RTSM: ADAPTING RECURSIVE MODELS TO MULTIVARIATE TIME-SERIES CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Accurate multivariate time-series classification requires extracting temporal features and integrating information across a sequence. Weight-tied refinement offers a way to repeatedly process these features without allocating new parameters at each stage. Applying this principle to time series requires choosing both a temporal representation and a latent-state organization suited to classification. We introduce the Recursive Time-Series Model (RTSM), which combines a learned convolutional temporal tokenizer, single-state weight-tied refinement, and a pooled classification readout. Across all 30 UEA datasets, RTSM achieves 72.81% mean accuracy under validation-based configuration and checkpoint selection, the highest observed mean among the 17 evaluated deep-learning models and 1.60 percentage points above MambaSL. Ablations on eight datasets favor the proposed tokenizer over a simple convolutional embedding and the single-state model over the tested two-state variants. The complete classifier reaches 86.10% mean accuracy versus 84.79% for a separately trained tokenizer-only classifier. These findings identify temporal encoding and state organization as key design choices for adapting recursive architectures to time series. The resulting classifier combines competitive accuracy with a shared refinement module whose parameter count is independent of computational depth.

1 INTRODUCTION

Multivariate time-series classification (TSC) assigns one label to a sequence of observations from multiple variables. Accurate prediction requires extracting temporal patterns and combining evidence across the sequence. The UEA archive illustrates the diversity of this task, with datasets that differ substantially in sequence length, dimensionality, and training-set size (Bagnall et al., 2018). Convolutional, attention-based, and state-space models offer different ways to build and process these representations (Jung & Kim, 2026). Our goal is to achieve strong classification accuracy across this diversity using a refinement module whose parameters are shared across computational depth.

Weight sharing allows repeated computation without allocating a separate set of weights to each processing stage. Recent Recursive Reasoning Models (RRMs), including the Hierarchical Reasoning Model (HRM) (Wang et al., 2025a) and Tiny Recursive Model (TRM) (Jolicoeur-Martineau, 2025), use this principle to refine latent states for structured prediction. Adapting these architectures to multivariate TSC raises two design questions: how to represent continuous temporal inputs for the shared module, and whether a two-state organization is useful when the output is one sequence-level class label. We therefore examine temporal encoding and latent-state organization together as the basis for an accurate classifier using shared refinement.

We introduce the *Recursive Time-Series Model (RTSM)*, which couples a learned temporal encoder to a single-state, weight-tied refinement module. The encoder extracts local temporal features; repeated applications of shared attention combine information across the resulting tokens; and a pooled readout produces the class prediction. Figure 1 illustrates the architecture. Our contributions are:

1. Temporal encoding with shared refinement. RTSM combines a convolutional temporal tokenizer with one evolving latent state and a pooled classification readout. The same attention module is reused throughout refinement. Training builds on TRM's deep supervision and adaptive halting, while inference uses a fixed refinement budget (Section 3).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

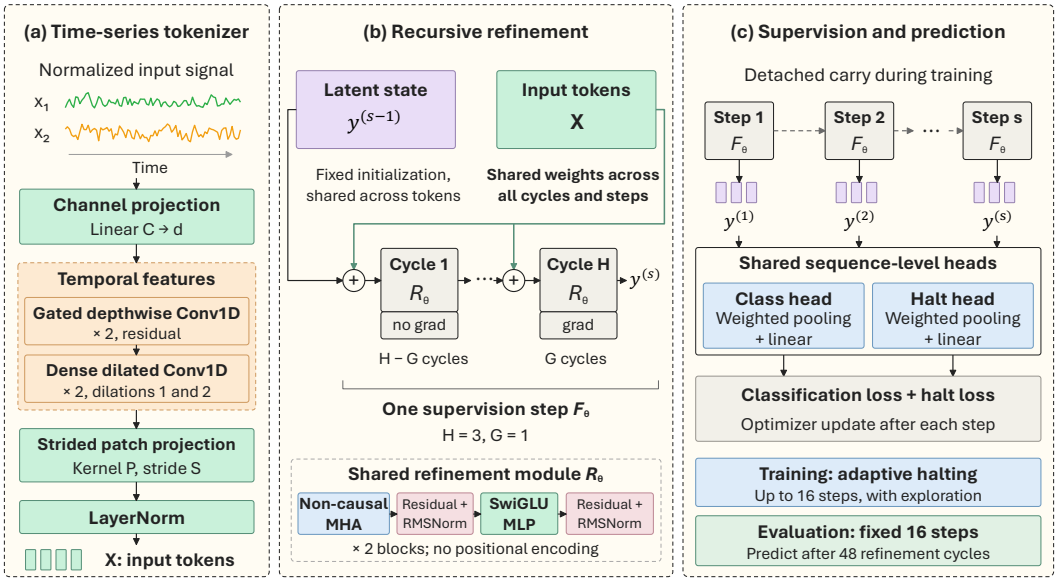


Figure 1: RTSM encodes a time series into input tokens X (green) and repeatedly updates one latent state y (purple). (a) The tokenizer extracts temporal features before forming patches. Temporal order enters through the tokenizer; refinement uses no positional encoding. (b) Each supervision step maps $y^{(s-1)}$ to $y^{(s)}$ through $H = 3$ applications of the shared module R_θ, adding X before every cycle. The initial state broadcasts a fixed random vector $y_{\text{init}} \in \mathbb{R}^d$ across tokens. Training retains gradients through the final $G = 1$ cycle. (c) Each step receives a classification loss, a halt loss, and an optimizer update; the carried state is detached before the next step. Training uses adaptive halting with exploration, up to 16 steps. Evaluation uses a fixed 16 steps (48 refinement cycles).

2. Classification accuracy across the UEA archive. Under validation-based configuration and checkpoint selection, RTSM achieves 72.81% mean accuracy across all 30 UEA datasets, the highest observed mean among the 17 evaluated deep-learning models and 1.60 percentage points above MambaSL (Section 4.1).

3. Design insights for recursive time-series classification. Across eight datasets, the proposed tokenizer achieves higher mean accuracy than a simple convolutional embedding in each tested comparison. The single-state model also outperforms the tested two-state variants, and the complete classifier improves on a separately trained tokenizer-only model by 1.31 percentage points. Further ablations examine depth and supervision, followed by an inference-budget analysis on 28 datasets (Sections 4.2 and 4.3).

2 Background and Related Work

2.1 Multivariate Time-Series Classification

Time-series classifiers differ in how they extract temporal features and combine them into a sequence-level prediction. ROCKET and MultiRocket use fixed convolutional filters, pooled feature summaries and linear classifiers (Dempster et al., 2020; Tan et al., 2022). Neural approaches learn representations jointly with the prediction objective: ModernTCN uses a convolutional backbone (Luo & Wang, 2024), while MambaSL adapts a selective state-space model for classification (Gu & Dao, 2024; Jung & Kim, 2026). Attention-based forecasting architectures, including Crossformer, have also been adapted and evaluated for classification within the MambaSL benchmark (Zhang & Yan, 2023; Jung & Kim, 2026).

Beyond the backbone, the input representation and sequence-level readout are important architectural choices. MambaSL explicitly studies input projection, feature extraction, and aggregation as parts of the classification pipeline (Jung & Kim, 2026). For a recursive classifier, these choices determine which temporal features condition the latent updates and how the resulting states produce a class

prediction. We therefore study the adaptation as a complete classification architecture, assessing temporal tokenization and state structure alongside refinement depth.

2.2 Recursive Architectures

Recursive architectures build computational depth by repeatedly applying shared update modules to an evolving latent representation. Recent *RRMs*, including HRM (Wang et al., 2025a) and TRM (Jolicoeur-Martineau, 2025), build on earlier work on adaptive computation and weight-tied depth (Graves, 2016; Dehghani et al., 2019). We describe their design through five components: *latent states*, *recursive update structure*, *gradient schedule*, *deep supervision*, and *halting mechanism*.

Latent states and recursive updates. A latent state y holds the representation used for prediction, while optional auxiliary states $\mathbf{z}$ support its updates. An encoder f_I embeds the input $\tilde{x}$, a recursive procedure F_θ updates the states, and a readout f_O produces the prediction:

$$x = f_I(\tilde{x}), \qquad (y, \mathbf{z}) \leftarrow F_\theta(y, \mathbf{z}, x), \qquad \hat{y} = f_O(y). \tag{1}$$

HRM and TRM each use one auxiliary state z. In our notation, a call to F_θ comprises H_{cycles} cycles, each containing L_{cycles} updates $z \leftarrow f_L(z, y, x)$ followed by one update $y \leftarrow f_H(y, z)$. HRM uses separate networks for the two updates; TRM shares one network. A single-state formulation omits $\mathbf{z}$, sets $L_{\text{cycles}} = 0$, and updates y directly conditioned on x.

Gradient schedule. Forward computation can extend beyond the portion retained for backpropagation. We denote by $G_{\text{cycles}} \leq H_{\text{cycles}}$ the number of final cycles tracked for gradients, with earlier cycles evaluated without gradient tracking. TRM differentiates through the complete final cycle $G_{\text{cycles}} = 1$; HRM retains only the final f_L and f_H updates. This is analogous to truncating backpropagation through time (Aicher et al., 2020), but across refinement cycles. Throughout the paper, $H/L/G$ denotes cycles, inner auxiliary-state updates, and gradient-tracked cycles, respectively.

Deep supervision and halting. Intermediate supervision provides a learning signal at successive stages of computation (Lee et al., 2015). In HRM and TRM, each *supervision step* applies the recursive procedure to the states carried from the preceding step and supervises the resulting prediction

$$\left(y^{(s)}, \mathbf{z}^{(s)}\right) = F_\theta\left(y^{(s-1)}, \mathbf{z}^{(s-1)}, x\right), \qquad \hat{y}^{(s)} = f_O\left(y^{(s)}\right). \tag{2}$$

The carried states are detached from the computation graph between steps. Adaptive halting allows a sample to finish before a maximum budget $S_{\max}$: HRM learns halt and continue values through Q-learning, whereas TRM trains a scalar halt score q to predict whether the current answer is correct. Halting controls the number of supervision steps, separately from the within-step depths H_{cycles} and L_{cycles}. The stopping rule may differ between training and inference; a fixed inference budget applies the same number of steps to every sample.

2.2.1 Inference-Time Computation

Inference computation can vary along two axes: *depth* D, the number of successive applications of F_θ along a trajectory, and *breadth* B, the number of stochastic trajectories explored. Depth is measured in units corresponding to supervision steps during training. In Equilibrium Reasoners (EqR) (Huang et al., 2026), models are trained with randomized state initializations and noise injection to support depth and breadth scaling. Probabilistic Tiny Recursive Model (PTRM) instead introduces noise only at inference and selects among candidate answers using the learned Q-head (Sghaier et al., 2026). These approaches motivate our characterization of prediction accuracy and latent-state changes under additional deterministic refinement and stochastic exploration (Section 4.3).

We next describe how RTSM adapts TRM's refinement and training framework to sequence classification.

3 Recursive Time-Series Model

RTSM adapts recursive computation to multivariate classification through temporal encoding, a single latent state, and a sequence-level readout. A convolutional tokenizer first converts the observations

into temporal feature vectors, or tokens. A shared attention-based module then updates a latent state containing one vector per token. Each update combines the current state with the tokenizer output, keeping the input features available throughout refinement. A learned pooling operation combines the updated token states into a sequence-level class prediction. Figure 1 illustrates these three components.

Building on TRM (Jolicoeur-Martineau, 2025), we supervise predictions at intermediate stages of refinement (deep supervision) and use learned halting during training. At inference, every sample receives a fixed number of refinement steps. Table 1 summarizes the adaptation from TRM. Section 3.4 describes the training procedure and evaluation budget, while Appendix C provides implementation details and a component-level comparison with HRM and TRM.

Table 1: From TRM to RTSM: the main adaptations and retained training mechanisms. The pooled readout follows MambaSL (Jung & Kim, 2026); Appendix C.3 gives the full component comparison.

Component	TRM	RTSM
Input	Discrete token embeddings	Convolutional temporal tokens from real-valued sequences
State	Prediction state y and auxiliary state z	One token-level latent state y refined using the input tokens
Readout	Per-token class logits	Pooled sequence-level class and correctness logits
Training	Deep supervision; gradients through the final cycle; correctness-based halting	Same recipe, with correctness targets for sequence classification

Task and notation. An input $\tilde{X} \in \mathbb{R}^{C \times T}$ contains C variables observed over T time points, with class label $c \in \{1, \ldots, K\}$. The task is to predict one label for the entire sequence. We use N for the number of tokens and d for their feature width.

3.1 Time-series Tokenizer

The tokenizer is a learned temporal encoder that extracts features before attention combines information across tokens. A learned channel projection first combines the observed variables at each time point. Gated residual convolutions then filter features across neighboring time points, using input-dependent gates to modulate the filtered features before adding them back to the representation. Dilated convolutions (van den Oord et al., 2016) extend the temporal context by filtering at spaced positions. Dense connections (Huang et al., 2017) give each dilated layer access to the feature maps produced by preceding layers. These operations preserve the sequence length; Appendix C.1 specifies their computations.

Let $V \in \mathbb{R}^{d \times T}$ denote the resulting feature map. To form tokens, an unpadded convolution maps each patch of P consecutive positions in V to a d-dimensional vector, advancing by S positions between patches. Layer normalization gives

$$X = f_I(\tilde{X}) = \mathrm{LN}\big(\mathrm{Conv}_{P,S}(V)^\top\big) \in \mathbb{R}^{N \times d}, \qquad N = 1 + \left\lfloor \frac{T - P}{S} \right\rfloor. \tag{3}$$

Each token therefore combines features that already incorporate temporal context from the preceding convolutions. Patch length P and stride S determine the token resolution at which the shared refinement module operates and are selected per dataset under the evaluation protocols of Section 4.

3.2 Single-state Refinement

RTSM maintains a latent state $y \in \mathbb{R}^{N \times d}$, whose rows represent the input tokens. Refinement repeatedly processes the encoded sequence: its cycles do not advance through the observed time points. The readout below converts the updated state into a class prediction.

One *refinement cycle* adds the input tokens X to the current state and applies the shared attention-based module R_θ. A *step* groups H_{cycles} such cycles, after which the model produces a prediction that is supervised during training. The reference configuration uses three cycles per step. For cycle h within step s,

$$y^{(s,h+1)} = R_\theta\Big(y^{(s,h)} + X\Big), \qquad h = 0, \ldots, H_{\text{cycles}} - 1. \tag{4}$$

Each step starts from the preceding step's terminal state, $y^{(s,0)} = y^{(s-1)}$, and returns $y^{(s)} = y^{(s,H_{\text{cycles}})}$. The input representation remains available at every cycle, while attention mixes information across the token states. All cycles use the same update module and act directly on y; the correspondence to TRM's two-state formulation is given in Appendix C.3.

For a new sample, a fixed, non-trainable random vector $y_{\text{init}} \in \mathbb{R}^d$ is broadcast across the N tokens. The reference module has two blocks, each comprising non-causal multi-head self-attention and a swish-gated linear unit (SwiGLU) multi-layer perceptron (MLP), with residual connections and RMSNorm after each sublayer. It uses $d = 512$, eight attention heads, and an MLP intermediate width of 1536, with no positional encoding in the refinement module.

3.3 Sequence-level Readout

After each refinement step, learned weighted pooling combines the token states into sequence-level outputs. The classification head produces class logits ℓ, while a second head produces a halt logit q, trained to estimate whether the current prediction is correct. This score controls training duration; benchmark inference uses a fixed budget.

Following the pooling construction of MambaSL (Jung & Kim, 2026), each head scores a token by taking the largest component of a learned projection, then normalizes the scores across tokens. Separate pooling parameters allow classification and halting to emphasize different tokens. For head $r \in \{\text{cls}, \text{halt}\}$ and token states $y_1, \ldots, y_N$, the weights and outputs are

$$\begin{aligned} a_n^r &= \mathrm{softmax}_n\Big(\max_j [W_r^{\text{pool}} y_n + b_r^{\text{pool}}]_j\Big), \\ \ell &= \sum_{n=1}^{N} a_n^{\text{cls}} W_{\text{cls}} y_n, \qquad q = w_q^\top \sum_{n=1}^{N} a_n^{\text{halt}} y_n + b_q. \end{aligned} \tag{5}$$

Here j indexes the learned pooling channels, and $\mathrm{softmax}_n$ normalizes over the N tokens. The predicted class is $\hat{c} = \arg\max_k \ell_k$.

With broadcast state initialization, position-independent refinement, and pooled prediction, the classifier is invariant to permutations of the tokenizer outputs. Temporal ordering enters through the tokenizer's convolutional receptive fields.

3.4 Training and Inference

A training trajectory starts from the broadcast initial state and continues through supervision steps until halting or the maximum budget. In the reference configuration, each step performs two refinement cycles without gradient tracking, followed by one cycle retained for backpropagation ($G_{\text{cycles}} = 1$). The resulting state supplies both heads, and the loss at step s supervises the class prediction and its estimated correctness:

$$\mathcal{L}^{(s)} = \mathcal{L}_{\text{SM}}\Big(\ell^{(s)}, c\Big) + \lambda_q \, \mathcal{L}_{\text{BCE}}\Big(q^{(s)}, \mathbb{1}\Big[\arg\max_k \ell_k^{(s)} = c\Big]\Big). \tag{6}$$

Here $\mathcal{L}_{\text{SM}}$ is the Stablemax cross-entropy (Prieto et al., 2025), adopted from TRM (Jolicoeur-Martineau, 2025), and $\lambda_q = 0.5$ weights the binary correctness objective. An optimizer update follows each supervision step. If the trajectory continues, the terminal latent state is detached from the computation graph and carried forward to initialize the next step. Gradients therefore pass through the final cycle of the current step, but not through earlier cycles or preceding steps.

During training, a positive halt logit can end a sample's trajectory before the maximum $S_{\max} = 16$ supervision steps. An exploration rule sometimes delays early halting, as specified in Appendix C.2.

At inference, parameters remain fixed and learned halting is disabled. Every sample receives $S_{\text{eval}} = 16$ steps of three cycles each, for 48 applications of R_θ, and the final step supplies the class prediction. In the inference analysis, depth D counts these three-cycle steps, so the benchmark uses $D = 16$ (Section 2.2.1). Evaluation uses an exponential moving average of the trainable parameters with decay 0.999. Appendix C.2 gives the remaining training settings.

Batch scheduling. Persistent batch slots accommodate sample-dependent halting. Each slot retains its sample and latent state until the trajectory ends, then accepts another sample. Training progress is counted in completed trajectories per sample, separately from the number of supervised updates. Appendix A.1 specifies how these counts determine the finite training budget.

Computational budget. Weight sharing separates model size from refinement depth: increasing the evaluation depth reuses the same parameters. At depth D, the reference configuration performs $3D$ applications of its two-block refinement module, corresponding to 96 attention–MLP block applications at the benchmark depth $D = 16$. The tokenizer-only baseline omits these refinement operations. Computation also depends on the number of tokens N, set by the patch length and stride. During training, the truncated gradient schedule retains only the final cycle for backpropagation within each step; the preceding cycles still incur forward computation.

4 Experiments

Our evaluation addresses three questions: how the complete model compares with established classifiers, which configurations support its adaptation to time series, and how its predictions change with additional computation.

4.1 Classification on the UEA Archive

Evaluation setup. We evaluate all 30 multivariate TSC datasets in the UEA archive (Bagnall et al., 2018), training a separate model for each dataset. Our primary evaluation uses an 80/20 stratified split of the official training partition. Model configurations and checkpoints are selected using validation accuracy, and the official test set is reserved for final evaluation. Normalization fitting scopes differ across historical runs and model pipelines, as detailed in Appendix A.

We compare RTSM with 16 deep-learning baselines from the MambaSL benchmark (Jung & Kim, 2026), re-evaluated under this validation protocol using their respective model search spaces. The comparison covers convolutional, attention-based, MLP-based, shape-based, and state-space models. Benchmark runs use seed 2021; the primary metric is classification accuracy averaged equally across the 30 datasets. The comparison evaluates complete model-and-training configurations, with their respective optimizers, training settings, and search spaces documented in Appendix A.

Results with validation-based selection. RTSM achieves the highest observed average accuracy among the 17 evaluated models, reaching 72.81% compared with 71.21% for MambaSL and 71.13% for TSCMamba (Figure 2). Its advantage over MambaSL is 1.60 percentage points, and it ranks among the top five on 21 of the 30 datasets. The aggregate performance is accompanied by strong results on datasets with different input structures. On the Large Synoptic Survey Telescope (LSST) dataset, RTSM reaches 65.37%, exceeding the next-best evaluated model by 7.83 points. It also achieves the highest accuracy among the evaluated models on EigenWorms, whose sequences contain $17{,}984$ time steps, and PEMS-SF, which has 963 input channels. These results demonstrate the applicability of the complete architecture across short, long and high-dimensional time series.

Comparison with the published MambaSL benchmark. For continuity with prior results, we also evaluate RTSM under the MambaSL protocol (Jung & Kim, 2026), which trains on the full official training partition and uses the official test set for configuration and checkpoint selection, including early stopping. We reuse the published results for MambaSL and its 20 baselines. RTSM reaches 80.82% average accuracy, the highest observed mean in this comparison, exceeding MambaSL's 79.82% by 1.00 percentage point. Figure 4 and Table 6 in Appendix B provide the legacy comparison. The protocols are reported separately because their training partitions and selection procedures differ; the validation protocol provides the primary evaluation with test-independent model selection.

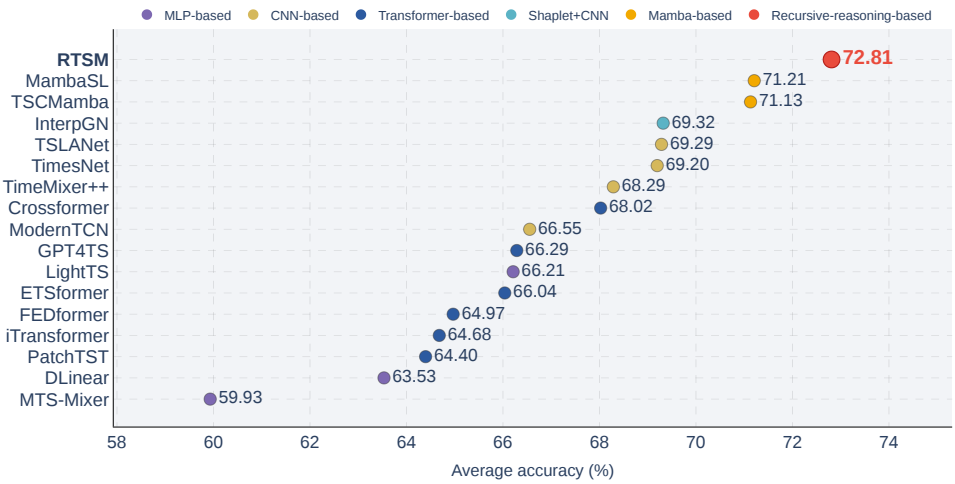


Figure 2: Mean classification accuracy (%) across all 30 UEA datasets for the 17 evaluated deep-learning models under validation-based configuration and checkpoint selection. Each dataset receives equal weight. Full per-dataset results and pairwise Wilcoxon tests appear in Appendix B.

4.2 Architectural and Training Ablations

The ablations examine the design choices involved in adapting the recursive framework to time-series classification: temporal encoding, state structure, and the refinement schedule. Ablations use eight UEA datasets (CT, DDG, HB, JV, LSST, NATO, PD, and SAD), retaining the per-dataset hyperparameters selected under the validation protocol and averaging over seeds 0, 1, and 2. Tables 2 and 3 summarize the main comparisons; per-seed results and further variants appear in Appendix B.3.

Temporal encoding and a simpler latent state. The proposed tokenizer improves mean accuracy wherever both tokenizers are evaluated (Table 2). With the RTSM tokenizer, the single-state model reaches 86.10%, compared with 84.98% and 84.29% for the two-state variants. These variants retain the TRM training recipe; separate update networks capture one architectural feature of HRM, rather than reproducing HRM's full procedure. These comparisons favor the proposed temporal encoder paired with a single recurrent state. They compare complete configurations with different parameter counts and computational costs.

Table 2: Tokenizer and state structure: mean accuracy (%) over eight datasets and three seeds. "None" applies the prediction head directly to tokenizer outputs. The two-state variants use shared or separate update networks within the TRM training recipe (Appendix B.3); a dash marks an unevaluated combination.

	State structure			
Tokenizer	None	Two-state shared f	Two-state separate f_H/f_L	Single state
Conv1D	73.72	78.96	79.69	–
RTSM	84.79	84.98	84.29	**86.10**

Complete model versus tokenizer-only classifier. The tokenizer-only classifier trains the temporal encoder and prediction head directly. Using the same tokenizer architecture, the complete single-state model reaches $86.10 \pm 0.31\%$ versus $84.79 \pm 0.41\%$ for this baseline, a gain of 1.31 percentage points. Each seed's score averages the eight datasets equally; $\pm$ denotes the sample standard deviation across the three seed-level scores. Figure 3 shows higher mean accuracy on six datasets, the same mean to two decimal places on Heartbeat, and lower mean accuracy on NATOPS. The overall difference is positive for each of the three seeds. These results favor the complete classifier over direct classification of tokenizer outputs on this subset. Because the comparison changes both attention computation and training, we examine depth and state reset separately below.

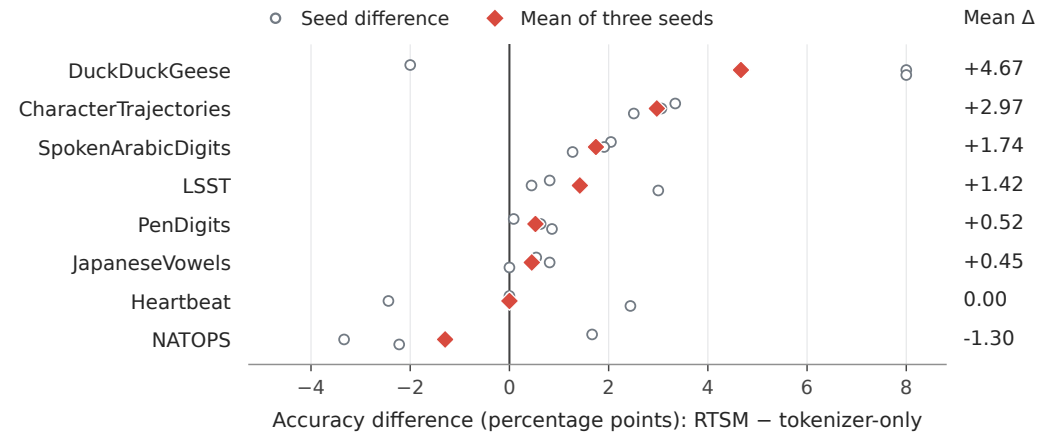


Figure 3: RTSM versus the tokenizer-only configuration on all eight ablation datasets. Positive differences favor RTSM. Open circles show the accuracy difference for each seed (0, 1, or 2); diamonds and numerical labels show their mean. Small vertical offsets separate the seed markers. Datasets are ordered by their mean difference. Full per-seed accuracies appear in Appendix B.3.

Sequence readout. The readout ablation yields similar observed mean accuracies: 86.14% with mean pooling and 86.10% with the reference learned weighted pooling (Appendix B.3).

Recursive depth and supervision. The single-state model maintains similar accuracy across the tested within-step depths: $H = 1$–5 yields 85.60–86.10%, with $H = 3$ giving the highest mean. At $H = 3$, adaptive training with up to 16 supervision steps reaches 86.10%, compared with 85.98% for one supervision step and 85.05% when all 16 steps are forced. At $H = 1$, updates still recur across supervision steps; at $H = 3$, one supervision step contains three refinement cycles. The sweep identifies several effective refinement budgets, with no monotonic accuracy gain from increasing depth. The state-reset variants reinitialize the latent state between supervision steps while retaining refinement within each step. These variants reach 86.00% at $H = 1$ and 86.09% at $H = 3$, both close to the reference mean of 86.10%. Appendix B.3 reports all variants.

Table 3: Recursive depth and training supervision: mean accuracy (%) over the same eight datasets and three seeds. Left: vary the number of recursive cycles H within each supervision step, using adaptive supervision capped at 16 steps. Right: vary the supervision schedule at $H = 3$.

Within-step depth H					Training supervision		
1	2	3	4	5	One step	Adaptive, up to 16	Fixed 16
85.77	85.60	**86.10**	85.61	85.78	85.98	**86.10**	85.05

4.3 Prediction Behavior across Inference Budgets

The preceding ablations change the architecture or training schedule. We now vary the inference budget for fixed trained checkpoints to examine how predictions change with additional computation. This probe covers 28 UEA datasets, excluding EigenWorms and InsectWingbeat because their runs did not complete within the allocated compute budget. We evaluate deterministic trajectories up to depth $D = 64$ (three recursive cycles per step) and stochastic trajectories with randomized initialization and between-step noise, using breadths up to $B = 512$. Perturbations are introduced only at inference. The comparisons hold checkpoints and preprocessing fixed within the probe; normalization differs from some historical benchmark runs. Appendix B.4 provides the full protocol, depth–breadth grids, and dataset-level measurements.

Deterministic trajectories. Average accuracy changes from 72.42% at step 1 to 72.61% at step 16 and 72.84% at step 64. Over the same trajectory, the mean root mean square (RMS) state-update magnitude decreases from 1.4172 to 0.0067. The latent updates therefore become much smaller while aggregate accuracy changes only modestly; extending the trajectory from 16 to 64 steps adds 0.23 percentage points. Thus, smaller state updates and higher classification accuracy are distinct outcomes of refinement; these measurements characterize state stabilization without establishing convergence to a fixed point.

Stochastic trajectories and selection. At depth $D = 1$, increasing breadth from 1 to 512 raises oracle pass@B accuracy from 72.03% to 77.71%. This oracle metric counts a sample as correct if any trajectory predicts its label and therefore measures coverage, not the accuracy of an implementable selector. At $D = 64$, the same breadth increase changes oracle accuracy from 72.78% to 73.36%. At the maximum budget ($D = 64, B = 512$), selecting by the halting score, majority vote, or minimum residual (the smallest recent state changes) yields 72.66%, 72.76%, and 72.74%, respectively, compared with 72.84% for the deterministic trajectory. Within this probe, additional depth yields only modest accuracy changes, and the tested stochastic selectors do not improve on deterministic inference at the maximum budget.

Weight reuse and deployment. At its reference inference budget, RTSM applies the same 6.82-million-parameter refinement module 48 times. Allocating independent weights to each application would require 327.16 million refinement parameters—a 48-fold difference in refinement-weight storage. At 16 bits per weight, these footprints correspond to 13 MiB and 624 MiB, respectively, excluding the tokenizer, readout, and activations. Reusing the smaller weight set creates an opportunity to reduce off-chip memory traffic when weights remain resident on-chip. This mechanism has demonstrated substantial gains in other recurrent architectures: Diamos et al. (2016) report approximately $30\times$ higher recurrent-kernel throughput than a matrix-multiplication implementation on a Titan X Graphics Processing Unit (GPU), using $1{,}152$ hidden units and a batch size of four. These results motivate hardware-aware implementations of RTSM for edge devices and GPU inference, with benefits depending on memory capacity, activation traffic, and execution scheduling.

5 Conclusion

We introduced RTSM, an adaptation of recursive architectures to multivariate TSC through a learned temporal tokenizer, single-state weight-tied refinement, and a pooled sequence-level readout. Across the 30 UEA datasets, RTSM achieves 72.81% average accuracy under validation-based selection, the highest observed mean among the evaluated deep-learning models. Eight-dataset ablations favor the proposed temporal tokenizer over a simple convolutional embedding and the single-state formulation over the tested two-state variants. The complete architecture also improves on a separately trained tokenizer-only classifier by 1.31 percentage points on this subset.

Depth and state-reset ablations identify several refinement schedules with similar mean accuracy. RTSM combines strong classification accuracy with a learned temporal encoder, one latent state, and a refinement module whose parameters are reused across computational depth.

Reproducibility statement

The experimental environment, dataset preprocessing, evaluation protocols, hyperparameter configurations and search spaces, random seeds, and checkpoint-selection procedures are documented in Appendix A. All datasets used in this work are publicly available and referenced therein. For evaluations of non-RTSM models, we adapt the implementations and benchmarking infrastructure released with MambaSL (Jung & Kim, 2026), modifying the evaluation procedure where necessary to follow the validation-based protocol described in this work.

AI use statement

In this work, we used generative AI tools, including Anthropic Claude and OpenAI Codex, to assist with performance engineering of the experimental code, including refactoring and improving execution efficiency, and to improve the grammar, clarity, and concision of author-written manuscript text. We also used these tools to search for and summarize related work, which helped us identify papers to read. The tools were not used to formulate research questions or hypotheses, design experiments, generate data or experimental results, or draw scientific conclusions. The authors reviewed all retained AI-assisted code changes and validated the final implementation by executing the relevant experiments and inspecting their recorded outputs. All AI-assisted language edits were reviewed and approved by the authors. The authors take responsibility for the final text, code, claims, and artifacts.

References

Md Atik Ahamed and Qiang Cheng. TSCMamba: Mamba meets multi-view learning for time series classification. *Information Fusion*, 120:103079, 2025. doi: 10.1016/j.inffus.2025.103079. URL https://www.sciencedirect.com/science/article/pii/S1566253525001526.

Christopher Aicher, Nicholas J. Foti, and Emily B. Fox. Adaptively truncating backpropagation through time to control gradient bias. In *Proceedings of the 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pp. 799–808. PMLR, 2020. URL https://proceedings.mlr.press/v115/aicher20a.html.

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

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Angus Dempster, François Petitjean, and Geoffrey I Webb. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020. doi: 10.1007/s10618-020-00701-z. URL https://link.springer.com/article/10.1007/s10618-020-00701-z.

Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. Hydra: Competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery*, 37:1779–1805, 2023. doi: 10.1007/s10618-023-00939-3. URL https://link.springer.com/article/10.1007/s10618-023-00939-3.

Greg Diamos, Shubho Sengupta, Bryan Catanzaro, Mike Chrzanowski, Adam Coates, Erich Elsen, Jesse Engel, Awni Hannun, and Sanjeev Satheesh. Persistent RNNs: Stashing recurrent weights on-chip. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2024–2033. PMLR, 2016. URL https://proceedings.mlr.press/v48/diamos16.html.

Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, and Xiaoli Li. TSLANet: Rethinking Transformers for time series representation learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning*

Research, pp. 12409–12428. PMLR, 2024. URL https://proceedings.mlr.press/v235/eldele24a.html.

Luigi Fusco, Mikhail Khalilov, Marcin Chrapek, Giridhar Chukkapalli, Thomas Schulthess, and Torsten Hoefler. Understanding data movement in tightly coupled heterogeneous systems: A case study with the Grace Hopper Superchip. arXiv preprint arXiv:2408.11556, 2024. URL https://arxiv.org/abs/2408.11556.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. URL https://arxiv.org/abs/1603.08983.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Benhao Huang, Zhengyang Geng, and Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning. arXiv preprint arXiv:2605.21488, 2026. URL https://arxiv.org/abs/2605.21488. ICML 2026.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708, 2017. URL https://openaccess.thecvf.com/content_cvpr_2017/html/Huang_Densely_Connected_Convolutional_CVPR_2017_paper.html.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. arXiv preprint arXiv:2510.04871, 2025. URL https://arxiv.org/abs/2510.04871.

Yoo-Min Jung and Leekyung Kim. MambaSL: Exploring single-layer Mamba for time series classification. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/4e07dbaa17e8344d5ef752a9668f3be3-Abstract-Conference.html.

Mieszko Komisarczyk, Saurabh Mathur, Maurice Kraus, Sriraam Natarajan, and Kristian Kersting. Recursive inference machines for neural reasoning. arXiv preprint arXiv:2603.05234, 2026. URL https://arxiv.org/abs/2603.05234.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pp. 562–570. PMLR, 2015. URL https://proceedings.mlr.press/v38/lee15a.html.

Zhe Li, Xuanxuan Li, Zhongwen Rao, Lujia Pan, and Zenglin Xu. MTS-Mixers: Multivariate time series forecasting via factorized temporal and channel mixing. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2025. doi: 10.1109/IJCNN64981.2025.11229402. URL https://ieeexplore.ieee.org/document/11229402/.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. iTransformer: Inverted Transformers are effective for time series forecasting. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/2ea18fdc667e0ef2ad82b2b4d65147ad-Abstract-Conference.html.

Donghao Luo and Xue Wang. ModernTCN: A modern pure convolution structure for general time series analysis. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/86b1437c1e4c3b3c4debff98234a67e7-Abstract-Conference.html.

Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. HIVE-COTE 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110:3211–3243, 2021. doi: 10.1007/s10994-021-06057-9. URL https://link.springer.com/article/10.1007/s10994-021-06057-9.

Matthew Middlehurst, Ali Ismail-Fawaz, Antoine Guillaume, Christopher Holder, David Guijo-Rubio, Guzal Bulatova, Leonidas Tsaprounis, Lukasz Mentel, Martin Walter, Patrick Schäfer, and Anthony Bagnall. aeon: a Python toolkit for learning from time series. *Journal of Machine Learning Research*, 25(289):1–10, 2024. URL https://www.jmlr.org/papers/v25/23-1444.html.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with Transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Jbdc0vTOcol.

Lucas Prieto, Melih Barsbey, Pedro Mediano, and Tolga Birdal. Grokking at the edge of numerical stability. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c9e6ac15e689e06139d7b39e1667b165-Abstract-Conference.html.

Kaleem Ullah Qasim and Jiashu Zhang. Accelerating training speed of tiny recursive models with curriculum guided adaptive recursion. arXiv preprint arXiv:2511.08653, 2025. URL https://arxiv.org/abs/2511.08653.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model. arXiv preprint arXiv:2605.19943, 2026. URL https://arxiv.org/abs/2605.19943.

Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn Keogh. Generalizing DTW to the multi-dimensional case requires an adaptive approach. *Data Mining and Knowledge Discovery*, 31(1):1–31, 2017. doi: 10.1007/s10618-016-0455-0. URL https://link.springer.com/article/10.1007/s10618-016-0455-0.

Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I Webb. MultiRocket: multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, 36:1623–1646, 2022. doi: 10.1007/s10618-022-00844-1. URL https://link.springer.com/article/10.1007/s10618-022-00844-1.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. arXiv preprint arXiv:1609.03499, 2016. URL https://arxiv.org/abs/1609.03499.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. arXiv preprint arXiv:2506.21734, 2025a. URL https://arxiv.org/abs/2506.21734.

Shiyu Wang, Jiawei Li, Xiaoming Shi, Zhou Ye, Baichuan Mo, Wenze Lin, Shengtong Ju, Zhixuan Chu, and Ming Jin. TimeMixer++: A general time series pattern machine for universal predictive analysis. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/2b187165e28fdfdc0ffb34d1bfff2b0c-Abstract-Conference.html.

Yunshi Wen, Tengfei Ma, Ronny Luss, Debarun Bhattacharjya, Achille Fokoue, and Anak Agung Julius. Shedding light on time series classification using interpretability gated networks. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7ca55dae829a2305716f34658cd33a97-Abstract-Conference.html.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. doi: 10.2307/3001968. URL https://www.jstor.org/stable/3001968.

Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. ETSformer: Exponential smoothing Transformers for time-series forecasting. arXiv preprint arXiv:2202.01381, 2022. URL https://arxiv.org/abs/2202.01381.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. TimesNet: Temporal 2D-variation modeling for general time series analysis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ju_Uqw384Oq.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are Transformers effective for time series forecasting? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 11121–11128, 2023. doi: 10.1609/aaai.v37i9.26317. URL https://ojs.aaai.org/index.php/AAAI/article/view/26317.

Tianping Zhang, Yizhuo Zhang, Wei Cao, Jiang Bian, Xiaohan Yi, Shun Zheng, and Jian Li. Less Is More: Fast multivariate time series forecasting with light sampling-oriented MLP structures. arXiv preprint arXiv:2207.01186, 2022. URL https://arxiv.org/abs/2207.01186.

Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=vSVLM2j9eie.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency enhanced decomposed Transformer for long-term series forecasting. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27268–27286. PMLR, 2022. URL https://proceedings.mlr.press/v162/zhou22g.html.

Tian Zhou, Peisong Niu, Xue Wang, Liang Sun, and Rong Jin. One Fits All: Power general time series analysis by pretrained LM. In *Advances in Neural Information Processing Systems*, volume 36, pp. 43322–43355, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/86c17de05579cde52025f9984e6e2ebb-Abstract-Conference.html.

A Experimental setup

A.1 Protocol and Implementation Specifications

Implementation and Computational Environment The published MambaSL benchmark results reused in this work were obtained using Python 3.12.8 and PyTorch 2.5.1, primarily on NVIDIA GTX 1080 Ti GPUs, with A100 GPUs used for memory-intensive runs. Classical baselines were evaluated with `aeon` toolkit (Middlehurst et al., 2024), while most deep-learning models used TSLib or adapted official implementations (Jung & Kim, 2026). RTSM was evaluated using a separate training pipeline with persistent batch slots and per-sample trajectory accounting (Section 3.4). Samples may halt after different numbers of supervision steps, so the loop tracks completed trajectories separately from optimizer updates. All newly conducted experiments were run on NVIDIA GH200 GPUs (Fusco et al., 2024), including RTSM under both protocols and the validation-protocol re-evaluation of the deep-learning baselines using the corresponding MambaSL implementations adapted to the validation-based training and selection procedure.

Epoch-Exact Slot-Buffer Accounting Adaptive halting allows different samples to complete their training trajectories after different numbers of supervision steps. The slot-buffer scheduler maintains a completion counter e_i for each training sample i, with E denoting the target number of epochs. A sample remains in its physical batch slot until it halts; after the corresponding optimizer update, its completed trajectory increments e_i and the slot is refilled from the set of unfinished samples. Samples reaching $e_i = E$ are removed from the refill pool, and training terminates once $e_i = E$ for all samples. If training reaches this budget without protocol-specific early stopping, every sample completes exactly E adaptive trajectories regardless of its halting depth, avoiding depth-dependent over- or undersampling. Non-halted samples retain their slot assignment and latent state across supervision steps, while newly assigned samples are initialized with the model's default latent state.

MambaSL Protocol The *MambaSL protocol* follows the evaluation procedure of MambaSL (Jung & Kim, 2026). Models are trained on the complete official UEA training partition, while the official test partition is evaluated during training and used for model and checkpoint selection as well as early stopping. We reuse the published results for MambaSL and its 20 baselines and evaluate RTSM under the same selection procedure.

Validation Protocol The *validation protocol* removes test-set involvement in model selection by partitioning the official UEA training set into an 80/20 stratified train-validation split. The official test partition is reserved for final evaluation, while model configurations and checkpoints are selected by highest validation accuracy, using validation loss as a tie-breaker between configurations. No early stopping is used under this protocol. We re-evaluate the deep-learning baselines and RTSM under this protocol while retaining their respective model search spaces. The validation-protocol comparison contains only the deep-learning models; the five non-deep-learning baselines are included in the separate MambaSL-protocol comparison.

Normalization. The reported RTSM benchmark includes two historical normalization implementations: statistics were fitted either on the full official training partition before the 80/20 split or on the 80% training subset after splitting. Both apply the fitted statistics to held-out data; neither fits the RTSM normalizer on the official test set. The shared UEA loader used for the re-evaluated baselines standardizes each partition independently, so validation and test statistics are not used to normalize training inputs. Model-specific transforms follow the respective baseline implementations. The comparison therefore reflects the complete model, training, and preprocessing pipelines.

Protocol Hyperparameters The basic training settings used across the four experimental groups are summarized in Table 4. For the deep-learning baselines, we retain the standardized settings of MambaSL (Jung & Kim, 2026) and modify only protocol-dependent choices where required by the validation setup. RTSM uses its own training configuration under both protocols.

Table 4: Basic training settings under the MambaSL and validation protocols.

	MambaSL Protocol		Validation Protocol	
Setting	DL	RTSM	DL	RTSM
Train epochs	100	100	100	100
Seed	2021	2021	2021	2021
Batch size	16†	dataset-specific	16	dataset-specific
Learning rate	10^{-3}	10^{-4}	10^{-3}	10^{-4}
LR scheduler	none	constant	none	constant
Optimizer	RAdam	AdamW	RAdam	AdamW
Patience	10	10	–	–
Dropout	0.1	0.0	0.1	0.0

† Smaller batch sizes are used where required by memory constraints.

A.2 Shared Benchmark Specification

Compared Methods The validation-protocol comparison includes RTSM and 16 deep-learning baselines from MambaSL (Jung & Kim, 2026): the linear and MLP-based models DLinear (Zeng et al., 2023), LightTS (Zhang et al., 2022), and MTS-Mixers (Li et al., 2025); models using convolutional, spectral, or multiscale processing, including TimesNet (Wu et al., 2023), ModernTCN (Luo & Wang, 2024), TSLANet (Eldele et al., 2024), and TimeMixer++ (Wang et al., 2025b); the attention-based FEDformer (Zhou et al., 2022), ETSformer (Woo et al., 2022), Crossformer (Zhang & Yan, 2023), PatchTST (Nie et al., 2023), GPT4TS (Zhou et al., 2023), and iTransformer (Liu et al., 2024); the shape-based InterpGN (Wen et al., 2025); and TSCMamba (Ahamed & Cheng, 2025) and MambaSL. The MambaSL-protocol comparison additionally includes five non-deep-learning methods: nearest neighbor with dependent multivariate DTW (DTW_D) (Shokoohi-Yekta et al., 2017), ROCKET (Dempster et al., 2020), HIVE-COTE 2.0 (HC2; Middlehurst et al., 2021), Hydra (Dempster et al., 2023), and MultiRocket (Tan et al., 2022) combined with Hydra.

Metrics Following the reporting convention of MambaSL (Jung & Kim, 2026), we use the following evaluation metrics:

- **avg. acc:** Mean classification accuracy (%) across all 30 UEA datasets.
- **avg. acc (10):** Mean classification accuracy (%) across the commonly used 10-dataset UEA subset.
- **avg. rank:** Mean per-dataset rank across all 30 UEA datasets. For the reported rank summaries, accuracies are rounded to two decimal places before ranking; ties receive the minimum occupied rank (e.g., 1, 1, 3). Ranks are computed among the models evaluated under each protocol.
- **# of top-N:** Number of datasets on which a model ranks among the top N methods under the same rounding and tie convention.
- **# of win/draw:** Number of datasets on which RTSM achieves higher or equal accuracy than the compared model.
- **# of loss/draw:** Number of datasets on which RTSM achieves lower or equal accuracy than the compared model.
- **rank test:** One-sided pairwise p-value of the Wilcoxon signed-rank test (Wilcoxon, 1945), with the alternative that RTSM has higher per-dataset accuracy than the compared model. These tests accompany the descriptive accuracy summaries.

Dataset The UEA archive (Bagnall et al., 2018) comprises 30 multivariate time-series classification datasets, summarized in Table 5, with a subset of 10 datasets (`EC`, `FD`, `HW`, `HB`, `JV`, `PEMS`, `SRS1`, `SRS2`, `SAD`, and `UW`) commonly used in recent TSC benchmarking through `TSLib`.

Table 5: Overview of the 30 UEA datasets.

Dataset	Abbr.	Train Cases	Test Cases	Length	Channels	Classes
EthanolConcentration	EC	261	263	1751	3	4
FaceDetection	FD	5890	3524	62	144	2
Handwriting	HW	150	850	152	3	26
Heartbeat	HB	204	205	405	61	2
JapaneseVowels	JV	270	370	7–29	12	9
PEMS-SF	PEMS	267	173	144	963	7
SelfRegulationSCP1	SRS1	268	293	896	6	2
SelfRegulationSCP2	SRS2	200	180	1152	7	2
SpokenArabicDigits	SAD	6599	2199	4–93	13	10
UWaveGestureLibrary	UW	120	320	315	3	8
ArticularyWordRecognition	AWR	275	300	144	9	25
AtrialFibrillation	AF	15	15	640	2	3
BasicMotions	BM	40	40	100	6	4
CharacterTrajectories	CT	1422	1436	60–182	3	20
Cricket	CR	108	72	1197	6	12
DuckDuckGeese	DDG	50	50	270	1345	5
EigenWorms	EW	128	131	17984	6	5
Epilepsy	EP	137	138	206	3	4
ERing	ER	30	270	65	4	6
FingerMovements	FM	316	100	50	28	2
HandMovementDirection	HMD	160	74	400	10	4
InsectWingbeat	IW	25000	25000	2–22	200	10
Libras	LIB	180	180	45	2	15
LSST	LSST	2459	2466	36	6	14
MotorImagery	MI	278	100	3000	64	2
NATOPS	NATO	180	180	51	24	6
PenDigits	PD	7494	3498	8	2	10
PhonemeSpectra	PS	3315	3353	217	11	39
RacketSports	RS	151	152	30	6	4
StandWalkJump	SWJ	12	15	2500	4	3
Min		12	15	2	2	2
Max		25000	25000	17984	1345	39

A.3 MODEL HYPERPARAMETERS FOR GRID SEARCH

For the baseline models, we retain the hyperparameter search spaces defined in the MambaSL benchmark (Jung & Kim, 2026), which were derived primarily from the respective original papers, official implementations, and `TSLib` configurations. For forecasting-origin and foundation models without dedicated classification settings, MambaSL additionally considered more compact configurations suitable for the generally smaller UEA datasets. The resulting searches cover more than 200 candidate configurations per model-dataset pair in most cases. We use the same search spaces under both evaluation protocols and list them below together with the corresponding RTSM hyperparameters.

- **Non-DL**
  - DTW_D: fixed implementations (no learnable hyperparameters).
  - ROCKET, HC2, Hydra, MR+Hydra: 3 combinations
    - `seed`: 0, 1, 2
    - Other parameters use the fixed configurations reported in the MambaSL benchmark.
- **Linear and MLP-based models**
  - DLinear: 15 combinations
    - `moving_avg`: 0.5%, 1%, 2%, 3%, 4%, 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50% of sequence length
    - Only the moving-average window is varied, giving 15 configurations.
  - LightTS: 100 combinations
    - `d_model`: 32, 64, 128, 256, 512
    - `chunk_size`: 1/21 – 1/2 of sequence length
    - The two searched hyperparameters give 100 configurations.
  - MTS-Mixers: 256 combinations
    - `e_layers`: 2
    - `d_model`: 128, 256, 512, 1024
    - `d_ff`: 0, 2, 4, 8, 16, 32, 64, 128
    - `fac_C`: 0 (False) if `d_ff` is 0, else 1 (True)
    - `down_sampling_window`: 0, 1%, 2%, 3%, 5%, 7.5%, 10%, 12.5% of sequence length
    - `fac_T`: 0 (False) if `down_sampling_window` is 0, else 1 (True)
    - `use_norm`: 1 (True)

- **Convolutional, spectral, and multiscale models**
  - TimesNet: 252 combinations
    - `e_layers`: 2, 3, 4
    - `(d_model, d_ff)`: (8,16), (8,32), (8,64), (16,16), (16,32), (16,64), (32,32), (32,64), (32,128), (64,64), (64,128), (64,256), (128,128), (128,256)
    - `top_k`: 1, 2, 3
    - `num_kernels`: 4, 6
  - ModernTCN: 252 combinations
    - `ffn_ratio`: 1, 2, 4
    - `patch_size`: 2.5%, 5%, 7.5%, 10%, 15%, 20%, 25% of sequence length
    - `patch_stride`: 50% of `patch_size`
    - `(num_blocks, large_size, small_size, dims)`: (“1”, “13”, “5”, “32”), (“1”, “13”, “5”, “64”), (“1”, “13”, “5”, “128”), (“1”, “13”, “5”, “256”), (“1 1”, “9 9”, “5 5”, “32 64”), (“1 1”, “9 9”, “5 5”, “64 128”), (“1 1”, “9 9”, “5 5”, “128 256”), (“1 1”, “13 13”, “5 5”, “32 64”), (“1 1”, “13 13”, “5 5”, “64 128”), (“1 1”, “13 13”, “5 5”, “128 256”), (“1 1 1”, “9 9 9”, “5 5 5”, “32 64 128”), (“1 1 1”, “13 13 13”, “5 5 5”, “32 64 128”)
  - TSLANet: 252 combinations
    - `depth`: 1, 2, 3
    - `emb_dim`: 32, 64, 128, 256
    - `mlp_ratio`: 1, 2, 3
    - `patch_size`: 2.5%, 5%, 7.5%, 10%, 15%, 20%, 25% of sequence length
    - `patch_stride`: 50% of `patch_size`
    - `masking_ratio`: 0.4
    - `ICB`: 1 (True)
    - `ASB`: 1 (True)
    - `adaptive_filter`: 1 (True)
    - `load_from_pretrained`: 1 (True)
    - `pretrain_lr`: 0.001
    - `pretrain_epoch`: 50
  - TimeMixer++: 216 combinations
    - `down_sampling_method`: “conv”
    - `down_sampling_layers`: 1, 2, 3
    - `down_sampling_window`: 2
    - `num_kernels`: 6
    - `e_layers`: 1, 2, 3, 4
    - `d_model`: 16, 32, 64
    - `d_ff`: 16, 32, 64
    - `n_heads`: 8
    - `top_k`: 2, 3

- **Transformer-based models**
  - FEDformer: 250 combinations
    - `moving_avg`: 0.5%, 1%, 2%, 3%, 5%, 10%, 20%, 30%, 40%, 50% of sequence length
    - `e_layers`: 2
    - `d_model`: 32, 64, 128, 256, 512
    - `d_ff`: 128, 256, 512, 1024, 2048
    - `n_heads`: 8
  - ETSformer: 240 combinations
    - `e_layers`: 1, 2
    - `d_model`: 32, 64, 128, 256, 512
    - `d_ff`: 64, 128, 256, 512, 1024, 2048
    - `n_heads`: 8
    - `top_k`: 1, 2, 3, 4
  - Crossformer: 288 combinations
    - `e_layers`: 1, 2, 3
    - `d_model`: 32, 64, 128, 256
    - `d_ff`: 64, 128, 256, 512
    - `n_heads`: 4
    - `factor`: 3, 10
    - `seg_len`: 6, 12, 24 (change the code of `TSLib` to modify the fixed value)
  - PatchTST: 252 combinations
    - `e_layers`: 1, 2, 3
    - `d_model`: 16, 32, 64, 128
    - `d_ff`: 64, 128, 256
    - `n_heads`: 4 if `d_model` $\in \{16, 32\}$ else 16
    - `patch_size`: 2.5%, 5%, 7.5%, 10%, 15%, 20%, 25% of sequence length
    - `patch_stride`: 50% of `patch_size`
  - GPT4TS: 252 combinations
    - `e_layers`: 3, 4, 5, 6
    - `d_model`: 768 (fixed since the model use pretrained GPT-2)
    - `d_ff`: 768 (fixed since the model use pretrained GPT-2)
    - `patch_size`: 1/25 – 1/5 of sequence length
    - `patch_stride`: 25%, 50%, 100% of `patch_size`
  - iTransformer: 240 combinations
    - `e_layers`: 1, 2, 3, 4
    - `d_model`: 64, 128, 256, 512, 1024, 2048
    - `d_ff`: 64, 128, 256, 512, 1024
    - `n_heads`: 8
    - `factor`: 1, 3

- **Shape-based model**
  - InterpGN: 243 combinations
    - `dnn_type`: "FCN"
    - `num_shaplet`: 5, 10, 15
    - `lambda_div`: 0, 0.1, 1.0
    - `lambda_reg`: 0, 0.1, 1.0
    - `epsilon`: 0.5, 1.0, 2.0
    - `gating_value`: 0.5, 0.75, 1.0
    - `distance_func`: "euclidean"
    - `memory_efficient`: False
    - `sbm_cls`: "linear"

- **Mamba-based models**
  - TSCMamba
    - `d_model`: 64, 128, 256
    - `e_layers`: 1, 2
    - `expand`: 1, 2
    - `d_conv`: 2, 4
    - `d_ff` (`d_state`): 32, 64, 128
    - `no_rocket`: 0 (False)
    - `half_rocket`: 0 (False), 1 (True)
    - `additive_fusion`: 0 (multiplicative fusion), 1 (additive fusion)
    - `max_pooling`: 0 (avg pooling), 1 (max pooling)
    - `channel_token_mixing`: 0 (False)
    - `only_forward_scan`: 0 (False)
    - `flip_dir`: 2 (vertical flip)
    - `reverse_flip`: 0 (False)
    - `patch_size`: 8
    - `patch_stride`: 8
    - `rescale_size`: 64
    - `variation`: 64
    - `initial_focus`: 1.0
  - MambaSL: 240 combinations
    - `d_model` (d_{m}): 32, 64, 128, 256, 512, 1024
    - `d_ff` (d_{s}): 1, 2, 4, 8, 16
    - `tv_dt` (θ_{Δ}): 0 (False), 1 (True)
    - `tv_B` ($\theta_{\boldsymbol{B}}$): 0 (False), 1 (True)
    - `tv_C` ($\theta_{\boldsymbol{C}}$): 0 (False), 1 (True)
    - `use_D`: 0 (False)
    - `expand`: 1
    - `d_conv`: 4

- **Recursive-reasoning-based models**
  - RTSM (ours): 240 combinations
    - `Batch size`: 8, 16, 32, 64
    - `patch_length`: 1, 2, 3, 4, 8, 10, 16, 24, 48, 64, 128, 400
    - `patch_stride`: 1, 2, 3, 8, 10

B EXPERIMENTAL RESULTS

B.1 FULL RESULTS ON THE UEA BENCHMARK UNDER MAMBASL PROTOCOL

Table 6 reports the complete per-dataset results under the MambaSL protocol. The baseline results are taken from the original MambaSL benchmark (Jung & Kim, 2026), while RTSM is evaluated under the same training and selection procedure. Full protocol and implementation details are provided in Appendix A.

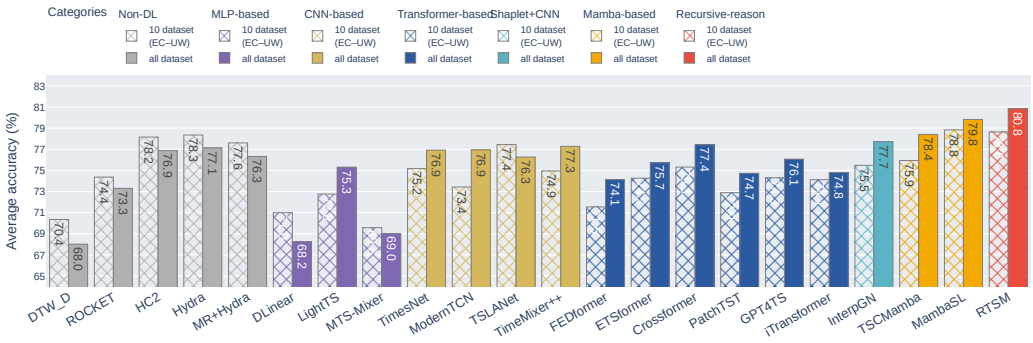


Figure 4: Mean classification accuracy (%) under the MambaSL protocol, which uses official test-set performance for configuration and checkpoint selection. The comparison includes RTSM, 16 other deep-learning models, and five non-deep-learning baselines. Solid bars show averages across all 30 UEA datasets; hatched bars show the 10-dataset subset. Each dataset receives equal weight within its respective average.

Table 6: Classification accuracy (%) of models on the 30 datasets of the UEA archive under the MambaSL protocol. The best and the second-best are highlighted in bold and underline, respectively.

(a) Classification results from DTW$_D$ to TimeMixer++.

	non-DL					Linear/MLP			Convolutional/spectral/multiscale			
	DTW$_D$ (2017)	ROCKET (2020)	HC2 (2021)	Hydra (2023)	MR+ Hydra (2023)	DLinear (2023)	LightTS (2022)	MTS-Mixers (2025)	TimesNet (2023)	Modern TCN (2024)	TSLANet (2024)	Time Mixer++ (2025b)
EC	32.319	44.487	52.852	54.373	**60.076**	31.179	32.319	30.798	33.460	32.319	31.939	34.601
FD	52.866	61.606	61.379	60.982	61.294	69.495	68.473	68.530	70.148	66.998	66.969	69.665
HW	60.706	59.294	56.000	56.353	53.529	23.765	25.059	19.647	37.529	30.118	62.000	33.647
HB	71.707	75.610	78.537	76.585	77.073	76.585	77.561	79.024	**83.902**	78.049	79.512	80.488
JV	95.946	83.784	98.378	97.838	98.378	96.486	96.757	95.946	98.649	98.649	98.919	99.189
PEMS	71.098	84.393	97.688	**98.266**	82.659	82.659	87.861	82.081	87.861	86.705^{A}	89.017	89.017
SRS1	77.474	85.666	89.078	86.689	**94.881**	92.833	92.833	78.840	92.150	93.515	89.761	92.150
SRS2	53.889	55.556	54.444	58.889	55.000	57.222	59.444	58.333	60.000	60.000	63.889	60.556
SAD	97.226	99.227	99.136	99.136	99.045	96.953	98.226	99.454	99.545	99.500	98.045	99.545
UW	90.313	94.063	94.063	94.375	94.063	82.813	89.063	83.125	88.438	88.438	94.375	90.625
AWR	98.667	**99.333**	**99.333**	**99.333**	**99.333**	97.667	98.333	99.000	98.667	98.667	**99.333**	99.000
AF	20.000	6.667	20.000	26.667	6.667	53.333	73.333	60.000	66.667	73.333	66.667	66.667
BM	97.500	**100.000**	**100.000**	**100.000**	**100.000**	87.500	**100.000**	**100.000**	**100.000**	**100.000**	**100.000**	**100.000**
CT	98.816	99.304	99.304	99.164	99.373	98.120	98.955	97.981	98.747	99.304	99.164	99.373
CR	**100.000**	**100.000**	**100.000**	**100.000**	98.611	91.667	93.056	97.222	97.222	97.222	91.667	98.611
DDG	58.000	52.000	70.000	68.000	60.000	66.000	56.000	68.000^{8}	68.000	66.000^{A}	74.000	74.000
EW	61.832	91.603	97.710	**98.473**	**98.473**	42.748^{A}	54.198^{8}	66.412	63.359	64.122	81.679	61.069^{A}
EP	96.377	99.275	**100.000**	**100.000**	**100.000**	65.217	97.101	66.667	96.377	96.377	**100.000**	94.928
ER	91.481	98.889	98.889	98.889	**99.259**	91.852	94.444	77.778	95.556	97.407	85.926	96.296
FM	53.000	57.000	61.000	55.000	60.000	62.000	64.000	63.000	66.000	67.000	68.000	67.000
HMD	18.919	51.351	48.649	51.351	36.486	66.216	66.216	70.270	**72.973**	66.216	48.649	**72.973**
IW	48.692	38.524	66.796	67.052	65.744	18.996	71.676	72.424	64.024	**75.588**	10.012	64.192
LIB	87.222	90.556	94.444	93.333	93.333	70.556	89.444	34.444	88.333	90.000	95.556	88.333
LSST	55.109	64.031	66.383	66.504	64.274	34.023	41.525	34.185	44.120	43.593	51.014	44.444
MI	50.000	54.000	50.000	54.000	51.000	64.000	68.000	66.000	68.000	67.000	**71.000**	67.000
NATO	88.333	89.444	91.111	90.000	93.333	96.111	97.222	95.000	98.333	95.556	**99.444**	96.111
PD	97.713	98.228	97.656	97.856	97.284	92.481	96.998	46.941	98.914	98.513	99.028	98.599
PS	15.121	27.826	33.015	33.164	**33.642**	6.651	16.224	5.428	14.524	13.749	4.921	15.508
RS	80.263	90.789	90.132	91.447	90.132	78.947	81.579	86.842	89.474	84.211	**94.079**	91.447
SWJ	20.000	46.667	40.000	40.000	66.667	53.333	73.333	66.667	66.667	80.000	73.333	73.333
avg. acc (10)	70.354	74.368	78.156	78.349	77.600	70.999	72.760	69.578	75.168	73.429	77.443	74.948
avg. acc	68.020	73.306	76.866	77.124	76.320	68.247	75.308	69.001	76.921	76.938	76.263	77.279
avg. rank	17.00	11.90	9.50	9.47	10.10	17.73	12.57	14.77	9.33	9.80	8.40	7.70
# of top-1	1	3	4	6	**8**	0	1	1	3	2	6	2
# of top-5	2	10	12	13	12	0	5	3	9	8	15	11
# of top-10	4	13	19	16	16	4	9	8	15	14	20	24
# of win/draw	30	25	21	22	23	28	24	28	22	24	21	24
# of lose/draw	1	9	12	11	9	2	8	4	10	10	12	10
rank test	0.000	0.000	0.022	0.024	0.014	0.000	0.001	0.000	0.006	0.015	0.019	0.005

All experiments were run on 1080 Ti with batch size 16 except below:

[8] The experiments were run with batch size 8

[4] The experiments were run with batch size 4

[A] The experiments were run on A100 GPU

Table 6: *(Continued.)* Classification accuracy (%) of models on the 30 datasets of the UEA archive under the MambaSL protocol. The best and the second-best are highlighted in bold and underline, respectively.

(b) Classification results from FEDformer to RTSM. "*" indicates "former" in Transformer-based models.

	Transformer-based						Shape-based	Mamba-based		Recursive-reasoning-based
	FED* (2022)	ETS* (2022)	Cross* (2023)	PatchTST (2023)	GPT4TS (2023)	iTrans* (2024)	InterpGN (2025)	TSC Mamba (2025)	MambaSL (2026)	**RTSM (ours)**
EC	33.840	33.080	43.346	32.700	32.319	31.939	28.897	30.798	42.586	36.880
FD	69.892	68.331	69.637	68.076	68.615	68.417	63.791	**70.204**	69.296	69.550
HW	33.412	35.647	30.353	29.529	34.118	31.294	58.706	45.059	60.824	**63.530**
HB	78.049	78.537	79.024	73.659	79.024	78.049	80.488	78.537	80.488	80.980
JV	97.568	98.378	98.378	97.027	98.108	98.108	**99.730**	98.378	98.649	**99.730**
PEMS	89.017	87.861	89.595[8]	89.017[4]	88.439	91.329[8]	86.127	90.173	85.549	91.910
SRS1	75.085	92.150	93.857	89.078	93.174	93.857	90.444	92.833	92.491	90.780
SRS2	59.444	60.000	61.667	61.667	60.556	58.889	58.333	62.222	**65.000**	61.670
SAD	99.454	99.000	98.863	98.818	99.591	99.318	99.864	95.816	**99.955**	99.500
UW	79.688	89.688	88.438	89.375	89.063	90.000	88.438	**95.313**	93.438	91.870
AWR	92.333	98.667	98.667	98.667	98.667	99.000	99.000	**99.333**	**99.333**	**99.330**
AF	73.333	**80.000**	73.333	**80.000**	66.667	66.667	46.667	66.667	53.333	60.000
BM	95.000	**100.000**	**100.000**	85.000	**100.000**	97.500	**100.000**	**100.000**	**100.000**	**100.000**
CT	97.284	98.468	98.955	98.398	99.373	98.886	**99.791**	99.513	99.721	99.720
CR	90.278	98.611	97.222	97.222	97.222	97.222	**100.000**	**100.000**	**100.000**	**100.000**
DDG	42.000	42.000	72.000[4]	40.000[4]	66.000	60.000[8]	76.000[A]	50.000	70.000	**78.000**
EW	63.359[4]	70.992[4]	58.015[4]	57.252	66.412	59.542	82.443[A]	88.550	83.969[4]	90.840
EP	92.754	92.754	95.652	98.551	89.855	80.435	97.826	99.275	97.826	99.280
ER	95.926	96.296	97.407	96.667	96.296	94.815	87.778	97.037	93.704	93.700
FM	**72.000**	66.000	64.000	63.000	66.000	64.000	67.000	66.000	71.000	67.000
HMD	71.622	63.514	66.216	63.514	70.270	58.108	45.946	60.811	70.270	63.510
IW	65.244	64.868	73.712	60.936	61.396	73.436	67.180	68.244[*]	66.304	66.400
LIB	93.333	90.000	90.556	81.667	87.778	91.667	**97.222**	91.667	91.667	93.890
LSST	40.998	41.241	41.768	56.732	38.078	40.673	59.570	60.057	45.580	**70.110**
MI	65.000	67.000[8]	68.000[4]	69.000	67.000	67.000	67.000[A]	67.000	69.000	67.000
NATO	96.111	93.333	93.889	80.556	96.111	88.889	98.889	92.222	98.889	98.330
PD	98.599	98.542	98.513	98.485	98.513	98.428	99.342	98.513	99.257	**99.370**
PS	11.691	14.316	14.256	15.091	13.123	11.542	33.075	23.800	30.331	26.250
RS	84.868	85.526	87.500	85.526	86.842	81.579	91.447	90.789	92.763	92.110
SWJ	66.667	66.667[8]	80.000	**86.667**	73.333	73.333	60.000	73.333	73.333	73.330
avg. acc (10)	71.545	74.267	75.316	72.895	74.301	74.120	75.482	75.933	**78.827**	78.640
avg. acc	74.128	75.716	77.427	74.729	76.065	74.797	77.700	78.405	79.819	**80.819**
avg. rank	13.17	11.43	9.13	13.40	10.23	12.63	8.50	7.67	5.73	**5.07**
# of top-1	1	2	1	2	1	0	5	5	5	**8**
# of top-5	5	2	11	4	6	4	13	10	17	**19**
# of top-10	10	12	16	8	15	9	20	25	25	**26**
# of win/draw	25	27	21	26	25	26	24	23	19	
# of lose/draw	5	6	11	6	8	6	11	13	17	
rank test	0.000	0.000	0.069	0.001	0.001	0.001	0.002	0.020	0.355	

The MambaSL benchmark experiments were run on 1080 Ti with batch size 16 except below; RTSM was run on NVIDIA GH200.

[8] The experiments were run with batch size 8

[4] The experiments were run with batch size 4

[A] The experiments were run on A100 GPU

[*] Little hyperparameter search was performed since it took over two weeks conducting a single experiment.

B.2 Full Results on the UEA Benchmark under Validation protocol

Table 7 reports the complete per-dataset results for the evaluated deep-learning models under the validation protocol. Model and checkpoint selection are performed exclusively on the validation split, with the official test set reserved for final evaluation. Full protocol and implementation details are provided in Appendix A.

Table 7: Classification accuracy (%) of deep-learning models on the 30 UEA datasets under the validation protocol. Best and second-best results are shown in bold and underlined, respectively.

	Linear/MLP			Convolutional/spectral/multiscale			
	DLinear (2023)	LightTS (2022)	MTS-Mixers (2025)	TimesNet (2023)	Modern TCN (2024)	TSLANet (2024)	Time Mixer++ (2025b)
EC	25.480	28.140	22.050	28.520	23.950	**30.420**	24.330
FD	67.820	64.560	65.580	66.030	64.100	65.490	64.390
HW	21.530	19.530	14.940	27.760	19.650	**56.350**	26.000
HB	73.170	73.170	73.660	**77.070**	68.780	**77.070**	75.610
JV	95.680	92.970	92.970	97.840	96.220	96.490	97.030
PEMS	72.250	74.570	72.830	76.880	81.500	80.350	80.350
SRS1	87.370	81.230	77.470	82.940	86.690	67.240	79.520
SRS2	50.000	53.890	48.890	49.440	52.780	51.110	52.780
SAD	95.180	98.180	98.680	99.140	97.140	99.320	99.140
UW	80.940	81.250	79.370	84.690	84.690	**90.310**	83.750
AWR	96.330	95.330	95.000	97.670	98.000	97.670	98.330
AF	33.330	33.330	20.000	40.000	20.000	33.330	26.670
BM	85.000	**100.000**	90.000	97.500	95.000	**100.000**	97.500
CT	97.080	98.470	97.560	97.350	98.120	98.470	98.400
CR	90.280	88.890	93.060	90.280	93.060	97.220	94.440
DDG	56.000	48.000	48.000	48.000	44.000	50.000	52.000
EW	28.240	47.330	49.620	51.910	50.380	70.230	51.150
EP	56.520	91.300	62.320	89.860	92.030	**98.550**	92.030
ER	92.220	94.440	68.890	93.330	**94.810**	72.960	92.960
FM	57.000	53.000	52.000	56.000	**62.000**	60.000	59.000
HMD	54.050	37.840	63.510	**64.860**	33.780	27.030	58.110
IW	18.280	70.330	70.700	62.280	**74.280**	10.080	61.250
LIB	68.890	79.440	29.440	82.220	85.000	91.110	77.220
LSST	32.850	20.560	8.960	33.290	36.540	56.810	35.850
MI	59.000	50.000	51.000	**60.000**	41.000	51.000	49.000
NATO	93.890	93.330	92.780	93.330	90.560	**97.780**	90.560
PD	92.140	96.430	44.170	98.310	97.740	98.200	97.970
PS	6.470	14.460	4.290	11.870	11.180	9.870	13.030
RS	72.370	73.030	83.550	84.210	76.970	90.790	80.260
SWJ	46.670	33.330	26.670	33.330	26.670	**53.330**	40.000
avg. acc (10)	66.942	66.749	64.644	69.031	67.550	71.415	68.290
avg. acc	63.534	66.211	59.932	69.197	66.554	69.286	68.288
avg. rank	11.13	10.03	12.30	7.03	9.27	6.07	7.90
# top-1	0	1	0	3	3	**8**	0
# top-5	6	7	2	9	7	17	10
# top-10	12	14	11	23	19	**25**	22
# win/draw	22	24	26	21	25	19	24
# lose/draw	8	7	5	9	7	14	8
rank test	0.002	0.003	0.000	0.029	0.001	0.162	0.003

All deep-learning baselines and RTSM were run on NVIDIA GH200 GPUs.

Table 7: *(Continued.)* Classification accuracy (%) of deep-learning models on the 30 datasets of the UEA archive under the validation protocol. The best and the second-best are highlighted in bold and underline, respectively.

(a) Classification results from FEDformer to RTSM. "*" indicates "former" in Transformer-based models.

	Transformer-based						Shape-based	Mamba-based		Recursive-reasoning-based
	FED* (2022)	ETS* (2022)	Cross* (2023)	PatchTST (2023)	GPT4TS (2023)	iTrans* (2024)	InterpGN (2025)	TSC Mamba (2025)	MambaSL (2026)	**RTSM (ours)**
EC	25.860	23.570	**30.420**	26.620	23.570	27.380	25.480	25.100	27.000	30.418
FD	68.620	65.270	67.170	63.710	65.010	64.780	62.150	**69.100**	67.110	67.196
HW	24.820	28.710	22.590	20.940	26.710	23.410	50.820	26.470	50.350	47.412
HB	70.240	70.730	73.660	60.980	75.120	72.200	45.850	76.100	69.760	68.780
JV	96.220	96.220	95.680	93.780	97.300	90.540	**98.380**	97.030	96.760	98.108
PEMS	81.500	77.460	87.280	83.820	75.140	82.660	80.920	83.240	72.830	**87.861**
SRS1	69.970	80.890	81.230	82.250	86.010	**89.760**	72.700	85.670	73.380	82.253
SRS2	51.670	52.220	47.220	55.560	48.330	50.560	**56.110**	50.560	52.780	51.667
SAD	97.950	98.500	98.320	97.770	98.000	98.500	99.230	98.820	**99.590**	98.363
UW	75.620	85.000	82.810	85.620	83.440	84.380	85.000	86.560	88.440	85.938
AWR	88.000	97.330	98.000	95.330	97.670	97.000	98.330	96.670	98.000	**98.667**
AF	**46.670**	20.000	40.000	40.000	33.330	26.670	20.000	40.000	33.330	26.667
BM	95.000	87.500	92.500	52.500	90.000	90.000	**100.000**	95.000	**100.000**	**100.000**
CT	96.660	97.350	96.730	97.280	98.540	97.630	**99.370**	98.190	98.820	99.304
CR	86.110	94.440	84.720	91.670	95.830	87.500	97.220	95.830	**98.610**	97.222
DDG	22.000	16.000	58.000	20.000	50.000	52.000	**64.000**	40.000	52.000	62.000
EW	47.330	56.490	46.560	45.800	51.910	44.270	19.080	78.630	71.760	**84.733**
EP	88.410	90.580	94.200	94.200	81.160	73.910	97.830	94.200	95.650	94.928
ER	91.850	92.960	93.700	92.220	90.740	89.630	87.780	91.850	87.780	88.519
FM	58.000	51.000	55.000	51.000	56.000	59.000	59.000	61.000	55.000	55.000
HMD	58.110	44.590	41.890	41.890	41.890	32.430	37.840	56.760	58.110	58.108
IW	62.780	64.030	72.320	60.290	59.260	72.360	64.100	68.980	64.110	64.472
LIB	83.890	84.440	82.220	77.780	73.890	87.780	**95.000**	83.890	82.220	87.222
LSST	20.920	36.620	36.580	52.070	9.980	11.030	56.450	57.540	40.880	**65.369**
MI	48.000	55.000	58.000	52.000	53.000	48.000	48.000	51.000	49.000	54.000
NATO	87.220	88.330	86.670	77.220	92.780	82.780	95.000	89.440	95.560	95.556
PD	98.230	97.630	97.030	96.970	98.060	97.400	98.340	97.570	97.570	**98.799**
PS	10.410	11.300	10.680	12.320	11.060	9.870	**30.810**	19.890	28.390	24.217
RS	76.970	76.970	82.890	76.970	84.870	76.970	88.160	85.530	**91.450**	84.868
SWJ	20.000	40.000	26.670	33.330	40.000	20.000	46.670	33.330	40.000	26.667
avg. acc (10)	66.247	67.857	68.638	67.105	67.863	68.417	67.664	69.865	69.800	**71.800**
avg. acc	64.968	66.038	68.025	64.396	66.287	64.680	69.321	71.132	71.208	**72.810**
avg. rank	10.77	9.57	8.90	10.73	9.27	10.67	6.37	6.43	6.00	**4.87**
# of top-1	1	0	1	0	0	1	7	1	4	7
# of top-5	4	5	10	6	7	6	18	14	16	**21**
# of top-10	13	18	16	13	17	11	22	**25**	24	**25**
# of win/draw	25	24	25	26	24	23	18	19	19	
# of lose/draw	7	6	8	5	7	8	14	11	15	
rank test	0.000	0.000	0.006	0.000	0.002	0.000	0.123	0.087	0.238	

All deep-learning baselines and RTSM were run on NVIDIA GH200 GPUs.

B.3 Recursive Architecture and Training Ablations

Configuration names encode the corresponding architectural and training choices. `H` denotes the number of outer recursive cycles, `L` the number of inner cycles used by two-state variants, and `G` the number of H-cycles through which gradients are propagated; accordingly, `2G`/`3G` extend the gradient window. `stepsN` denotes the maximum number of deep-supervision steps. For two-state models, `shared` uses a single reasoning network f for both state updates, as in TRM (Jolicoeur-Martineau, 2025), whereas `separate` uses distinct f_H/f_L networks, implementing the network separation used by HRM (Wang et al., 2025a). All other training choices follow the TRM recipe, including its gradient schedule, loss, halting, and parameter EMA; hence, the separate-network variant is not an evaluation of HRM itself. `Conv` denotes the simple convolutional patch tokenizer, which uses strided 1-D convolutions to map local temporal windows into patch representations; configurations without this prefix use the proposed time-series tokenizer.

Additional suffixes denote further architectural or training modifications. `tokenizer_only` omits the attention-based refinement stage and applies the sequence-level prediction head directly to the tokenizer representation. `nocarry` resets the latent state between supervision steps. `forcefull` disables Q-based halting and forces every sample through the full training-step budget. `mlpt` replaces attention with a temporal MLP. `meanpooling` and `adaptivepooling_noQ` replace the default pooling operation. The latter applies learned weighted pooling only to the classification head, whereas the reference model applies learned weighted pooling to both the classification and halting heads. `rima` enables the learned EMA-style latent-state reweighting of RIMA (Komisarczyk et al., 2026), while `pdc` applies the Progressive Depth Curriculum of CGAR (Qasim & Zhang, 2025), gradually increasing recursive depth during training.

Table 8: Classification accuracy (%) across the recursive architecture and training ablations on eight UEA datasets (CT, DDG, HB, JV, LSST, NATO, PD, and SAD) under validation-based selection. Each seed's score is the equally weighted mean across these datasets. Entries report the mean ± sample standard deviation of the three scores for seeds 0, 1, and 2. Bold marks the reference configuration.

Configuration	Avg. Acc. (%)
Main-text ablations	
`Conv_tokenizer_only`	73.72 ± 0.71
`Conv_3H6L1G_steps16_shared`	78.96 ± 1.03
`Conv_3H6L1G_steps16_separate`	79.69 ± 0.62
`tokenizer_only`	84.79 ± 0.41
`3H6L1G_steps16_shared`	84.98 ± 1.04
`3H6L1G_steps16_separate`	84.29 ± 1.26
`3H1G_steps1`	85.98 ± 0.71
`3H1G_steps16_forcefull`	85.05 ± 0.91
`1H1G_steps16`	85.77 ± 0.59
`2H1G_steps16`	85.60 ± 0.42
`3H1G_steps16`	$\mathbf{86.10 \pm 0.31}$
`4H1G_steps16`	85.61 ± 0.75
`5H1G_steps16`	85.78 ± 0.64
Additional ablations	
`1H1G_steps16_nocarry`	86.00 ± 0.76
`3H1G_steps16_nocarry`	86.09 ± 0.59
`3H2G_steps16`	86.06 ± 0.63
`3H3G_steps16`	85.91 ± 0.33
`3H1G_steps16_meanpooling`	86.14 ± 0.33
`3H1G_steps16_adaptivepooling_noQ`	85.85 ± 0.20
`3H1G_steps16_mlpt`	83.85 ± 0.47
`3H1G_steps16_rima`	86.00 ± 0.55
`3H1G_steps16_pdc`	86.02 ± 0.28
`3H1G_steps16_pdc_rima`	85.96 ± 0.87

Table 9: Comparison of the proposed `3H1G_steps16` configuration with the `tokenizer_only` baseline across the eight ablation datasets. Results report classification accuracy (%) for each seed and the mean ± sample standard deviation across three seeds. In the Overall row, datasets are averaged equally within each seed before computing the mean and standard deviation. Δ denotes the difference in mean accuracy between the proposed configuration and the tokenizer-only baseline.

	3H1G_steps16				tokenizer_only				
Dataset	**Seed 0**	**Seed 1**	**Seed 2**	**Mean ± Std.**	**Seed 0**	**Seed 1**	**Seed 2**	**Mean ± Std.**	**Δ**
CharacterTrajectories	99.234	99.095	98.886	$\mathbf{99.072 \pm 0.175}$	95.891	96.031	96.379	96.100 ± 0.251	+2.971
PenDigits	98.571	98.742	99.085	$\mathbf{98.799 \pm 0.262}$	98.485	98.113	98.228	98.275 ± 0.190	+0.524
SpokenArabicDigits	98.408	98.681	98.590	$\mathbf{98.560 \pm 0.139}$	96.362	96.771	97.317	96.817 ± 0.479	+1.743
JapaneseVowels	97.297	97.838	96.757	$\mathbf{97.297 \pm 0.541}$	96.757	97.027	96.757	96.847 ± 0.156	+0.450
NATOPS	91.111	89.444	91.111	90.555 ± 0.962	89.444	92.778	93.333	$\mathbf{91.852 \pm 2.103}$	-1.296
Heartbeat	73.659	75.122	74.146	$\mathbf{74.309 \pm 0.745}$	73.659	77.561	71.707	$\mathbf{74.309 \pm 2.981}$	+0.000
LSST	67.680	66.951	67.883	$\mathbf{67.505 \pm 0.490}$	66.869	66.504	64.882	66.085 ± 1.058	+1.419
DuckDuckGeese	60.000	64.000	64.000	$\mathbf{62.667 \pm 2.309}$	62.000	56.000	56.000	58.000 ± 3.464	+4.667
Overall	85.745	86.234	86.307	$\mathbf{86.095 \pm 0.306}$	84.933	85.098	84.325	84.786 ± 0.407	**+1.310**

Table 10: Comparison of the proposed `3H1G_steps16` configuration with the `3H6L1G_steps16_shared` two-state variant across the eight ablation datasets. Results report classification accuracy (%) for each seed and the mean ± sample standard deviation across three seeds. In the Overall row, datasets are averaged equally within each seed before computing the mean and standard deviation. Δ denotes the difference in mean accuracy between the proposed configuration and the shared-network two-state variant.

	3H1G_steps16				3H6L1G_steps16_shared				
Dataset	**Seed 0**	**Seed 1**	**Seed 2**	**Mean ± Std.**	**Seed 0**	**Seed 1**	**Seed 2**	**Mean ± Std.**	**Δ**
CharacterTrajectories	99.234	99.095	98.886	$\mathbf{99.072 \pm 0.175}$	99.304	98.189	98.886	98.793 ± 0.563	+0.279
PenDigits	98.571	98.742	99.085	$\mathbf{98.799 \pm 0.262}$	98.856	98.571	98.370	98.599 ± 0.244	+0.200
SpokenArabicDigits	98.408	98.681	98.590	$\mathbf{98.560 \pm 0.139}$	97.954	98.090	98.363	98.136 ± 0.208	+0.424
JapaneseVowels	97.297	97.838	96.757	$\mathbf{97.297 \pm 0.541}$	98.108	96.486	96.486	97.027 ± 0.936	+0.271
NATOPS	91.111	89.444	91.111	$\mathbf{90.555 \pm 0.962}$	82.778	77.778	95.556	85.371 ± 9.168	+5.185
Heartbeat	73.659	75.122	74.146	$\mathbf{74.309 \pm 0.745}$	75.122	70.244	72.195	72.520 ± 2.455	+1.789
LSST	67.680	66.951	67.883	$\mathbf{67.505 \pm 0.490}$	66.464	65.815	67.843	66.707 ± 1.036	+0.797
DuckDuckGeese	60.000	64.000	64.000	$\mathbf{62.667 \pm 2.309}$	62.000	66.000	60.000	$\mathbf{62.667 \pm 3.055}$	+0.000
Overall	85.745	86.234	86.307	$\mathbf{86.095 \pm 0.306}$	85.073	83.897	85.962	84.977 ± 1.036	**+1.118**

Table 11: Comparison of the `3H6L1G_steps16_shared` two-state variant with the `tokenizer_only` baseline across the eight ablation datasets. Results report classification accuracy (%) for each seed and the mean ± sample standard deviation across three seeds. In the Overall row, datasets are averaged equally within each seed before computing the mean and standard deviation. Δ denotes the difference in mean accuracy between the shared-network two-state variant and the tokenizer-only baseline.

	3H6L1G_steps16_shared				tokenizer_only				
Dataset	**Seed 0**	**Seed 1**	**Seed 2**	**Mean ± Std.**	**Seed 0**	**Seed 1**	**Seed 2**	**Mean ± Std.**	**Δ**
CharacterTrajectories	99.304	98.189	98.886	$\mathbf{98.793 \pm 0.563}$	95.891	96.031	96.379	96.100 ± 0.251	+2.693
PenDigits	98.856	98.571	98.370	$\mathbf{98.599 \pm 0.244}$	98.485	98.113	98.228	98.275 ± 0.190	+0.324
SpokenArabicDigits	97.954	98.090	98.363	$\mathbf{98.136 \pm 0.208}$	96.362	96.771	97.317	96.817 ± 0.479	+1.319
JapaneseVowels	98.108	96.486	96.486	$\mathbf{97.027 \pm 0.936}$	96.757	97.027	96.757	96.847 ± 0.156	+0.180
NATOPS	82.778	77.778	95.556	85.371 ± 9.168	89.444	92.778	93.333	$\mathbf{91.852 \pm 2.103}$	-6.481
Heartbeat	75.122	70.244	72.195	72.520 ± 2.455	73.659	77.561	71.707	$\mathbf{74.309 \pm 2.981}$	-1.789
LSST	66.464	65.815	67.843	$\mathbf{66.707 \pm 1.036}$	66.869	66.504	64.882	66.085 ± 1.058	+0.622
DuckDuckGeese	62.000	66.000	60.000	$\mathbf{62.667 \pm 3.055}$	62.000	56.000	56.000	58.000 ± 3.464	+4.667
Overall	85.073	83.897	85.962	$\mathbf{84.977 \pm 1.036}$	84.933	85.098	84.325	84.786 ± 0.407	**+0.191**

B.4 Recursive Dynamics and Inference-Time Scaling Analysis

This appendix documents the experimental protocol, complete measurements, and validity diagnostics underlying Section 4.3. The interpretation of these results is given there; here we report what was measured and how.

B.4.1 Inference-time probe

Following the inference-time scaling formulations described in Section 2.2.1, we evaluate RTSM using stochastic perturbations only at inference. The checkpoints are not retrained with the randomized initialization (RI) and noise injection (NI) used by EqR, and our RI+NI probe is not an exact PTRM reproduction because it additionally randomizes the initial state.

All experiments use the validation-protocol checkpoints described in Appendix A and evaluate the official UEA test partitions. Runs completed for 28 of the 30 UEA datasets; EigenWorms (EW) and InsectWingbeat (IW) did not complete within the allocated compute budget and are excluded. All macro-averaged quantities weight these 28 datasets equally. Normalization differs between this probe and some historical benchmark runs. The depth–breadth comparisons therefore characterize the fixed checkpoints within the probe's preprocessing pipeline.

For each test sample we record one deterministic trajectory and $B_{\max} = 512$ stochastic trajectories, each evaluated for up to $D = 64$ recursive inference steps, each comprising three recursive cycles. The deterministic trajectory uses the checkpoint's standard initial state without perturbation. Each stochastic trajectory uses a fresh randomized initialization with $\sigma_{\mathrm{RI}} = 1.0$, followed by noise injection between recursive steps with $\sigma_{\mathrm{NI}} = 0.01$. Since the final RTSM is single-state, both interventions act on its recursive latent state. A single maximum-budget run per dataset is sufficient, because the trajectory at any smaller depth $d < D$ is a prefix of the recorded trajectory and any smaller breadth $B < B_{\max}$ is a subset of the recorded rollouts; all (D, B) combinations are therefore evaluated offline from the same run.

For a sample i, rollout b, and recursion step d, let $\hat{c}_{i,b}^{(d)}$ denote the predicted class, with true label c_i and n_{test} test samples. Oracle pass@B accuracy is

$$\mathrm{pass@}\, B(D) = \frac{1}{n_{\mathrm{test}}} \sum_{i=1}^{n_{\mathrm{test}}} \mathbb{I}\left[\exists\, b \le B : \hat{c}_{i,b}^{(D)} = c_i\right]. \tag{7}$$

This is an *oracle upper bound*: it reports whether at least one of the first B stochastic trajectories reaches the correct class, without requiring that this trajectory be identifiable at inference time. We define *headroom* as oracle pass@B accuracy minus deterministic accuracy at the same depth.

To characterize state stabilization, we record the per-sample RMS state-update magnitude

$$r_d = \sqrt{\frac{1}{M} \left\| y^{(d)} - y^{(d-1)} \right\|_F^2}, \tag{8}$$

where $y^{(d)}$ is the latent state after step d, M is its number of scalar elements, and $\| \cdot \|_F$ is the Frobenius norm. This is a state-stabilization diagnostic rather than a task metric.

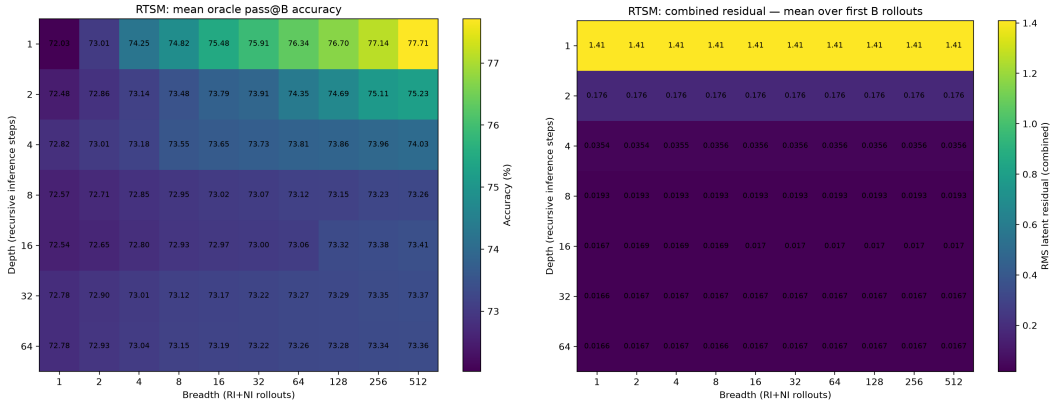
Three label-free selection rules are evaluated at the maximum inference budget. *Best-Q* selects the trajectory with the largest halting-head score, *majority* returns the modal class across trajectories, and *minimum residual* selects the trajectory with the smallest residual averaged over its final three recursive steps.

Validity check. Stochastic trajectories are evaluated in expanded batches for computational efficiency, whereas the deterministic trajectory is evaluated normally. To exclude batch-composition artifacts, we repeat the deterministic trajectory in the expanded rollout layout with RI and NI disabled and require identical predictions at every recursion step. This zero-noise equivalence check passed on all 28 datasets, with zero prediction mismatches across more than $50{,}000$ checked sample–trajectory rows.

B.4.2 FULL DEPTH–BREADTH GRID

Figure 5 and Table 12 report the complete 7×10 grid summarised in Section 4.3. The full grid supplements the selected budgets reported in the main text. At $B = 512$, oracle accuracy falls from 77.71% at $D = 1$ to 75.23% at $D = 2$ and 74.03% at $D = 4$, after which values vary within a narrow band without a consistent trend. Across the full grid, the highest oracle accuracy occurs at $D = 1$, $B = 512$ rather than at the largest joint depth–breadth budget.

The residual grid is comparatively simple. The mean stochastic residual falls from approximately 1.409 at $D = 1$ to 0.176 at $D = 2$, 0.0356 at $D = 4$, and 0.0193 at $D = 8$, settling near 0.0167 for $D \geq 16$. At every fixed depth, changing breadth from $B = 1$ to $B = 512$ leaves the mean residual essentially unchanged. In this probe, depth has the larger measured effect on the average state-update magnitude.



(a) Oracle pass@B accuracy.

(b) Mean latent-state residual.

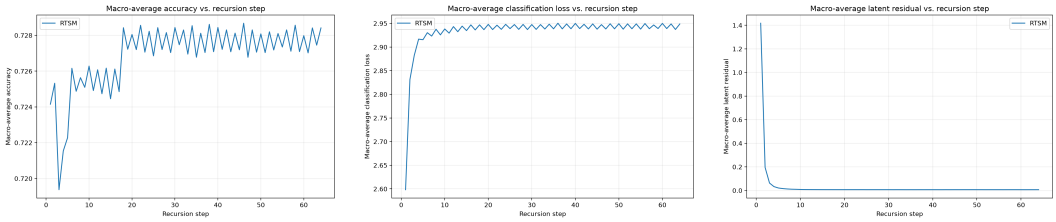
Figure 5: Complete breadth–depth grid for RTSM. Macro-averaged oracle pass@B accuracy and RMS latent-state residual for stochastic RI+NI trajectories across 28 UEA datasets.

Table 12: Macro-averaged oracle pass@B accuracy (%) across the full depth–breadth grid, over 28 UEA datasets.

$D \backslash B$	1	2	4	8	16	32	64	128	256	512
1	72.03	73.01	74.25	74.82	75.48	75.91	76.34	76.70	77.14	**77.71**
2	72.48	72.86	73.14	73.48	73.79	73.91	74.35	74.69	75.11	75.23
4	72.82	73.01	73.18	73.55	73.65	73.73	73.81	73.86	73.96	74.03
8	72.57	72.71	72.85	72.95	73.02	73.07	73.12	73.15	73.23	73.26
16	72.54	72.65	72.80	72.93	72.97	73.00	73.06	73.32	73.38	73.41
32	72.78	72.90	73.01	73.12	73.17	73.22	73.27	73.29	73.35	73.37
64	72.78	72.93	73.04	73.15	73.19	73.22	73.26	73.28	73.34	73.36

B.4.3 DETERMINISTIC RECURSIVE DYNAMICS

Figure 6 reports the full 64-step traces of classification accuracy, cross-entropy loss, and latent residual along the deterministic trajectory; Table 13 lists selected steps. The residual decreases from 1.4172 at the first step to 0.0330 at step 4, 0.0108 at step 8, and 0.0067 at step 64, with most of the decrease occurring within the first few steps. Accuracy is already 72.42% after the first step and fluctuates within roughly half a percentage point thereafter. Cross-entropy is reported as an additional diagnostic; it increases overall across the trajectory, which provides no evidence that further refinement improves the probabilistic predictions, but we do not draw further conclusions from it here.



(a) Accuracy. (b) Cross-entropy loss. (c) RMS latent residual.

Figure 6: Deterministic recursive dynamics of RTSM. Macro-averaged classification accuracy, cross-entropy loss, and RMS latent-state residual across 64 recursion steps on 28 UEA datasets.

Table 13: Deterministic trajectory metrics at selected recursion steps, macro-averaged over 28 datasets.

Step	Accuracy (%)	CE loss	RMS residual
1	72.42	2.5983	1.41722
2	72.53	2.8313	0.19541
4	72.15	2.9168	0.03303
8	72.56	2.9377	0.01080
16	72.61	2.9470	0.00716
32	72.83	2.9476	0.00684
64	72.84	2.9488	0.00673

B.4.4 Reachable headroom and trajectory selection

Headroom is oracle pass@B accuracy minus deterministic accuracy at the same depth. Table 14 reports it at the maximum evaluated budget alongside the three label-free selectors, and Table 15 reports its distribution across datasets. The two tables serve different purposes: the macro table places the oracle bound and all selectors on a common footing, whereas the distribution shows that the macro average is not representative of a typical dataset.

The Best-Q column is particularly relevant to PTRM, which selects among stochastic trajectories using the model's existing halting head. It therefore tests whether the same type of Q-based selection can exploit the stochastic headroom available under RTSM's learned dynamics.

Table 14: Macro-averaged accuracy (%) and headroom (pp) at the maximum evaluated budget, $D = 64$ and $B = 512$. Oracle pass@B uses ground-truth labels and is an upper bound; Best-Q, majority, and minimum-residual selection are label-free.

	Det.	Oracle	Headroom	Best-Q	Majority	Min-res.
RTSM	72.84	73.36	+0.52	72.66	72.76	72.74

Table 15: Distribution of per-dataset headroom (oracle pass@B minus deterministic accuracy) at $D = 64$, $B = 512$, over 28 UEA datasets.

Statistic	Headroom (pp)
Mean	+0.52
Median	0.00
Q25	0.00
Q75	+0.37
IQR	0.37
Minimum	0.00
Maximum	+5.40
Datasets > 0	9 / 28
Datasets $= 0$	19 / 28
Datasets < 0	0 / 28

Table 16 gives the per-dataset breakdown. The positive headroom is concentrated in PhonemeSpectra and EthanolConcentration, whose deterministic accuracies are among the lowest across the 28 evaluated datasets; all other positive gains are below one percentage point.

Table 16: Per-dataset results at $D = 64$, $B = 512$, sorted by headroom. All values are classification accuracy (%) except headroom, reported in percentage points.

Dataset	Det.	Oracle	Headroom	Best-Q	Majority	Min-res.
PS	24.16	29.56	+5.40	23.08	23.89	23.53
EC	29.66	34.98	+5.32	25.86	28.52	28.52
FD	67.22	68.22	+0.99	66.97	66.91	67.00
EP	94.93	95.65	+0.72	95.65	94.93	94.93
NATO	95.56	96.11	+0.56	95.56	95.56	95.56
LSST	65.37	65.90	+0.53	65.41	65.45	65.41
PD	98.74	99.14	+0.40	98.80	98.68	98.60
SAD	98.36	98.73	+0.36	98.68	98.36	98.14
CT	99.30	99.51	+0.21	99.03	99.16	99.16
AF	33.33	33.33	0.00	33.33	33.33	33.33
AWR	98.67	98.67	0.00	98.33	98.33	98.33
BM	100.00	100.00	0.00	100.00	100.00	100.00
CR	97.22	97.22	0.00	97.22	97.22	97.22
DDG	62.00	62.00	0.00	62.00	62.00	62.00
ER	88.52	88.52	0.00	88.52	88.52	88.52
FM	55.00	55.00	0.00	55.00	55.00	55.00
HB	68.78	68.78	0.00	68.78	68.78	68.78
HMD	58.11	58.11	0.00	58.11	58.11	58.11
HW	47.41	47.41	0.00	47.41	47.41	47.41
JV	98.11	98.11	0.00	97.84	98.11	98.11
LIB	87.22	87.22	0.00	87.22	87.22	87.22
MI	54.00	54.00	0.00	54.00	54.00	54.00
PEMS	87.86	87.86	0.00	87.86	87.86	87.86
RS	84.87	84.87	0.00	84.87	84.87	84.87
SRS1	80.89	80.89	0.00	80.89	80.89	80.89
SRS2	51.67	51.67	0.00	51.67	51.67	51.67
SWJ	26.67	26.67	0.00	26.67	26.67	26.67
UW	85.94	85.94	0.00	85.62	85.94	85.94
Mean	**72.84**	**73.36**	**+0.52**	**72.66**	**72.76**	**72.74**

B.4.5 Scope and relation to prior inference-time scaling

EqR applies RI and NI during training to shape the learned dynamics, whereas the RTSM checkpoints evaluated here were trained without those interventions (Huang et al., 2026); whether such training would induce similar scaling in time-series classification remains untested. The setting is closer to PTRM, which perturbs only at inference (Sghaier et al., 2026), but is not an exact reproduction: our probe randomizes the initial state in addition to injecting noise along the trajectory.

For these checkpoints and perturbations, greater breadth adds little oracle coverage at the largest depth, and the tested selectors do not improve accuracy. These findings concern the evaluated probe. Follow-ups include EqR-style training and NI-only inference matching PTRM's intervention.

C RTSM IMPLEMENTATION DETAILS

C.1 TOKENIZER COMPUTATIONS

The tokenizer combines channel projection, gated residual temporal filtering, densely connected dilated convolutions, and a final patch projection. A learned affine map first gives $U_0 = \phi_C(\tilde{X}^\top) \in \mathbb{R}^{T\times d}$. For gated block j, the updates are

$$\begin{aligned}(g_j, v_j) &= \mathrm{split}\big(\phi_j^{\mathrm{in}}(\mathrm{LN}(U_{j-1}))\big)\,,\\ U_j &= U_{j-1} + \phi_j^{\mathrm{out}}\big(\mathrm{SiLU}(g_j)\odot \mathrm{DWConv}_{k_g}(v_j)\big)\,.\end{aligned} \tag{9}$$

The input projection maps d features to $2d$ features, which are split into the two branches; the output projection maps back to d features. The depthwise convolution filters each feature along time, while the pointwise projections mix features.

Let $V_0 = U_J^\top \in \mathbb{R}^{d\times T}$. Dilated branch i computes

$$V_i = \mathrm{GELU}(\mathrm{Conv}_{k_{\mathrm{dil}},\delta_i}([V_0;\ldots;V_{i-1}]))\,, \qquad i = 1,\ldots,I, \tag{10}$$

where concatenation is along the feature dimension and the full convolution returns d features. Each branch therefore combines the preceding feature maps with temporal context at dilation δ_i. These convolutions preserve temporal length. The map $V = V_I$ is passed to the unpadded strided projection in Eq. 3. The reference tokenizer uses $J = 2$ gated blocks with kernel size $k_g = 5$ and $I = 2$ densely connected dilated branches with kernel size $k_{\mathrm{dil}} = 3$ and dilations $\delta_1 = 1$ and $\delta_2 = 2$.

C.2 TRAINING SETTINGS

The benchmark configuration uses AdamW with learning rate 10^{-4}, $(\beta_1, \beta_2) = (0.9, 0.95)$, and weight decay 1. The learning rate is constant, with no warm-up, and dropout is zero. The nominal training budget is 100 epochs, with batch size selected per dataset. Protocol-specific early stopping and checkpoint selection are described in Appendix A; trajectory accounting is specified in Appendix A.1.

The classification and halting heads each use four pooling channels. The halt projection is initialized with $w_q = 0$ and $b_q = -5$, so the initial halt logit is negative. An exponential moving average with decay 0.999 tracks the trainable parameters and supplies the evaluation weights. The fixed random initial latent vector is not trained. During training, a positive halt logit can end a trajectory before the maximum $S_{\max} = 16$ supervision steps. With probability $p_{\exp} = 0.1$, a minimum depth is drawn uniformly from $\{2,\ldots,S_{\max}\}$ and early halting is suppressed until that depth is reached. This exploration rule affects training only; benchmark inference always uses 16 steps.

C.3 Component Comparison

Table 17 records the relationship between RTSM and its recursive-model predecessors. The main architectural changes concern the input representation, state formulation, and sequence-level readout.

Table 17: Component-level comparison of HRM, TRM, and RTSM using the framework of Section 2.2. The comparison follows the released HRM and TRM implementations. HRM provides architectural context; the HRM-style ablation in Section 4.2 changes state structure and network sharing within the TRM training recipe. $H/L/G$ denotes cycles, inner auxiliary-state updates, and gradient-tracked cycles.

Component	**HRM** (Wang et al., 2025a)	**TRM** (Jolicoeur-Martineau, 2025)	**RTSM (ours)**
State initialization	frozen random vectors	frozen random vectors	frozen random vector (y only)
Update networks	separate f_L, f_H	one shared network f	one shared network R_θ
Gradient schedule	one-step gradient: final f_L and f_H updates only	full backpropagation through the final cycle ($G = 1$)	full backpropagation through the final cycle ($G = 1$)
Deep supervision	up to $S_{\max} = 16$ steps, states detached between steps	up to $S_{\max} = 16$ steps, states detached between steps	up to $S_{\max} = 16$ steps, states detached between steps
Halting, training	Q-learning halt/continue head with exploration	correctness-based halt score q with BCE and exploration	same as TRM, using sequence-level correctness
Halting, evaluation	fixed depth S_{eval}	fixed depth S_{eval}	fixed depth S_{eval}
Loss	Stablemax cross-entropy + Q-learning loss	Stablemax cross-entropy + λ_q halting BCE	Stablemax cross-entropy + λ_q halting BCE
Parameter EMA	no	yes	yes
Latent states	z_H, z_L	y, z	y
Reference cycles	$H/L = 2/2$	$H/L = 3/6$	$H/L = 3/0$
Input injection	X enters the low-level update; $z_H \leftarrow f_H(z_L + z_H)$	X enters the z-update; $y \leftarrow f(y + z)$	$y \leftarrow R_\theta(y + X)$ in every cycle
Encoder f_I	discrete token + puzzle embeddings; RoPE	discrete token + puzzle embeddings; RoPE (attention variant)	convolutional time-series tokenizer, N patch tokens; no positional encoding
Output head f_O	per-token linear head	per-token linear head	pooled sequence-level class and halt heads (Eq. 5)
Training loop	persistent ACT carry with halted-sample replacement	persistent ACT carry with halted-sample replacement	per-sample trajectory accounting (Section 3.4)