enables coarse-to-fine evaluation, it does not provide real-time prompt-aware refinement as in MemPromptTSS or PromptTSS.

Implementation Details. We split each dataset chronologically into 70% training, 15% validation, and 15% testing, ensuring that temporal order is preserved and no future information leaks into training. 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 $T = 256$ for most datasets and $T = 512$ for USC-HAD, which requires longer context due to its extended activity durations. Following prior work, S is chosen as one quarter of T (i.e., $S = 64$ when $T = 256$, $S = 128$ when $T = 512$), balancing efficiency and temporal coverage.

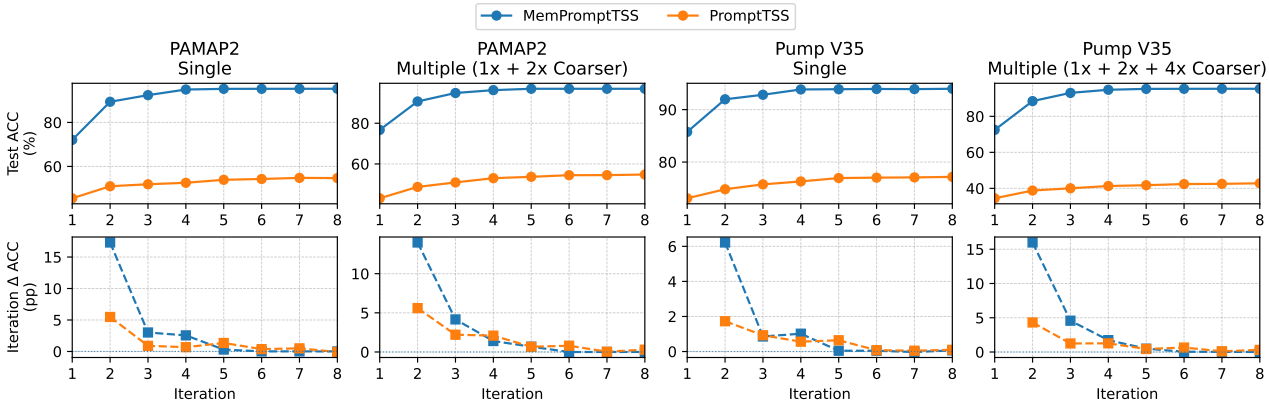

Figure 4: Segmentation performance under the multiple-iteration inference setting, evaluated on two datasets (PAMAP2 and Pump V35), each with both single and multiple granularities of states. Top row: Test accuracy (ACC, %). Bottom row: Iteration Δ ACC (percentage points, pp).

MemPromptTSS employs iterative prompting at the subsequence level. In each iteration, a fixed budget of $N_p = 4$ prompts (label or boundary) is sampled and encoded into memory tokens using the Memory Encoder, then stored in the subsequence's memory bank. We set the context length equal to the window length ($T_{\text{ctx}} = T$). Across *all* experiments, we target a total prompt density of 5% of timestamps per subsequence; for iterative settings, prompts are allocated across iterations such that the *cumulative* coverage at the end of the iterations equals 5%. Given $W = 8$ windows per subsequence, prompts are placed in exactly two windows, leaving the other six windows unprompted to reflect realistic user-interaction constraints. During training, we use $N_r = 8$ iterations, adding $N_p = 4$ prompts each iteration while enforcing the above cumulative 5% coverage; the same protocol is followed at inference.

We train all models with the AdamW optimizer (learning rate 1×10^{-4}, weight decay 0.01) and gradient clipping at 1.0. The time series encoder uses three Transformer layers, and the state decoder includes six Two-Way attention blocks with hidden dimension $D = 128$ and 2 attention heads. A patch length of 16 and stride of 8 are applied for tokenization, with overlap-averaging used to depatchify predictions. We apply a dropout rate of 0.1, and all models are trained using early stopping to ensure sufficient convergence. Experiments are conducted on an NVIDIA RTX 3070 GPU with mixed precision training enabled.

5.1 Single Iteration Inference

In the first setting, we evaluate single iteration inference, where all prompts are provided at once following the standard setup described in the implementation details. Each subsequence receives 5% prompt coverage, concentrated in only a subset of sliding windows. The results are summarized in Table 2.

Under this setting, MemPromptTSS achieves an average accuracy improvement of 54% (23% on single-granularity and 85% on multi-granularity datasets) compared to the best-performing baseline across all datasets. If PromptTSS is excluded from the comparison, the margin grows further to 75% (34% on single-granularity and 117% on multi-granularity datasets). The much larger gains in the multi-granularity case demonstrate that prompting together with persistent memory is especially important when handling hierarchical state structures, where limited supervision must propagate across different levels of granularity. By storing subsequence-level memory tokens, MemPromptTSS can spread the influence of sparse prompts across all sliding windows, whereas baselines only improve in windows that directly contain prompts. This persistence is crucial for ensuring that minimal user input can scale effectively across the entire subsequence, enabling more reliable segmentation with sparse supervision.

We further analyze training and inference time for all methods, as shown in Figure 3. MemPromptTSS and PromptTSS exhibit very similar runtime profiles, indicating that the addition of the memory component does not introduce significant overhead. Compared to other baselines, MemPromptTSS remain highly competitive due to the patching mechanism in the time series encoder, which reduces computational complexity while maintaining segmentation accuracy.

5.2 Iterative Inference

We next evaluate iterative inference, where prompts are provided progressively across multiple iterations. In this setting, we focus on PAMAP2 and Pump V35, and compare only MemPromptTSS and PromptTSS since these two methods are the only ones that natively support iterative refinement. At each iteration, four new prompts are sampled, and by the end of the eighth iteration the cumulative coverage reaches approximately 5% of timestamps per subsequence. The results are illustrated in Figure 4, showing accuracy from iteration 1 through iteration 8.

We observe that accuracy improves steadily as more prompts are added, confirming the benefit of iterative refinement. The improvement becomes marginal after around iteration 4, which is consistent with the design of prompting, as prompts are inherently sparse and not intended to scale indefinitely. This indicates that near-optimal performance can already be achieved well before the 5% budget is reached.

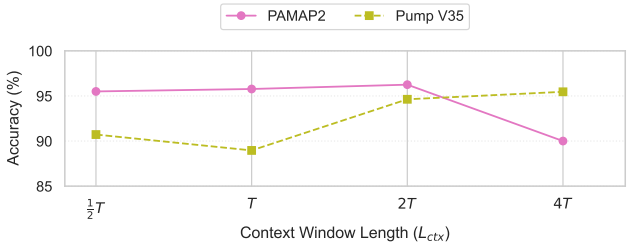

Figure 5: Ablation study on the impact of context window length T_{ctx} on segmentation accuracy for PAMAP2 and PumpV35.

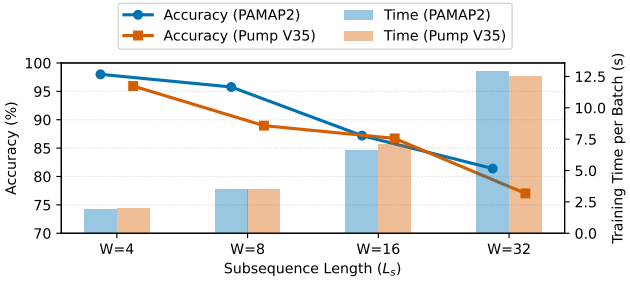

Figure 6: Ablation study on the impact of subsequence length L_s on segmentation accuracy and training time for PAMAP2 and PumpV35.

Directly comparing accuracy curves can be misleading, since MemPromptTSS and PromptTSS differ substantially in their overall performance. To better highlight refinement capability, we examine the absolute gain in accuracy per iteration. Across all eight iterations, MemPromptTSS achieves an average improvement that is 1.47 percentage points higher than PromptTSS (2.66 vs. 1.19). When focusing on the first four iterations, where most of the improvement occurs, this gap widens to 3.82 percentage points (6.07 vs. 2.25). These results highlight the importance of persistent memory: by storing and reusing subsequence-level tokens across iterations, MemPromptTSS achieves stronger global consistency and more effective refinement than PromptTSS, which operates only at the sliding-window level.

5.3 Ablation Study

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 5.

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.

Effect of Subsequence Length. We next study the effect of subsequence length L_s, which directly determines the number of sliding windows per subsequence. Experiments are conducted on PAMAP2 and Pump V35, where we adjust L_s so that each subsequence contains 4, 8, 16, or 32 sliding windows, while keeping the prompt density fixed at 5% of timestamps. The results are presented in Figure 6.

We observe that increasing the subsequence length leads to a noticeable drop in accuracy and a steady increase in training time per batch. This trend indicates that although the memory bank stores all prompt information, the model's ability to effectively leverage these prompts diminishes when subsequences become very long. As the number of windows grows, memory tokens must cover a broader temporal range, making it harder to maintain consistent guidance across the entire subsequence. These findings suggest that while persistent memory improves robustness under moderate subsequence lengths, scaling to very long subsequences remains challenging and calls for more efficient mechanisms for memory utilization.

6 Conclusion

In this work, we introduced MemPromptTSS, a prompting-based framework for time series segmentation with persistent memory. Our method extends prior prompting approaches by incorporating a memory mechanism that operates at the subsequence level. Specifically, prompts are first encoded together with local time series context through a memory write operation, then stored as memory tokens that persist across iterations. A memory read mechanism then fuses these tokens with sliding-window representations, allowing sparse prompts to influence all windows within a subsequence. This design enables MemPromptTSS to refine predictions iteratively and maintain consistency across different levels of granularity.

Extensive experiments on six datasets covering both wearable sensing and industrial monitoring show that MemPromptTSS consistently outperforms state-of-the-art baselines. In the single iteration inference setting, MemPromptTSS achieves 23% and 85% accuracy improvements over the best baseline in single- and multi-granularity segmentation, respectively. In iterative inference, MemPromptTSS provides stronger refinement capability, with average per-iteration gains of 2.66 percentage points compared to 1.19 for PromptTSS, and 6.07 versus 2.25 when focusing on the first four iterations. These results demonstrate that persistent memory, together with prompt-guided refinement, is essential for reliable segmentation under sparse supervision.

For future work, we will explore (i) a confidence-gated write strategy, where only high-confidence prompts are stored in the memory bank to improve robustness, and (ii) methods to scale subsequence length without losing accuracy or incurring high training cost, addressing the performance dip observed when longer subsequences are used.

Acknowledgments

The authors would like to thank GoEdge.ai for providing internship support, computing resources, and valuable domain expertise that contributed to this research.

References

[1] Ahmed T Alhasani. 2025. 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* 3, 2 (2025), 367–381.

[2] Unai Bermejo, Aitor Almeida, Aritz Bilbao-Jayo, and Gorka Azkune. 2021. Embedding-based real-time change point detection with application to activity segmentation in smart home time series data. *Expert Systems with Applications* 185 (2021), 115641.

[3] Ching Chang, Chiao-Tung Chan, Wei-Yao Wang, Wen-Chih Peng, and Tien-Fu Chen. 2024. TimeDRL: Disentangled Representation Learning for Multivariate Time-Series. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 625–638. doi:10.1109/ICDE60146.2024.00054

[4] Ching Chang, Chan Chiao-Tung, Wei-Yao Wang, Wen-Chih Peng, and Tien-Fu Chen. 2024. Self-Supervised Learning of Disentangled Representations for Multivariate Time-Series. In *NeurIPS 2024 Workshop: Self-Supervised Learning-Theory and Practice*.

[5] Ching Chang, Jeehyun Hwang, Yidan Shi, Haixin Wang, Wen-Chih Peng, Tien-Fu Chen, and Wei Wang. 2025. Time-IMM: A Dataset and Benchmark for Irregular Multimodal Multivariate Time Series. *arXiv preprint arXiv:2506.10412* (2025).

[6] Ching Chang, Ming-Chih Lo, Wen-Chih Peng, and Tien-Fu Chen. 2025. PromptTSS: A Prompting-Based Approach for Interactive Multi-Granularity Time Series Segmentation. *arXiv preprint arXiv:2506.11170* (2025).

[7] Ching Chang, Yidan Shi, Defu Cao, Wei Yang, Jeehyun Hwang, Haixin Wang, Jiacheng Pang, Wei Wang, Yan Liu, Wen-Chih Peng, et al. 2025. A Survey of Reasoning and Agentic Systems in Time Series with Large Language Models. *arXiv preprint arXiv:2509.11575* (2025).

[8] Ching Chang, Wei-Yao Wang, Wen-Chih Peng, and Tien-Fu Chen. 2025. LLM4TS: Aligning Pre-Trained LLMs as Data-Efficient Time-Series Forecasters. *ACM Trans. Intell. Syst. Technol.* 16, 3, Article 60 (April 2025), 20 pages. doi:10.1145/3719207

[9] Ching Chang, Wei-Yao Wang, Wen-Chih Peng, Tien-Fu Chen, and Sagar Samtani. 2024. Align and Fine-Tune: Enhancing LLMs for Time-Series Forecasting. In *NeurIPS Workshop on Time Series in the Age of Large Models*. https://openreview.net/forum?id=AaRCmJieG4

[10] Mouxiang Chen, Lefei Shen, Han Fu, Zhuo Li, Jianling Sun, and Chenghao Liu. 2024. 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*. 341–352.

[11] Luiz Carlos de Jesus Jr, Francisco Fernández-Navarro, and Mariano Carbonero-Ruz. 2025. Enhancing financial time series forecasting through topological data analysis. *Neural Computing and Applications* 37, 9 (2025), 6527–6545.

[12] Zihan Dong, Xinyu Fan, and Zhiyuan Peng. 2024. Fnspid: A comprehensive financial news dataset in time series. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4918–4927.

[13] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee-Keong Kwoh, and Xiaoli Li. 2024. Label-efficient time series representation learning: A review. *IEEE Transactions on Artificial Intelligence* (2024).

[14] Yingchun Fu, Zhe Zhu, Liangyun Liu, Wenfeng Zhan, Tao He, Huanfeng Shen, Jun Zhao, Yongxue Liu, Hongsheng Zhang, Zihan Liu, et al. 2024. Remote sensing time series analysis: A review of data and applications. *Journal of Remote Sensing* 4 (2024), 0285.

[15] Stefan Gaugel and Manfred Reichert. 2023. PrecTime: A deep learning architecture for precise time series segmentation in industrial manufacturing operations. *Engineering Applications of Artificial Intelligence* 122 (2023), 106078.

[16] Dimitra Gkatzia, Helen Hastie, and Oliver Lemon. 2014. 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)*. 1231–1240.

[17] David Hallac, Sagar Vare, Stephen Boyd, and Jure Leskovec. 2017. 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*. 215–223.

[18] Huan He, Owen Queen, Teddy Koker, Consuelo Cuevas, Theodoros Tsiligkaridis, and Marinka Zitnik. 2023. Domain adaptation for time series under feature and label shifts. In *International conference on machine learning*. PMLR, 12746–12774.

[19] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. 2023. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4015–4026.

[20] Rumena Komitova, Dominik Raabe, Robert Rein, and Daniel Memmert. 2022. Time series data mining for sport data: A review. *Journal homepage: http://iacss. org/index. php? id* 21, 2 (2022).

[21] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. 2011. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter* 12, 2 (2011), 74–82.

[22] Shijie Li, Yazan Abu Farha, Yun Liu, Ming-Ming Cheng, and Juergen Gall. 2020. Ms-tcn++: Multi-stage temporal convolutional network for action segmentation. *IEEE transactions on pattern analysis and machine intelligence* 45, 6 (2020), 6647–6658.

[23] Cheng-Ming Lin, Ching Chang, Wei-Yao Wang, Kuang-Da Wang, and Wen-Chih Peng. 2024. Root Cause Analysis in Microservice Using Neural Granger Causal Discovery. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 1 (Mar. 2024), 206–213. doi:10.1609/aaai.v38i1.27772

[24] Weiwei Liu, Haobo Wang, Xiaobo Shen, and Ivor W Tsang. 2021. The emerging trends of multi-label learning. *IEEE transactions on pattern analysis and machine intelligence* 44, 11 (2021), 7955–7974.

[25] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. 2024. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=JePfAI8fah

[26] Ming-Chih Lo, Ching Chang, and Wen-Chih Peng. 2024. Text2Freq: Learning Series Patterns from Text via Frequency Domain. In *NeurIPS Workshop on Time Series in the Age of Large Models*. https://openreview.net/forum?id=Pi6sA1MSSr

[27] Lakmal Meegahapola, Hamza Hassoune, and Daniel Gatica-Perez. 2024. M3BAT: Unsupervised Domain Adaptation for Multimodal Mobile Sensing with Multi-Branch Adversarial Training. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 8, 2 (2024), 1–30.

[28] Christoforos Nalmpantis and Dimitris Vrakas. 2020. On time series representations for multi-label NILM. *Neural Computing and Applications* 32 (2020), 17275–17290.

[29] Ashwin Narayan, Francisco Anaya Reyes, Meifeng Ren, and Yu Haoyong. 2021. Real-time hierarchical classification of time series data for locomotion mode detection. *IEEE Journal of Biomedical and Health Informatics* 26, 4 (2021), 1749–1760.

[30] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *ICLR*. OpenReview.net.

[31] Francisco Javier Ordóñez and Daniel Roggen. 2016. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors* 16, 1 (2016), 115.

[32] Ting-Yun Ou, Ching Chang, and Wen-Chih Peng. 2024. 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* (Boise, ID, USA) *(CIKM '24)*. Association for Computing Machinery, New York, NY, USA, 4803–4810. doi:10.1145/3627673.3680083

[33] Fengchao Peng, Qiong Luo, and Lionel M Ni. 2017. ACTS: an active learning method for time series classification. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 175–178.

[34] Mathias Perslev, Michael Jensen, Sune Darkner, Poul Jørgen Jennum, and Christian Igel. 2019. U-time: A fully convolutional network for time series segmentation applied to sleep staging. *Advances in Neural Information Processing Systems* 32 (2019).

[35] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. 2024. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714* (2024).

[36] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th International Symposium on Wearable Computers*. IEEE, 108–109.

[37] Burr Settles. 2009. Active learning literature survey. (2009).

[38] Yanru Sun, Zongxia Xie, Dongyue Chen, Emadeldeen Eldele, and Qinghua Hu. 2025. Hierarchical classification auxiliary network for time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 20743–20751.

[39] Chengyu Wang, Kui Wu, Tongqing Zhou, and Zhiping Cai. 2023. Time2state: An unsupervised framework for inferring the latent states in time series data. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–18.

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[41] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

[42] Mi Zhang and Alexander A Sawchuk. 2012. USC-HAD: A daily activity dataset for ubiquitous activity recognition using wearable sensors. In *Proceedings of the 2012 ACM conference on ubiquitous computing*. 1036–1043.