SEA-NET: A SMALL AND INHERENTLY INTERPRETABLE NEURAL NETWORK FOR TIME SERIES CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Traditional time-series classification methods often operate as black boxes, providing a class prediction without indicating which time point or time interval within the series led to that prediction. To improve temporal interpretability, MILLET (Multiple Instance Learning with Temporal Explanation), which combines an InceptionTime encoder with Conjunctive Pooling, provides both classification and temporal explanations. However, MILLET contains approximately 424 K parameters, making it unsuitable for deployment on resource-constrained micro-controllers, and it achieves an overall time-series accuracy of 0.888 and an explanation score of 0.681 NDCG@n. We therefore propose SEA-Net, a lightweight architecture using small multi-scale depthwise-separable convolutional TSC encoders and improved Multiple Instance Learning (MIL) pooling heads. Our SEA-Net bottleneck encoder with Top-k pooling achieves 0.931 accuracy and 0.765 NDCG@n with only 41 K parameters. Another variant, using a gated encoder with Top-k pooling, achieves 0.955 accuracy and 0.750 NDCG@n with 62 K parameters. These results demonstrate that lightweight architectures can maintain strong classification performance and temporal explanations while substantially reducing model size. The proposed variants address specific limitations of the pooling strategy used by MILLET, providing a practical direction for interpretable time-series classification on resource-constrained edge devices.

1 INTRODUCTION

Time Series Classification (TSC) assigns a label $y \in \{1, \ldots, C\}$ to a univariate time series $X = (x_1, x_2, \ldots, x_t)$, for example distinguishing a normal from an abnormal heartbeat recording, or a healthy from a faulty machine. Deep learning (DL) models solve this task well, and convolutional architectures in particular are its standard baselines. Accuracy alone is not enough in the applications that motivate this work, which impose three requirements *at the same time*. The first is *accurate classification*: the model must remain competitive with the strong convolutional baselines used in the TSC literature. The second is *interpretable predictions*: the model must show *which time steps* drove its decision, and this explanation must be part of the computation that produces the prediction itself, not a post-hoc approximation produced afterward by a separate tool. The third is *a small parameter budget*: the model must be small enough to be a realistic candidate for deployment on an edge device or a micro-controller unit (MCU), where memory is measured in hundreds of kilobytes.

We build on MILLET (Early et al., 2024; Dietterich et al., 1997), which performs TSC as a *Multiple Instance Learning* problem. In MIL a label is attached to a bag (Ilse et al., 2018) of *instances*, not to each instance. The whole time series and each time step are treated as the bag and the instances respectively, and only the bag-level label y is observed. This is exactly the structure of time series classification: the series as a whole is labeled, but only a few time steps carry the evidence; the abnormal beat in a long recording, the spike in a traffic trace. Treating the series as the bag and each time step as an instance turns interpretability into a MIL problem, recovering instance-level contributions from bag-level labels and so yielding interpretability alongside classification. MILLET provides both the time-series classification and a faithful built-in explanation.

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.

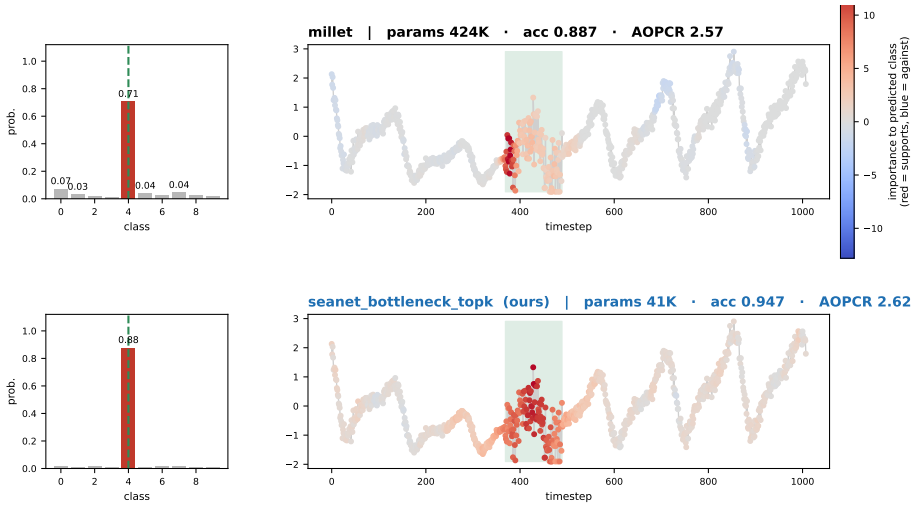


Figure 1: The same WebTraffic series classified by three models. Each panel shows the predicted class and, its right side provides the explanation or interpretation of the prediction, given by the model. The red marks provide the evidence, supporting the predicted class at each time step. The "green shaded window" is the original ground truth of the time steps and was never used during training. MILLET (top) predicts the same class with 71 % confidence and does explain itself, but needs 424 K parameters. SEA-Net (bottom) is more confident, produces a sharper explanation, and uses 41 K parameters.

But its *InceptionTime* (Ismail Fawaz et al., 2020) backbone needs about 423,707 parameters and 848 MFLOPs for one series of length 1008, so at 32-bit precision the weights alone take about 1.6 MB, which already exceeds the memory budget above, before activations, runtime or application code. Swapping MILLET's interpretable head for ordinary global average pooling on the same backbone removes only 1,041 of those parameters (Appendix C.1): the ability to explain itself costs a quarter of a percent of the model, and almost all of the cost is the encoder. This paper builds on that observation. Edge-oriented models fit micro-controllers (Banbury et al., 2021a; Jacob et al., 2018) but predict only a class label; to our knowledge no model meets all three requirements. Hence, we propose our ***S**elective **E**vidence **A**ggregation **Net**work (SEA-Net)*, an inherently interpretable architecture for time series classification. Both parts already exist — separable convolutions (Chollet, 2017; Howard et al., 2017), squeeze-and-expand bottlenecks (He et al., 2016; Iandola et al., 2016; Sandler et al., 2018) and k-max selection in MIL (Durand et al., 2016), and we claim none as new; what is new is their combination inside an interpretable MIL classifier.Our model has 41,324 parameters and 76.7 MFLOPs, $10.3\times$ and $11.1\times$ below MILLET. On WebTraffic, the one dataset whose ground truth marks the time steps carrying the class label, it beats MILLET on both jobs: accuracy 0.931 against 0.888 and NDCG@n 0.765 against 0.681 (Figure 1). Across the 85 UCR datasets, our method stays within two accuracy points of MILLET (Section 4.1).[1] The following are our key contributions: **i) A MIL encoder small enough to be a candidate for small devices** (Section 3.2): multi-scale depthwise separable dilated convolutions with a bottlenecked pointwise mix, keeping the length unchanged so that the per-time-step interpretation is not lost. **ii) Per-class Top-k conjunctive pooling** (Section 3.3), which adds no parameters and becomes *exactly* the baseline head at $\kappa = 1$ (Property 2). This property is what makes the κ study of Section 4.1 a controlled experiment rather than a comparison of two different models. **iii) A warning about the AOPCR measure** (Section 4.1): AOPCR has no fixed scale, and with the architecture and the code held fixed it changes by $4.9\times$ when only the training budget changes, so we compare only against baselines we retrained ourselves.

2 Related Work

Deep learning for time series classification: In Wang et al. (2017), the "Fully Convolutional Network (FCN)" and the "Residual Network (ResNet)" are established as strong baselines for TSC. The InceptionTime (Ismail Fawaz et al., 2020) architecture extends this baseline by applying filters

[1] We reach 0.810 on the 85 UCR datasets against 0.827 for our re-run of MILLET using the authors' released model and code, whose published figure is 0.845; the authors did not reply to our query about the gap.

of several kernel sizes in parallel, capturing patterns of different durations. Temporal Convolutional Networks (TCNs) (Bai et al., 2018) have demonstrated their effectiveness for time-series analysis, and we take inspiration from them for our proposed model. TCNs use *dilated convolutions*, which enable the network to capture information over a wider temporal context. This enlarges the receptive field and allows the model to capture both short-term and long-term temporal dependencies without significantly increasing the number of model parameters. The comprehensive review by Ismail Fawaz et al. (2019) shows that convolutional architectures are an effective family of approaches for TSC. Despite their architectural differences, these models typically conclude with a *Global Average Pooling (GAP)* layer, which removes the explicit temporal resolution of the learned representations.

Interpretability and multiple instance learning: Interpretability methods can be broadly divided into post-hoc approaches and models that incorporate explanations directly into the prediction process. Post-hoc attribution methods, such as LIME (Ribeiro et al., 2016), SHAP (Lundberg & Lee, 2017), and Integrated Gradients (Sundararajan et al., 2017), explain a trained model after prediction, whereas Rudin (2019) advocates incorporating interpretability directly into the model architecture. For time series, two main directions follow this principle. The first direction uses *shapelets*, i.e., short representative subsequences whose presence can explain a prediction. VQShape (Wen et al., 2024) learns a codebook of representative shapes, while InterpGN (Wen et al., 2025) combines a shapelet expert with a deep expert through a confidence gate. The second direction, which we follow, is based on MIL (Dietterich et al., 1997). Attention-based MIL (Ilse et al., 2018) introduced learnable instance-level pooling, later extended by models such as DSMIL (Li et al., 2021) for histopathology. MILLET (Early et al., 2024) adapted this idea to TSC.

Small architectures and deployment budgets: Depthwise separable convolutions (Chollet, 2017; Howard et al., 2017) and squeeze-and-expand bottlenecks (He et al., 2016; Iandola et al., 2016; Sandler et al., 2018) are standard techniques for reducing the parameter count and computational cost. For time series, ATCN (Baharani & Tabkhi, 2022) combines residual connections with separable convolutions for efficient inference, while the method in Risso et al. (2023) optimizes the dilation and channel structure of TCNs under edge-device constraints. However, these approaches focus on efficiency rather than interpretability: they produce a prediction without explicitly indicating which temporal regions contributed to it.

For deployment on resource-constrained devices, MCUNet (Lin et al., 2020), MicroNets (Banbury et al., 2021b), and MLPerf Tiny (Banbury et al., 2021a) provide benchmarks and design principles based on constraints such as flash memory, SRAM, and computational cost. These works demonstrate how neural networks can be made suitable for micro-controller-class hardware, but they do not address the additional challenge of maintaining interpretability under such resource constraints.

3 METHOD

3.1 PROBLEM SETUP AND NOTATION

We propose the ***S**elective **E**vidence **A**ggregation **Net**work* (SEA-Net), a MIL model for TSC that keeps MILLET's two-part structure but replaces its *InceptionTime* backbone with a far smaller encoder (Section 3.2) and its `Conjunctive` head with three selective pooling variants (Section 3.3). SEA-Net has two components: an encoder $\psi_{enc-embed}$, which produces one embedding per time point, and a MIL pooling head ψ_{pool}, which turns those embeddings into a class prediction. A dataset holds n univariate series $X_i = (x_i^1, \ldots, x_i^t)$, $i \in \{1, \ldots, n\}$, of common length t, each carrying a single label $Y_i \in \{1, \ldots, C\}$, where C is the number of classes. We treat each series as a MIL *bag* and each time point as an *instance*, so the encoder maps X_i to $Z_i = [z_i^1, \ldots, z_i^t] \in \mathbb{R}^{t \times d}$, one d-dimensional embedding per instance. The instance labels $\{y_i^1, \ldots, y_i^t\}$ are never annotated in TSC datasets and are never used for training; recovering them from the bag label alone is precisely what MIL techniques are capable of. Unlike classical MIL (Hashimoto et al., 2020; Li et al., 2021), our instances are ordered rather than i.i.d., and every layer of SEA-Net preserves that order.

Whether that per-instance evidence survives depends on ψ_{pool}. Classical *embedded pooling* (Wang et al., 2018) applies global average pooling, $P_i = \frac{1}{t}\sum_{j=1}^{t} z_i^j$, then a classifier $\hat{Y}_i = \psi_{clf}(P_i)$ to classify: the time axis is collapsed *before* classification, so no per-time-point prediction exists and the model cannot report which time points drove the prediction for the series. Our pooling heads (Section 3.3) preserve this order: they score each time point first and aggregate afterward,

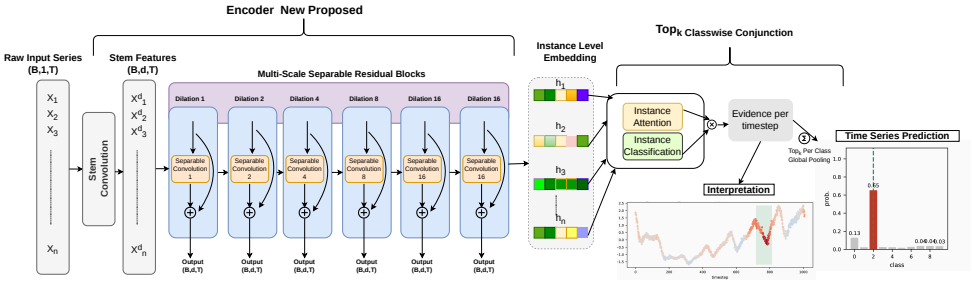


Figure 2: Overview of a complete SEA-Net model, read left to right, with the tensor shape under each stage. **Left, the SEA-Net encoder:** the series $(B, 1, t)$ passes through a stem convolution to (B, d, t) and then through six multi-scale separable residual blocks, whose dilation grows $1, 2, 4, 8$ and is then capped at 16; the curved arrow inside each block is the residual connection, and t is unchanged at every stage. **Middle:** one instance-level embedding $\mathbf{z}_j$ per time step. **Right, the MIL pooling head (Section 3.3):**

which makes the interpretation $\hat{I}_i \in \mathbb{R}^{t\times C}$ available. We keep MILLET's two-part structure but replace *InceptionTime* with a smaller, cheaper encoder, and its `Conjunctive` pooling head with three pooling variants of our own: `Class-wise Conjunctive`, `Soft-Max Conjunctive with Learnable Sharpness` and `Top-k Conjunctive` pooling.

3.2 SEA-Net Encoder

Figure 2 gives an overview of the encoder and its interface with the MIL pooling head. The encoder module begins with a *stem convolution layer*, that extracts local temporal features from the input time series while simultaneously increasing its representation from a single input channel of a time point to d feature channels. The stem convolution is followed by a stack of *multi-scale separable residual blocks* (Section 3.2). The output ($Z_i \in \mathbb{R}^{t\times d}$) of the encoder module is given as the input to the MIL pooling module, which in turn provides the prediction as the posterior probabilities $\hat{Y}_i \in \mathbb{R}^{1\times C}$ of C output classes and the interpretation score of each time step $\hat{I}_i \in \mathbb{R}^{t\times C}$.

Multi-scale separable residual blocks: The encoder body is a stack of six identical blocks built from two established concepts. *Dilated convolutions* (Bai et al., 2018) insert gaps between the filter taps, widening the temporal window a filter sees without adding weights, and *depthwise separable convolutions* (Chollet, 2017; Howard et al., 2017) split a convolution into a depthwise part $D_k^{(\delta)}$, filtering each channel independently along time, and a pointwise 1×1 part P mixing channels at each time point. The smallest element is a *unit*. Three depthwise convolutions read the same input in parallel at kernel sizes $\mathcal{K} = \{5, 11, 23\}$, so short, medium and long patterns are examined at once and no single scale has to be chosen in advance; their outputs are summed and mixed by P:

$$\mathcal{U}(\mathbf{h}) = \mathrm{Drop}\Big(\mathrm{ReLU}\Big(\mathrm{BN}\Big(P\Big(\textstyle\sum_{k\in\mathcal{K}} D_k^{(\delta)}(\mathbf{h})\Big)\Big)\Big)\Big), \qquad \mathcal{K} = \{5, 11, 23\}, \tag{1}$$

where $\mathbf{h} \in \mathbb{R}^{t\times d}$ is the input and δ the dilation. A block is two units in sequence with a residual connection, so it learns a correction to its input rather than replacing one representation with another:

$$\mathrm{Block}(\mathbf{h}) = \mathrm{ReLU}\big(\mathbf{h} + (\mathcal{U}\circ\mathcal{U})(\mathbf{h})\big). \tag{2}$$

Six blocks are stacked, each feeding the next, with the dilation growing geometrically and then held: $\delta_i = \min(2^i, \delta_{\max})$, $\delta_{\max} = 16$. This operation has two properties. First, the receptive field widens quickly but stays bounded: the five taps of the $k = 5$ filter span five consecutive time points at $\delta = 1$, whereas they span up to 65 time points at $\delta = 16$, so each embedding summarizes a local, finite region at several scales. Second, every convolution uses "same" padding with a stride of one, and no pooling or temporal downsampling is applied; the length t is therefore unchanged at every stage, which is what delivers one embedding per time point to the pooling head.

SEA-Net Bottleneck pointwise mixing: In the separable unit, the depthwise filters are cheap because each channel has one small kernel. Most of the parameters are in the pointwise mixing P

of Equation (1), which costs d^2 weights per unit. We therefore factorize P into two smaller 1×1 convolutions: the first squeezes the d channels to a width $r < d$, and the second expands them back:

$$P(\mathbf{h}) \;=\; P_{\text{exp}}\big(P_{\text{sq}}(\mathbf{h})\big), \qquad P_{\text{sq}} : \mathbb{R}^d \to \mathbb{R}^r, \quad P_{\text{exp}} : \mathbb{R}^r \to \mathbb{R}^d, \qquad \frac{2dr}{d^2} = \frac{2r}{d}. \tag{3}$$

Because there is no non-linearity between the two convolutions, the pair is equivalent to a single mixing matrix of rank r. This leaves the depth of the block unchanged and adds no extra layer. With $r = d/4$, the pointwise cost drops from d^2 to $2dr = d^2/2$, a halving at any width. At $d = 64$ this reduces the whole model from 57,580 to 41,324 parameters (-28%). The block still outputs a matrix of shape $\mathbb{R}^{t \times d}$, so the pooling head needs no change.

SEA-Net Channel re-weighting variants: Both variants introduce a channel re-weighting layer after the last *multi-scale separable residual* block. The purpose of this layer is to adaptively adjust the contribution of the d feature channels according to their relevance to the input time series, while leaving the encoder blocks and the MIL pooling head unchanged. In both variants, the output of the sixth block, $Z_i \in \mathbb{R}^{t \times d}$, is first passed through the channel re-weighting layer. Each time-step embedding z_i^j, $j = 1, \ldots, t$, is then re-weighted according to:

$$\tilde{z}_i^j = z_i^j \odot (1 + g_i^j), \qquad \underbrace{g_i^j = \sigma(W_g\, s_i)}_{\text{Gated (mean)}}, \quad s_i = \tfrac{1}{t}\textstyle\sum_j z_i^j \qquad \underbrace{g_i^j = \sigma(W_g * X_i)^j}_{\text{Input Gated}}, \qquad g_i^j \in [0,1]^d. \tag{4}$$

where $g_i^j \in [0,1]^d$ contains one weight for each feature channel. The use of $(1 + g_i^j) \in [1,2]^d$, rather than g_i^j alone, preserves the original features while allowing informative channels to be amplified. This also preserves the differences between time-step embeddings, which are used to compute the interpretation score $\hat{I}_i$. Since the re-weighting is applied only along the channel dimension, the temporal structure of Z_i remains unchanged. Consequently, both the following variants produce an output $\tilde{Z}_i \in \mathbb{R}^{t \times d}$ that can be directly passed to the same MIL pooling head.

SEA-NET GATED (MEAN): In the first variant, a single set of channel weights is computed for the entire time series. The output Z_i of the sixth encoder block is first averaged over its t time points to obtain a series-level representation. This representation is passed through a linear transformation $W_g \in \mathbb{R}^{d \times d}$ followed by a sigmoid function to obtain one weight for each channel: These weights are applied to every time-step embedding as in Equation (4). Thus, this variant determines which feature channels are generally more informative for the entire time series. However, the same weights are applied at every time point and therefore cannot account for changes in the importance of a channel over time. The mean-based version is used in the main experiments (others in Appendix B.3)

SEA-NET INPUT GATED: The second variant performs channel re-weighting at each time point. Instead of computing the weights from the deep representation Z_i, they are obtained directly from the raw input time series X_i. A 1D convolution with kernel size 7 transforms the single input channel into d channels, followed by a sigmoid function to produce a separate set of channel weights $g_i^j \in [0,1]^d$ for each time point: where $W_g * X_i$ denotes the 1D convolution. Unlike the previous variant, the channel weights can therefore vary both across channels and across time points. Because they are computed directly from the raw input, they can respond to local events, such as a sudden peak, before these patterns are potentially smoothed by the encoder's receptive field. These two variants allow us to study the effect of coarse, series-level channel re-weighting versus fine-grained, time-dependent channel re-weighting while keeping the remainder of the architecture unchanged (Appendix B).

SEA-Net Spike/Trend: A stack of convolutions followed by batch normalization is biased toward smooth and sustained patterns, whereas in some datasets the evidence of a class is a single sharp discontinuity (i.e., a spike) in the time series. In this variant, we therefore run a second, high-pass-filter-like branch in parallel with the encoder blocks and merge the two branches before the pooling head. The output of the encoder is called the *trend* features, and the output of this new high-pass-filter-like branch is called the *spike* features. In Figure 2, the encoder path (the stem convolution and the six blocks) is untouched and gives the trend features $Z_i \in \mathbb{R}^{t \times d}$. The new spike branch takes the original time series X_i as input, applies a moving average $\text{avg}_w(X_i)$ with window $w = 5$, and then

subtracts this smoothed signal from the original time series. This subtraction removes the smooth, slowly varying component of the signal, leaving a residual that is close to zero in smooth regions and becomes large around sudden changes or discontinuities. Then a convolution with kernel size 3, followed by BN and ReLU layers, is applied, which turns the output into $d_{\text{spike}} = 16$ channels. Finally, at each time point, the trend features and spike features are concatenated and projected back to d channels using the pointwise convolution P, as defined in Equation (1). The resulting representation is then aggregated using the mean operation described in Equation (4).

$$\text{spike}_i^j = \text{ReLU}\left(\text{BN}\left(\text{Conv}_3(X_i - \text{avg}_w(X_i))\right)\right)^j \in \mathbb{R}^{d_{\text{spike}}} \tag{5}$$

$$\tilde{z}_i^j = P\big([\, z_i^j \,;\, \text{spike}_i^j \,]\big) \odot (1 + g_i) \tag{6}$$

where $[\,\cdot\,;\,\cdot\,]$ is the concatenation along the channel axis, giving $d + d_{\text{spike}}$ channels, and g_i is computed as in Equation (4) on the mixed features. With this branch, the pooling head directly receives features which mark the discontinuities of the series. Hence, we expect an advantage on datasets where the class evidence has impulsive patterns.

3.3 MIL Pooling Head

With the encoder complete, we now describe the second component of SEA-Net, the MIL pooling head ψ_{pool}, shown on the right side of Figure 2. Given the instance-level embeddings $Z_i \in \mathbb{R}^{t\times d} = [z_i^1, \ldots, z_i^t]$, the pooling head has two branches: an attention branch that determines the importance of each time point, and a classifier branch that determines the class evidence provided by each time point. Their product defines the evidence $e_i^{j,k}$ of time point j for class k. Aggregating this evidence over time produces the class prediction $\hat{Y}_i \in \mathbb{R}^{1\times C}$, while retaining the evidence at each time point gives the interpretation score $\hat{I}_i \in \mathbb{R}^{t\times C}$. Our pooling head is based on the `Conjunctive` pooling of MILLET (Equation (7)) (Early et al., 2024), which uses a single attention weight a_i^j for each time point and a class-wise classifier output $\hat{y}_i^j$:

$$\begin{aligned} a_i^j &= \sigma\big(\mathbf{w}_a^\top \tanh(W_a z_i^j)\big) \in [0,1] \\ \hat{y}_i^j &= W_c z_i^j + \mathbf{b}_c \in \mathbb{R}^C \\ \hat{Y}_i{}^k &= \tfrac{1}{t}\textstyle\sum_{j=1}^t a_i^j \hat{y}_i^{j,k}, \quad \hat{I}_i^{j,k} = a_i^j \hat{y}_i^{j,k} \end{aligned} \tag{7}$$

$$\begin{aligned} a_i^j &= \sigma\big(W_2 \tanh(W_1 z_i^j)\big) \in [0,1]^C \\ e_i^{j,k} &= a_i^{j,k} \hat{y}_i^{j,k}, \quad \hat{I}_i^{j,k} = e_i^{j,k} \\ \hat{Y}_i{}^k &= \tfrac{1}{t}\textstyle\sum_{j=1}^t e_i^{j,k} \end{aligned} \tag{8}$$

where $W_a \in \mathbb{R}^{d_a\times d}$ and $\mathbf{w}_a \in \mathbb{R}^{d_a}$ define the attention branch, while $W_c \in \mathbb{R}^{C\times d}$ and $\mathbf{b}_c \in \mathbb{R}^C$ define the classifier branch; $W_1 \in \mathbb{R}^{d_a\times d}$ and $W_2 \in \mathbb{R}^{C\times d_a}$ are its class-wise counterparts below.

Conjunctive pooling has three limitations that motivate our pooling variants. First, the scalar attention weight a_i^j is shared across all C classes and therefore cannot represent class-specific importance. Second, the sigmoid weights are not normalized over time: although $a_i^j \in [0,1]$, their sum depends on the series length t, while the aggregation divides by t. Consequently, the magnitude of $\hat{Y}_i{}^k$ can depend on the series length, which is particularly relevant for the UCR archive, where series lengths range from 15 to 2,844. Third, averaging all t evidence values can dilute a small number of highly informative time points when most of the series contains background information. We developed seven pooling variants; here we describe the three that directly address the limitations above, while the remaining four are presented in Appendix C. All three variants retain the two-branch structure and the classifier output $\hat{y}_i^j$ of Equation (7). They differ only in how the attention weights are computed and/or how the resulting evidence is aggregated. In every case, the interpretation $\hat{I}_i$ is obtained from the same evidence used to compute $\hat{Y}_i$, ensuring consistency between prediction and interpretation.

Class-wise / Soft-Max / Top-k conjunctive pooling: The `Class-wise` variant (Equation (8)) addresses the first limitation by replacing the single scalar attention weight with a class-specific vector: Each time point can therefore have a different importance for different classes. The modification only widens the final attention layer, adding $(C-1)(d_a+1)$ parameters compared with the original attention branch. If $a_i^{j,k} = a_i^j$ for all k, the formulation reduces exactly to the original `Conjunctive` pooling. The `Soft-Max` variant addresses the second limitation by normalizing the attention weights across

time. For each class, the attention scores are converted into a probability distribution using a Soft-Max with learnable temperature $\tau > 0$ (Equation (9)):

$$
\begin{aligned}
s_i^{j,k} &= \left(W_2 \tanh(W_1 z_i^j)\right)_k \\
\alpha_i^{j,k} &= \frac{\exp(s_i^{j,k}/\tau)}{\sum_{l=1}^{t} \exp(s_i^{l,k}/\tau)} \\
\hat{Y}_i^{\,k} &= \textstyle\sum_{j=1}^{t} \alpha_i^{j,k} \hat{y}_i^{j,k}, \quad \hat{I}_i^{j,k} = \alpha_i^{j,k} \hat{y}_i^{j,k}
\end{aligned}
\tag{9}
$$

$$
\begin{aligned}
m &= \max\left(1, \lceil \kappa t \rceil\right) \\
\mathcal{S}_i^k &= \operatorname{top}_m\left(e_i^{1,k}, \ldots, e_i^{t,k}\right) \\
\hat{Y}_i^{\,k} &= \tfrac{1}{m} \textstyle\sum_{j \in \mathcal{S}_i^k} e_i^{j,k}, \quad \hat{I}_i^{j,k} = e_i^{j,k}
\end{aligned}
\tag{10}
$$

Because $\sum_j \alpha_i^{j,k} = 1$, the prediction is no longer directly affected by the series length, making series of different lengths more comparable. The temperature controls the concentration of the attention: large τ produces a more uniform distribution approaching $1/t$, whereas small τ concentrates the weight on the highest-scoring time points. We parameterize $\tau = \exp(\rho)$ with learnable ρ, ensuring $\tau > 0$ while introducing only one additional parameter.

The `Top-`k `conjunctive` variant addresses the third limitation by aggregating only the most informative time points rather than averaging the evidence over the entire series. It uses the class-wise attention of Equation (8) and retains only the m largest evidence values for each class (Equation (10)): Here, $\kappa \in [0, 1]$ controls the fraction of time points retained and introduces no additional learnable parameters. When $\kappa = 1$, this formulation reduces to the class-wise pooling of Equation (8); for $\kappa < 1$, the aggregation focuses on the most informative time points, reducing the influence of background regions. Consequently, the prediction is based on an explicit subset of time points, which also makes the resulting interpretation more focused. This is the pooling head used in our best-performing models.

Every encoder and head above shares the same input and output shapes, so any pair combines from configuration alone. Section 4 evaluates these combinations, taking the bottleneck encoder with Top-k pooling as our main model.

4 Experimental Setup and Results

Research questions and datasets: We ask whether the small model retains classification accuracy, retains a useful interpretation, and how much cost it actually removes; two further questions, **RQ4** what the bottleneck contributes and **RQ5** what Top-k pooling contributes, are answered by the ablation of Section 4.1. We answer them on two datasets (Table 15).

We use the same WebTraffic dataset as MILLET (Early et al., 2024): a synthetic set of 1,000 series of length 1,008 over 10 classes, split 500/500, in which each class signature is injected inside a known window (Figure 14). It is the only dataset with a ground truth per time point, hence the interpretation can be scored rather than merely inspected. For wider coverage we use the UCR archive (Dau et al., 2019), reporting the 85-dataset subset used by MILLET so that the numbers are comparable; we also report the performance on the full 128-dataset archive in Appendix E.5. Class signatures are not annotated there, so on UCR the interpretation can only be evaluated with AOPCR.

We use the test accuracy for the classification, and *AOPCR* (Samek et al., 2017) together with *NDCG@*n (Järvelin & Kekäläinen, 2002) for the interpretation; both metrics are defined in Appendix D.4. *AOPCR* progressively removes time points according to their estimated importance and accumulates the resulting decrease in the score of the predicted class. Computing *AOPCR* does not require ground truth, so we can report this metric on both datasets. NDCG@n ranks the time points by the importance the model assigns them and compares that ranking with the ground truth.

4.1 Classification and Interpretation Results

WebTraffic: Table 1 gives one complete model per row, written as *encoder* + *pooling head*. Our SEA-Net model exceeds MILLET's accuracy and achieves a better NDCG@n, while being about ten times smaller: **0.931** $\pm$ 0.032 accuracy and **0.765** $\pm$ 0.013 NDCG@n with 41,324 parameters, against 0.888 ± 0.007 and 0.681 ± 0.013 with 423,707 parameters for MILLET under the same budget (Table 1). The gated (mean) + Top-k variant is more accurate and far more stable (0.955 ± 0.002),

Table 1: Classification and explanation quality on WebTraffic, each model written as *encoder + pooling head*. Rows marked * are mean ± s.d. over the *Runs* column. Higher is better except for NLL. AOPCR is unnormalised, so it compares only models trained under the same recipe; rows marked † use MILLET's recipe (1500 epochs) and are never marked best. Run conventions in Appendix D.3, parameter counts in Table 22.

Model (encoder + pooling head)	Runs	Accuracy	AUROC	NLL	AOPCR	NDCG@n
SEA-Net (ours): encoder + Top-k conjunctive head						
gated (mean) + Top-k conj.*	3	$\mathbf{0.955 \pm 0.002}$	$\mathbf{0.998 \pm 0.000}$	$\mathbf{0.254 \pm 0.010}$	2.225 ± 0.107	0.750 ± 0.018
bottleneck + Top-k conj. ($\lambda_{\text{focus}} = 0$)	1	0.950	**0.998**	0.263	2.303	0.765
spike/trend + Top-k conj.	1	0.950	0.997	0.288	2.316	0.756
input-gated + Top-k conj.	1	0.946	0.997	0.261	2.803	0.748
bottleneck + Top-k conj.	1	0.942	0.996	0.294	2.951	**0.786**
bottleneck + Top-k conj.*	3	0.931 ± 0.032	0.996 ± 0.002	0.312 ± 0.063	2.498 ± 0.149	0.765 ± 0.013
Baselines re-trained here under the same recipe						
MILLET (InceptionTime + conjunctive)*	4	0.888 ± 0.007	0.988 ± 0.002	0.513 ± 0.019	2.457 ± 0.655	0.681 ± 0.013
ResNet + conjunctive	1	0.772	0.957	0.824	2.952	0.554
FCN + conjunctive	1	0.742	0.955	0.910	**3.826**	0.533
MILLET under its own recipe						
MILLET, re-trained here with their recipe*	2	0.918 ± 0.002	0.994 ± 0.001	0.392 ± 0.021	$12.060 \pm 1.208^{\dagger}$	0.643 ± 0.019
MILLET, published (mean of 5 seeds)	5	0.924	0.994	0.379	$12.768^{\dagger}$	0.674
MILLET, published 5-model ensemble	1	0.940	0.996	0.277	$11.100^{\dagger}$	0.694

but needs 61,740 parameters and a lower NDCG@n (0.750 ± 0.018), so the bottleneck model stays our headline. Two further columns point the same way. Our negative log-likelihood is 0.312 ± 0.063 against 0.513 ± 0.019 for the matched-budget baseline, a 39 % reduction, and it stays below MILLET's own 1,500-epoch recipe (0.392 ± 0.021) and its published figure (0.379); AUROC moves the same way, 0.996 ± 0.002 against 0.988 ± 0.002. The 41 K model is therefore better calibrated than the baseline even when the baseline is given nearly four times the training budget. **UCR archive:** On UCR, the comparison depends on the size budget. Our 41 K model reaches 0.810 mean accuracy, 1.8 points below the matched-budget MILLET baseline (0.827) and 3.5 points below MILLET's published accuracy (0.845), at a tenth of the parameters and with mean rank 15.40 against 12.60. Counted per dataset against MILLET's published accuracies, it wins on 20, ties on 15 and loses on 50 of the 85, where the ten-times-larger matched-budget baseline scores 13/25/46: more outright wins than the baseline, but also more losses (Table 29). The gap comes from size, not from design. Re-running the *same* architecture, with the same blocks and head, in a *wide* configuration (d from 64 to 128, blocks from 4 to 6) reaches 0.829 accuracy and mean rank 11.53, the best rank in Table 29: ahead of the matched-budget MILLET baseline on accuracy (0.827), on rank (12.60) and on wins (26 against 13), with 36 % fewer parameters. At a comparable size and the same training budget, replacing the MILLET backbone by ours is worth about 1.1 rank places. The SEA-Net design is therefore not what loses accuracy on UCR; the smaller parameter budget is.

RQ4, the bottleneck: Table 21, one seed per row, isolates the encoder: the two rows of each pair share the same pooling head and differ only in whether the bottleneck is used. Replacing the plain encoder by the bottleneck encoder removes 16,256 parameters, that is 28 % of the 57,580 of the plain model. Accuracy then changes by -0.014 with the class-wise head and by -0.001 with the Top-k head, and NDCG@n by $+0.014$ and -0.013 respectively. The accuracy changes are well inside the three-run spread of ± 0.032 and the NDCG@n changes sit at its ± 0.013 spread (Table 1), so none can be resolved from these single-seed runs. We therefore claim only that the bottleneck delivers the size reduction *at no measurable cost in accuracy or interpretation*.

RQ5, Top-k pooling: Read the other way, the same table isolates the head: the two rows of each pair share the same encoder and differ only in the pooling head. Replacing the class-wise head by Top-k raises NDCG@n from 0.719 to 0.778 on the plain encoder ($+0.059$) and from 0.733 to 0.765 on the bottleneck encoder ($+0.032$), two to five times the ± 0.013 seed spread. This gain in interpretation ranking is the claim we make for the head. Accuracy moves the same way, by $+0.016$ and $+0.029$, but both sit inside the ± 0.032 accuracy spread, so we do not present Top-k as an accuracy gain.

Table 26 varies only the selection fraction κ, with the architecture and the training recipe unchanged. By Property 2, the $\kappa = 1$ row *is* the class-wise baseline head, so the difference between its NDCG@n of 0.722 and the 0.755 reached at $\kappa = 0.10$ comes from the selection alone and not from a change of model. Two caveats follow. First, κ has a best value rather than a monotone trend: $\kappa = 0.05$ is worse than $\kappa = 0.10$ on AOPCR and NDCG@n and no better on accuracy, so keeping too few time

Table 2: Performance on the 84 UCR datasets shared by every row, as *mean / average rank / number of wins*. Higher is better except for NLL. Our models and the re-trained baselines share one training recipe; the published MILLET rows use their own (1500 epochs) and are ranked with the rest. Ranking rules, the full protocol and parameter counts are in Appendix D.3 and Table 22.

Model	Accuracy	Bal. accuracy	AUROC	NLL
SEA-Net (ours), our recipe				
gated (mean) + Top-k conj.	0.806 / 9.17 / 11	0.778 / 9.38 / 11	0.916 / 9.39 / 11	0.561 / 7.28 / 3
base (wide, $d = 128$) + (MILLET)	0.822 / 7.75 / 13	0.797 / 7.64 / 13	0.927 / 7.60 / 15	0.522 / 6.35 / 7
gated (max) + Top-k conj.	0.809 / 8.73 / 11	0.783 / 8.61 / 10	0.916 / 8.38 / 13	0.551 / 6.92 / 6
spike/trend + Top-k conj.	0.807 / 8.44 / 9	0.781 / 8.32 / 9	0.913 / 8.57 / 12	0.551 / 7.06 / 6
reconstruction-residual + adaptive class-wise	0.812 / 8.96 / 11	0.784 / 8.84 / 10	0.922 / 8.74 / 12	0.555 / 7.95 / 4
base (wide, $d = 128$) + class-wise	**0.827 / 7.32 / 16**	0.802 / 7.10 / 15	0.925 / 7.96 / 16	**0.516 / 6.60 / 8**
input-gated + Top-k conj.	0.811 / 8.29 / 12	0.786 / 8.27 / 11	0.922 / 8.02 / 16	0.541 / 6.29 / 4
bottleneck + Top-k conj.	0.808 / 9.10 / 7	0.781 / 9.31 / 7	0.917 / 9.47 / 7	0.556 / 7.29 / 2
Baselines re-trained here				
MILLET (InceptionTime + conjunctive), our recipe	0.827 / 8.08 / 5	0.800 / 8.14 / 5	0.927 / 7.96 / 9	0.530 / 7.45 / 4
ResNet + conjunctive	0.813 / 8.23 / 15	0.782 / 8.76 / 12	0.923 / 8.03 / 14	0.575 / 8.44 / 3
FCN + conjunctive	0.812 / 8.92 / 10	0.782 / 9.43 / 11	0.921 / 8.90 / 10	0.578 / 8.73 / 6
MILLET, published (Early et al., 2024)				
Conjunctive FCN, published (mean of 5 seeds)	0.828 / 8.12 / 10	0.803 / 8.02 / 9	0.928 / 7.70 / 13	1.059 / 9.31 / 9
Conjunctive ResNet, published (mean of 5 seeds)	0.834 / 7.29 / 10	0.811 / 7.15 / 9	0.930 / 7.35 / 9	1.186 / 10.79 / 2
Conjunctive InceptionTime, published (mean of 5 seeds)	**0.843 / 5.79 / 12**	**0.822 / 5.45 / 13**	0.932 / 6.44 / 13	1.232 / 10.12 / 5

points hurts as much as keeping too many. Second, AOPCR does not track κ the way NDCG@n does, peaking at $\kappa = 0.50$ (3.114) where NDCG@n is among its lowest (0.732), which is one more reason to read AOPCR with caution (Section 4.1). Table 27 switches off the attention-entropy penalty λ_{focus} (Section 4), a control independent of κ: removing it raises accuracy from 0.888 to 0.950 and NDCG@n from 0.755 to 0.765, while AOPCR falls from 2.409 to 2.303. On this single seed the penalty does not pay for itself. As every cell here is one seed, we read these rows as a pattern rather than as exact values. Re-trained under MILLET's own 1,500-epoch recipe, MILLET scores AOPCR 12.060 ± 1.208 on WebTraffic, against 2.457 ± 0.655 for the same model under our matched budget (Table 1). The architecture, code, metric and data are identical; only the recipe differs. A $4.9\times$ higher AOPCR would normally mean a far better explanation, but it does not here. AOPCR averages the drop in the predicted-class *logit* as the top-ranked time points are removed (Appendix D.4), and logits have no fixed scale: a model trained longer is more confident, so every removal drops the logit further, whether or not its ranking improved. NDCG@n, which scores the ranking itself and is normalized to $[0, 1]$, confirms this: over the same pair it *falls* from 0.681 ± 0.013 to 0.643 ± 0.019, a drop of twice its own spread. AOPCR is therefore comparable only within a fixed training budget, hence not across papers: we retrain every baseline under our own budget rather than copying published numbers, never compare against MILLET's published AOPCR, and base every interpretation claim on NDCG@n.

5 Discussion and Conclusion

Our trade-off depends on where the model runs: saving $10.3\times$ in parameters for 1.8 accuracy points on UCR buys nothing on a workstation, but for the micro-controller class of MCUNet (Lin et al., 2020) and MicroNets (Banbury et al., 2021b) a 41 K model is a candidate where a 424 K one is not, and the per-time-step interpretation is kept. The price is paid for the size and not the design, since widening the same encoder recovers the UCR gap (Section 4.1). Our costs are parameter counts, analytic FLOPs and GPU timings; nothing was quantized or run on a micro-controller (Table 22), so deployment fitness is argued from size rather than shown. Activation memory does not shrink, because the encoder preserves length: peak memory exceeds the baseline's, and where SRAM rather than flash binds, streaming or strided variants are the next step. Interpretation is scored only on WebTraffic, which is also our design set, and its backup metric AOPCR is budget-sensitive (Section 4.1). Seeds are uneven: three models use three seeds, each ablation one, read against those spreads (Section 4). Training is slower, since GPU kernels favor dense over depthwise convolutions; the saving is at inference only. In summary, the cost of MIL lies in the encoder and not in the interpretability: our model keeps the per-time-step explanation and beats the equally-trained MILLET baseline on WebTraffic at a tenth of the parameters. Built-in interpretability is therefore not what makes these models large, and need not be given up to make them small.

Reproducibility Statement

Everything needed to reproduce the results is either in the paper or in the anonymous code release. The models are defined in Section 3 and, for the encoder and pooling variants not used in the main text, in Appendices B and C. The single training setting used for every model and every dataset is listed in Section 4 and repeated with all the remaining hyperparameters in Appendix D.3. The two evaluation metrics are defined in Section 4, and the perturbation procedure behind AOPCR, which Section 4.1 shows depends on the training recipe, is given in Appendix D.4. WebTraffic is produced by the released generator of Early et al. (2024) at the settings in Appendix D.3; the UCR archive (Dau et al., 2019) is public and used with its published train/test splits and no preprocessing other than standardizing each series. Seed counts are given per row in Table 2 and the per-seed values are in Appendix E.4. Cost figures come from the measurement procedure in Section 4; the script that produces them is in the code release. All 72 model combinations are listed row by row in Appendix E.1. Anonymous code, configurations and result tables: `https://anonymous.4open.science/r/seanet-ANON`.

Ethics Statement

This work collects no new data and involves no human subjects. It uses the public UCR archive (Dau et al., 2019) under its original terms and a synthetic benchmark produced by released code; some UCR datasets come from human physiological recordings, but they are distributed in anonymized form by their maintainers and we did not try to re-identify them or add to them. The one risk worth naming is specific to this topic: a model that shows an explanation makes people trust it more than one that does not, and Section 4.1 shows that a common faithfulness score can be raised by training longer rather than by explaining better. Interpretations from models of this kind should therefore be checked on the target domain before they are used to support decisions about people. The code we build on is released under the Apache 2.0 license and its attribution notice is kept in our release.

AI Use Statement

In this work, we used generative AI tools for writing and editing the text of this paper, for literature search assistance while preparing Section 2, and for producing supporting experiment-management and plotting code. We have not used generative AI tools to generate, select, filter or interpret experimental results, to design the models or the experimental protocol, or to produce any of the numbers reported in this paper. We have reviewed all AI-assisted work. Every numeric value in the paper was traced by the authors to the specific result file and column it came from before it was written down, and the tables were checked against those files a second time after the paper was complete; AI-assisted code was read and tested by the authors, and all reported measurements come from runs the authors executed themselves. Every reference was checked against the original publication, and claims of novelty were checked against a manual literature survey rather than against tool output. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

References

Mohammadreza Baharani and Hamed Tabkhi. ATCN: Resource-efficient processing of time series on edge. *ACM Transactions on Embedded Computing Systems*, 21(5):1–21, 2022.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong. MLPerf tiny benchmark. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2021a.

Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers. In *Proceedings of Machine Learning and Systems (MLSys)*, 2021b.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1251–1258, 2017.

Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

Thibaut Durand, Nicolas Thome, and Matthieu Cord. WELDON: Weakly supervised learning of deep convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4743–4752, 2016.

Joseph Early, Gavin K. C. Cheung, Kurt Cutajar, Hanting Xie, Jasmina Kandola, and Niall Twomey. Inherently interpretable time series classification via multiple instance learning. In *International Conference on Learning Representations (ICLR)*, 2024.

Noriaki Hashimoto, Daisuke Fukushima, Ryoichi Koga, Yusuke Takagi, Kaho Ko, Kei Kohno, Masato Nakaguro, Shigeo Nakamura, Hidekata Hontani, and Ichiro Takeuchi. Multi-scale domain-adversarial multiple-instance CNN for cancer subtype classification with unannotated histopathological images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3852–3861, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7132–7141, 2018.

Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

Maximilian Ilse, Jakub M. Tomczak, and Max Welling. Attention-based deep multiple instance learning. In *International Conference on Machine Learning (ICML)*, pp. 2127–2136, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pp. 448–456, 2015.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding AlexNet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, 2020.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2704–2713, 2018.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Bin Li, Yin Li, and Kevin W. Eliceiri. Dual-stream multiple instance learning network for whole slide image classification with self-supervised contrastive learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14318–14328, 2021.

Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. MCUNet: Tiny deep learning on IoT devices. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4765–4774, 2017.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 2016.

Matteo Risso, Alessio Burrello, Francesco Conti, Lorenzo Lamberti, Yukai Chen, Luca Benini, Enrico Macii, Massimo Poncino, and Daniele Jahier Pagliari. Lightweight neural architecture search for temporal convolutional networks at the edge. *IEEE Transactions on Computers*, 72(3): 744–758, 2023.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2660–2673, 2017.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, 2018.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning (ICML)*, pp. 3319–3328, 2017.

Xinggang Wang, Yongluan Yan, Peng Tang, Xiang Bai, and Wenyu Liu. Revisiting multiple instance neural networks. *Pattern Recognition*, 74:15–24, 2018. doi: 10.1016/j.patcog.2017.08.026.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1578–1585, 2017.

Yunshi Wen, Tengfei Ma, Tsui-Wei Weng, Lam M. Nguyen, and Anak Agung Julius. Abstracted shapes as tokens – a generalizable and interpretable model for time-series classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://openreview.net/forum?id=pwKkNSuuEs.

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 *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=n34taxF0TC.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1492–1500, 2017.

Appendix of

SEA-Net: A Small and Inherently Interpretable Neural Network for Time Series Classification

A Why MIL Makes SEA-Net Interpretable

Section 1 and Section 3.1 treat time series classification as a *Multiple Instance Learning* (MIL) problem (Dietterich et al., 1997; Early et al., 2024): the series X_i is the bag, each time point x_i^j is an instance, and only the bag label Y_i is observed. MIL has one main role in SEA-Net. The instance labels y_i^j are never annotated, yet the model must still say which time points carry the evidence for its prediction. MIL recovers this from the bag label alone: the pooling head first gives every time point its own class evidence and only then aggregates it into one prediction, so the same per-time-point evidence that forms the prediction is also the interpretation $\hat{I}_i \in \mathbb{R}^{t \times C}$. The interpretation is therefore part of the forward pass and not a post-hoc approximation.

The main text states this view in words. This appendix adds two details: Figure 3 draws the full pipeline from the bag to the prediction and the interpretation, and Equation (11) writes out the embedded pooling that this view replaces, so that the order of its two steps can be compared with the order of our heads.

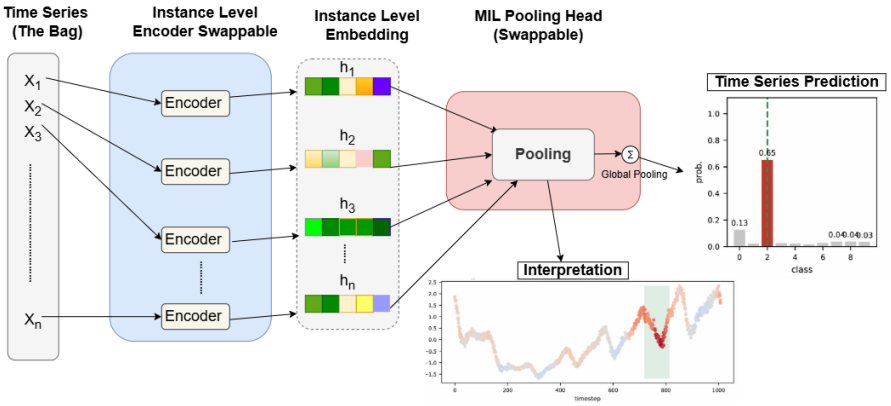


Figure 3: The MIL view of a time series, read from left to right. The series X_i is the bag and each time point is one instance. The encoder gives one embedding z_i^j per instance; the pooling head weights those embeddings and combines them into the prediction $\hat{Y}_i$ (right). The same weighted contributions, kept instead of discarded, are the interpretation $\hat{I}_i$ drawn on the series below, where red pushes towards the predicted class and blue pushes away from it. The two grey boxes are interchangeable, which is what lets any encoder of Appendix B be paired with any head of Appendix C.

The alternative is worth writing down once, because the rest of the design follows from it. *Embedded pooling* (Wang et al., 2018) first averages the embeddings and then classifies the average:

$$P_i = \frac{1}{t}\sum_{j=1}^{t} z_i^j \in \mathbb{R}^d, \qquad \hat{Y}_i = \psi_{clf}(P_i). \tag{11}$$

Once P_i is formed, the time index is gone, so there is no per-time-point prediction left to report. Every head of Appendix C, starting with the conjunctive pooling of MILLET (Equation (7)), runs the two steps in the opposite order: it classifies each time point first and aggregates afterwards, which is why it can report the interpretation.

B How the Encoder Stays Small

Section 3.2 describes the SEA-Net encoder: a stem convolution followed by a stack of multi-scale separable residual blocks (Equation (1) and Equation (2)), which keeps the length t unchanged, so

that one embedding per time point reaches the pooling head, as the MIL view of Appendix A requires. It also gives four variants of this encoder: the bottleneck pointwise mixing (Equation (3)), the two channel re-weighting variants, *Gated (mean)* and *Input Gated* (Equation (4)), and the Spike/Trend encoder (Equation (6)). Every variant returns the same output $Z_i \in \mathbb{R}^{t \times d}$, so any encoder can be paired with any pooling head.

This appendix gives the details that the main text leaves out. Appendix B.1 draws one block and the effect of the dilation, Appendix B.2 works out where the saving of the bottleneck comes from, and Appendix B.3 and Appendix B.4 write out the gated and the Spike/Trend variants step by step. Appendix B.5 to Appendix B.7 then describe one more encoder and two wrappers that the main text does not cover, and Appendix B.8 lists the two main encoders layer by layer.

B.1 Multi-scale Separable Blocks

Figure 4 draws one block of Section 3.2 as a data path, i.e. Equation (1) and Equation (2). Two terms of Equation (1) need a short explanation. In a depthwise convolution $D_k^{(\delta)}$, each input channel is filtered with its own kernel, and the channels are not mixed. It is a grouped convolution with as many groups as channels, so it needs d times fewer parameters and FLOPs than a full convolution; the same idea is the basis of efficient architectures such as those of Howard et al. (2017) and Xie et al. (2017). The pointwise 1×1 convolution P is therefore the only place where the channels are mixed, and it also holds most of the parameters; this is why the bottleneck of Equation (3) factorizes P and nothing else. Batch normalization (Ioffe & Szegedy, 2015) stabilizes the activations and dropout regularizes the model; neither of them changes the shape.

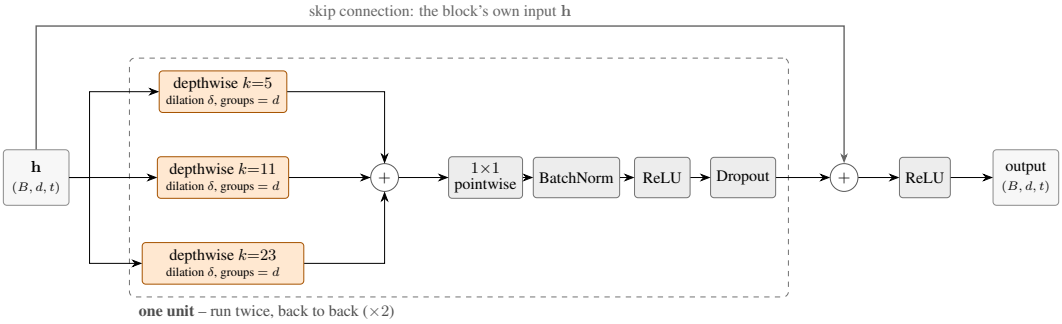
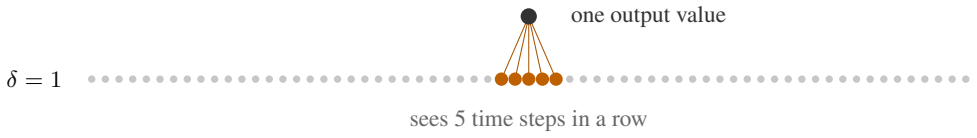


Figure 4: One multi-scale separable block. The three depth wise convolutions read the same input at the kernel sizes 5, 11 and 23, their outputs are summed, and the point-wise 1×1 convolution P mixes the channels; BatchNorm, ReLU and dropout close the unit $\mathcal{U}$ of Equation (1). The dashed box is what is applied twice, and the arrow over the top is the residual connection of Equation (2).

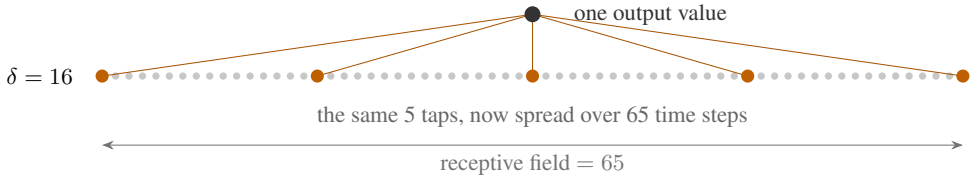
Figure 5 shows what the dilation of Section 3.2 gives. A kernel of size 5 always holds five weights. At $\delta = 1$ they read five neighbouring time points; at $\delta = 16$ the same five weights read time points that are 16 apart and together cover 65. The number of parameters does not change, so the receptive field grows at no cost in weights, and the cap $\delta_{\max} = 16$ keeps it bounded, so that each embedding still describes a local region of the series.

B.2 Where the Bottleneck Saving Comes From

Equation (3) gives the ratio $2r/d$; here is the arithmetic behind the 28 % saving. At $d = 64$ the pointwise step costs $d^2 = 4{,}096$ weights per unit, and with $r = d/4 = 16$ it costs $2dr = 2{,}048$, so each unit saves 2,048 weights. A block holds two units and the model holds four blocks, which gives $8 \times 2{,}048 = 16{,}384$; the bias terms of the factorized layers bring the measured saving to 16,256, i.e. from 57,580 to 41,324 parameters (Tables 3 and 4). The depthwise filters are left unchanged, because with the kernel sizes $\{5, 11, 23\}$ they hold only $39d$ weights per unit and are not where the cost is.



(a) $\delta = 1$: the five taps read five neighbouring time points.



(b) $\delta = 16$: the same five taps span 65 time points.

Figure 5: The effect of the dilation, drawn with the same five taps of the $k = 5$ filter in both rows. Widening the region that the filter sees costs no extra weight, which is why the receptive field of the encoder can be grown without paying for it in parameters.

B.3 The Two Gated Variants, Written Out

Equation (4) writes both channel re-weighting variants in one line. Written separately, *Gated (mean)* first averages the embeddings over time and computes one weight vector from that summary,

$$s_i = \frac{1}{t}\sum_{j=1}^{t} z_i^j \in \mathbb{R}^d, \qquad g_i = \sigma(W_g\, s_i) \in [0,1]^d, \qquad W_g \in \mathbb{R}^{d\times d}, \tag{12}$$

and the same g_i is then applied at every time point, i.e. $g_i^j = g_i$ for all j. The mean is one of three ways to summarize the time axis; the other two are the maximum and the last time point. The mean is used in the main experiments, and the other two appear in Table 21. The cost is one $d \times d$ matrix.

Input Gated keeps the index j: the gate g_i^j is computed from the raw series by a convolution with kernel size 7 and "same" padding, so the gate has exactly t entries, one per time point. Its cost is $7d$ weights, i.e. less than the mean gate, and its gradient reaches the input directly instead of passing through the encoder blocks.

B.4 SEA-Net Spike/Trend Encoder

Equation (6) writes the spike branch as one expression. In the implementation it is four steps in this order: a moving average with window $w = 5$, its subtraction from the input, and then Conv_3, BatchNorm and ReLU, all with "same" padding, so the length t is kept. The mixed features are gated by Equation (12) before they reach the head. We expected an advantage on datasets where the class evidence is a sharp spike, but Table 25 shows that this was the weakest of the five encoders, so the expectation was not met.

B.5 SEA-Net Reconstruction-residual Encoder

The spike branch decides in advance what the encoder misses, namely the sharp changes. This variant learns it instead, by asking which part of the input the encoder features cannot rebuild. A 1×1 convolution reconstructs the raw series from the features, the reconstruction is subtracted from the real input, and the remainder becomes $d_{\text{res}} = 16$ extra channels:

$$\widehat{X}_i = P_{\text{rec}}(Z_i), \qquad \tilde{z}_i^j = P\big([\, z_i^j \,;\, \phi(X_i - \widehat{X}_i)^j \,]\big), \tag{13}$$

where ϕ is the same small convolution, BatchNorm and ReLU used by the spike branch. There is no separate reconstruction loss: the branch is trained end to end with the classifier, so this is a learned residual feature and not an auto-encoder. It was the second strongest encoder in Table 25.

B.6 SEA-NET MULTI-SCALE CHANNELS (WRAPPER)

A *wrapper* changes the input or the output of an encoder but leaves the encoder itself unchanged. The stem receives a single raw channel, so any local statistic that it needs must be learned from scratch. This wrapper adds the rolling maximum, minimum and standard deviation over several windows, together with the sign of the first difference, as extra input channels:

$$X_i^{+} = \Big[X_i \,;\, \big(\text{roll_max}_w(X_i)\big)_{w\in\mathcal{W}} \,;\, \big(\text{roll_min}_w(X_i)\big)_{w\in\mathcal{W}} \,; \\ \big(\text{roll_std}_w(X_i)\big)_{w\in\mathcal{W}} \,;\, \text{sign}(\Delta X_i) \Big], \tag{14}$$

with $\mathcal{W} = \{3, 7, 15, 31\}$, all odd so that each window stays centred. The input grows from 1 to $1 + 4 \cdot 3 + 1 = 14$ channels before the stem, and the rest is one of the encoders above. A rolling mean is not used on purpose: it is linear, so the stem convolution can learn it on its own, whereas the other three statistics are not linear. A BatchNorm first normalizes the 14 channels, because they do not share the scale of the raw series; removing it reduced the accuracy.

B.7 SEA-NET MULTI-SCALE PYRAMID (WRAPPER)

The dilation cap bounds the receptive field, so evidence that is visible only over a long span cannot be seen. This wrapper runs the same encoder on the series downsampled by 1, 2, 4 and 8, stretches the outputs back to length t, and combines them with an attention over the scales:

$$\tilde{z}_i^{\,j} = \sum_{s\in\mathcal{S}} \alpha_i^{j,s}\, z_i^{j,(s)}, \qquad \alpha_i^{j,s} = \text{softmax}_s\big(\text{score}(z_i^{j,(s)})\big), \qquad \mathcal{S} = \{1, 2, 4, 8\}. \tag{15}$$

The softmax normalizes over the *scales* and never over time, so the per-time-point signal on which the two interpretation metrics depend is kept. One set of encoder weights is shared by all scales, so the number of parameters barely changes and the extra cost is compute, about $1.9\times$. This is the one-dimensional analogue of the multi-magnification MIL of Hashimoto et al. (2020), and it was the weakest variant we built (Table 25). It also has a constraint: the wrapped encoder must be shallow enough that its receptive field does not exceed $t/\max(\mathcal{S})$, otherwise the coarsest scales collapse to a nearly constant signal. The pyramid therefore uses a two-block base, and the pipeline checks this automatically.

Alternatives we rejected. A Transformer encoder costs $\mathcal{O}(t^2)$ in time and memory, which is too expensive for series of 1,000–1,500 points on the target hardware. Its unrestricted attention also lets any output position read any input position, which removes the bounded receptive field that the interpretation relies on. Pruning or quantizing `InceptionTime` would reduce its size, but would not change where the cost is, i.e. in the encoder, which is the point of this paper.

B.8 THE TWO ENCODERS, LAYER BY LAYER

Table 3 and Table 4 give one row per operation, with its output shape and its number of parameters. Read side by side, they show that the saving of Appendix B.2 comes only from the pointwise layers.

Table 3: The SEA-Net bottleneck encoder ($d = 64$, 4 blocks, kernels 5/11/23, dilations 1/2/4/8), layer by layer, with the output shape and the number of parameters, built from `top/top_bottleneck_topk` for WebTraffic ($c = 10$). The point-wise convolution of every unit is factorised as $d \to d/4 \to d$, which is where the parameter saving comes from. Every layer keeps the series length t, so the head sees one embedding per time point. Depthwise = one filter per channel (groups $= d$).

Stage	**Layer**	**Output**	**Params**
Input	z-normalised series	$b \times 1 \times t$	–
Stem	Conv1d $1 \to 64$, kernel 7	$b \times 64 \times t$	512
Block 1 (dilation 1), $2\times$ unit:	depthwise Conv1d $\times 3$ (kernels 5, 11, 23), summed	$b \times 64 \times t$	2,688
	pointwise Conv1d $64 \to 16$ (reduce)	$b \times 16 \times t$	1,040
	pointwise Conv1d $16 \to 64$ (expand)	$b \times 64 \times t$	1,088
	BatchNorm + ReLU + dropout	$b \times 64 \times t$	128
	after both units: residual add + ReLU	$b \times 64 \times t$	*block total 9,888*
Block 2 (dilation 2), $2\times$ unit:	depthwise Conv1d $\times 3$ (kernels 5, 11, 23), summed	$b \times 64 \times t$	2,688
	pointwise Conv1d $64 \to 16$ (reduce)	$b \times 16 \times t$	1,040
	pointwise Conv1d $16 \to 64$ (expand)	$b \times 64 \times t$	1,088
	BatchNorm + ReLU + dropout	$b \times 64 \times t$	128
	after both units: residual add + ReLU	$b \times 64 \times t$	*block total 9,888*
Block 3 (dilation 4), $2\times$ unit:	depthwise Conv1d $\times 3$ (kernels 5, 11, 23), summed	$b \times 64 \times t$	2,688
	pointwise Conv1d $64 \to 16$ (reduce)	$b \times 16 \times t$	1,040
	pointwise Conv1d $16 \to 64$ (expand)	$b \times 64 \times t$	1,088
	BatchNorm + ReLU + dropout	$b \times 64 \times t$	128
	after both units: residual add + ReLU	$b \times 64 \times t$	*block total 9,888*
Block 4 (dilation 8), $2\times$ unit:	depthwise Conv1d $\times 3$ (kernels 5, 11, 23), summed	$b \times 64 \times t$	2,688
	pointwise Conv1d $64 \to 16$ (reduce)	$b \times 16 \times t$	1,040
	pointwise Conv1d $16 \to 64$ (expand)	$b \times 64 \times t$	1,088
	BatchNorm + ReLU + dropout	$b \times 64 \times t$	128
	after both units: residual add + ReLU	$b \times 64 \times t$	*block total 9,888*
Encoder total		$b \times 64 \times t$ (time-point embs.)	40,064
Pooling head	Top-k conjunctive ($a = 8$, $c = 10$)	$b \times 10$	1,260
Model total			41,324

Table 4: The SEA-Net base encoder ($d = 64$, 4 blocks, kernels 5/11/23, dilations 1/2/4/8), layer by layer, with the output shape and the number of parameters, built from `seanet/seanet_slim_topk` for WebTraffic ($c = 10$). It has the same skeleton with the plain $d \to d$ point-wise convolution, for comparison. Every layer keeps the series length t, so the head sees one embedding per time point. Depthwise = one filter per channel (groups $= d$).

Stage	**Layer**	**Output**	**Params**
Input	z-normalised series	$b \times 1 \times t$	–
Stem	Conv1d $1 \to 64$, kernel 7	$b \times 64 \times t$	512
Block 1 (dilation 1), $2\times$ unit:	depthwise Conv1d $\times 3$ (kernels 5, 11, 23), summed	$b \times 64 \times t$	2,688
	pointwise Conv1d $64 \to 64$	$b \times 64 \times t$	4,160
	BatchNorm + ReLU + dropout	$b \times 64 \times t$	128
	after both units: residual add + ReLU	$b \times 64 \times t$	*block total 13,952*
Block 2 (dilation 2), $2\times$ unit:	depthwise Conv1d $\times 3$ (kernels 5, 11, 23), summed	$b \times 64 \times t$	2,688
	pointwise Conv1d $64 \to 64$	$b \times 64 \times t$	4,160
	BatchNorm + ReLU + dropout	$b \times 64 \times t$	128
	after both units: residual add + ReLU	$b \times 64 \times t$	*block total 13,952*
Block 3 (dilation 4), $2\times$ unit:	depthwise Conv1d $\times 3$ (kernels 5, 11, 23), summed	$b \times 64 \times t$	2,688
	pointwise Conv1d $64 \to 64$	$b \times 64 \times t$	4,160
	BatchNorm + ReLU + dropout	$b \times 64 \times t$	128
	after both units: residual add + ReLU	$b \times 64 \times t$	*block total 13,952*
Block 4 (dilation 8), $2\times$ unit:	depthwise Conv1d $\times 3$ (kernels 5, 11, 23), summed	$b \times 64 \times t$	2,688
	pointwise Conv1d $64 \to 64$	$b \times 64 \times t$	4,160
	BatchNorm + ReLU + dropout	$b \times 64 \times t$	128
	after both units: residual add + ReLU	$b \times 64 \times t$	*block total 13,952*
Encoder total		$b \times 64 \times t$ (time-point embs.)	56,320
Pooling head	Top-k conjunctive ($a = 8$, $c = 10$)	$b \times 10$	1,260
Model total			57,580

C Pooling Heads Compared

Section 3.3 describes the MIL pooling head ψ_{pool}. It has two branches: an attention branch, which decides how much each time point counts, and a classifier branch, which decides which class each time point argues for. Their product is the evidence $e_i^{j,k}$ of time point j for class k; aggregating it over time gives the prediction $\hat{Y}_i$, and keeping it per time point gives the interpretation $\hat{I}_i$. The main text starts from the conjunctive pooling of MILLET (Equation (7)), states its three limitations, and gives the three heads that remove them: class-wise (Equation (8)), Soft-Max (Equation (9)) and Top-k (Equation (10)) conjunctive pooling.

This appendix adds three things. First, it draws every head in the same way, one full-width figure per head (Figure 6 to Figure 13). All seven of our heads keep the classifier branch $\hat{y}_i^j = W_c z_i^j + \mathbf{b}_c$ unchanged and differ from the baseline only in two places, the attention head and the aggregation over time; these two places are drawn in blue, so two heads can be compared by looking at the blue boxes only. Second, Appendix C.5 to Appendix C.8 describe the four heads that the main text does not cover, with their equations and the reason why each was built. Third, Appendix C.9 summarizes the eight heads and gives each of them layer by layer.

C.1 Conjunctive Pooling

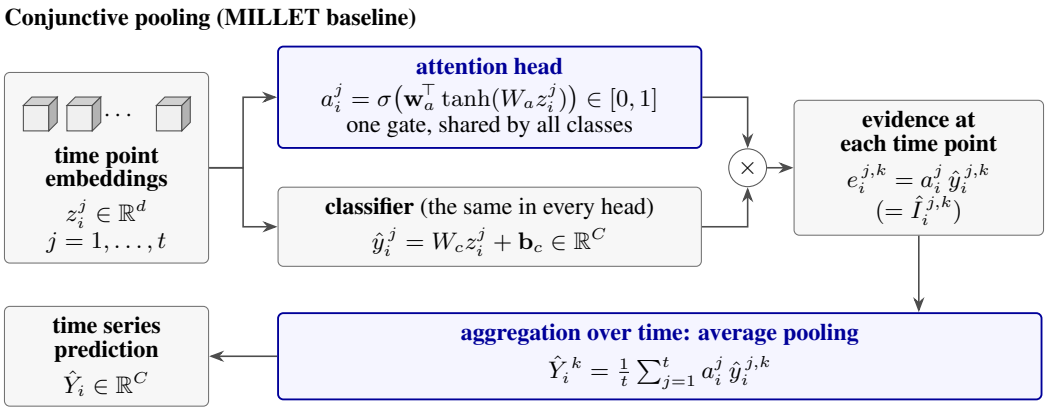


Figure 6: Conjunctive pooling, the MILLET baseline of Equation (7). Read from left to right: the time point embeddings feed an attention head and a classifier, their product is the evidence at each time point, which is also the interpretation, and aggregating that evidence over time gives the prediction for the series. The classifier is the same in every head; only the two blue boxes, the attention head and the aggregation, change from one head to the next, so Figure 7 to Figure 13 are best read against this figure.

Figure 6 draws the baseline of Equation (7). The head itself is almost free: replacing it by global average pooling on the same `InceptionTime` backbone removes only 1,041 of the 423,707 parameters, so it is the encoder, and not the interpretability, that makes the baseline large. Section 3.3 states the three limitations of this head. We call them **L1** (one gate is shared by all classes), **L2** (the weights are not normalized over time) and **L3** (the fixed mean dilutes a few decisive time points in a long series), and use these names below and in Table 5.

C.2 Class-wise Conjunctive Pooling

Compared with Equation (7), Equation (8) changes only the attention head (Figure 7). The gate becomes a vector of length C, so the evidence is formed class by class, and a time point can support one class while arguing against another. This head targets **L1**.

C.3 Soft-Max Conjunctive Pooling

Here the aggregation changes as well, as Figure 8 shows: the sigmoid gate becomes a Soft-Max over time, and the mean becomes a weighted sum. This head targets **L2**.

Class-wise conjunctive pooling

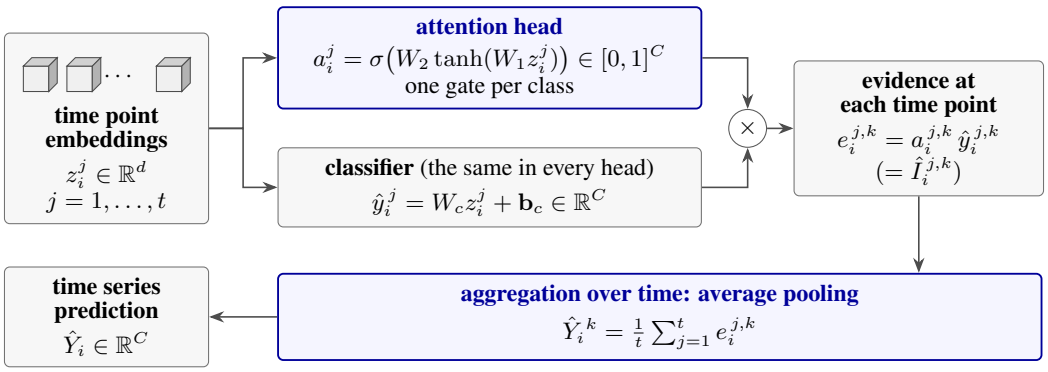


Figure 7: Class-wise conjunctive pooling. Against Figure 6, only the attention head changes: widening its last layer to C outputs costs $(C-1)(d_a+1)$ weights and makes the evidence, and therefore the interpretation, class-specific.

Soft-Max conjunctive pooling

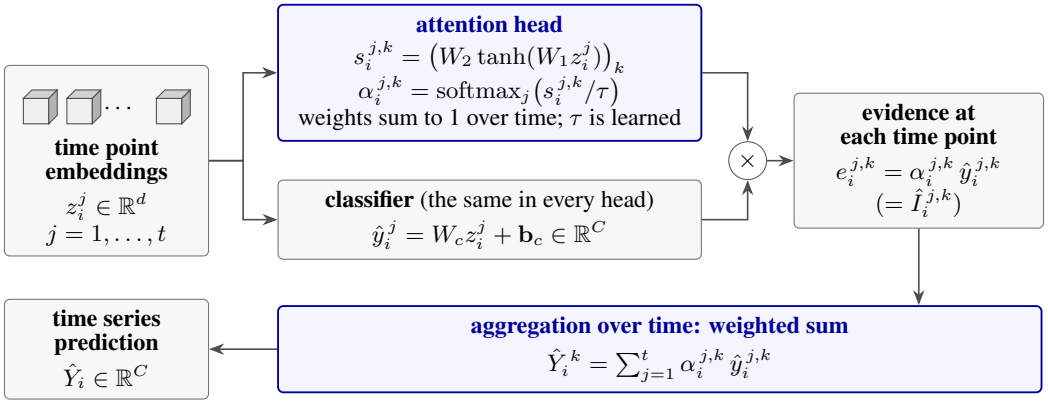


Figure 8: Soft-Max conjunctive pooling. Both blue boxes change compared with Figure 6: the attention is normalized over time, and the aggregation is a weighted sum rather than a mean. The single extra weight is the sharpness $\tau = \exp(\rho)$.

Property 1 (Length invariance)**.** *Since the weights of Equation (9) sum to one over j, $\hat{Y}_i^{\,k}$ does not depend on t. As $\tau \to \infty$ they flatten to $1/t$ and the head becomes a plain mean over time; as $\tau \to 0$ it keeps the single highest-scoring time point.*

C.4 Top-k Conjunctive Pooling

The attention head is the class-wise one, and only the aggregation changes, as shown in Figure 9. This head targets **L3**.

Property 2 (Strict generalisation)**.** *At $\kappa = 1$, Equation (10) is exactly Equation (8). For $\kappa < 1$ the gradient reaches only the selected time points.*

Property 2 makes the κ study of Table 26 a controlled experiment: the $\kappa = 1$ row *is* the class-wise head, so any difference from it comes from the selection alone and not from a change of model.

The remaining four heads are not described in the main text. Each of them keeps the class-wise classifier and changes the attention head, the aggregation or both, and each targets one or two of the limitations above.

C.5 Adaptive Class-wise Pooling

This head also targets **L3**, but from another side. A fixed mean suits evidence that is spread over the series, and a maximum suits a single short event; no fixed operator is right for both. The head

Top-k conjunctive pooling

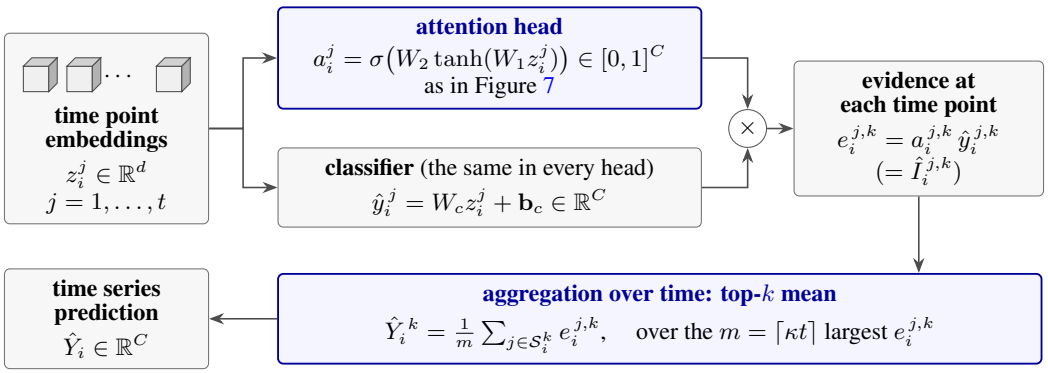


Figure 9: Top-k conjunctive pooling. The attention head is the one of Figure 7, and only the aggregation moves: the mean runs over the m largest evidence values of the class instead of over all t. It adds no weight, because κ is a hyperparameter.

Adaptive class-wise pooling

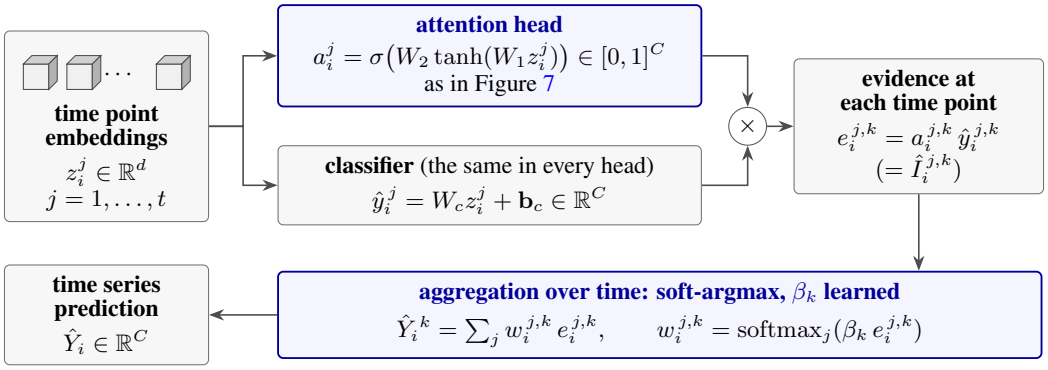


Figure 10: Adaptive class-wise pooling (Equation (16)). The attention head is the one of Figure 7; the aggregation slides between the mean and the maximum according to one learned number per class.

therefore keeps the class-wise gate and aggregates with a learnable sharpness $\beta_k \geq 0$ per class, written as a numerically stable soft-argmax (Figure 10):

$$\beta_k = \mathrm{softplus}(\theta_k), \qquad w_i^{j,k} = \mathrm{softmax}_j\big(\beta_k\, e_i^{j,k}\big), \qquad \hat{Y}_i^k = \sum_{j=1}^{t} w_i^{j,k}\, e_i^{j,k}. \tag{16}$$

It adds C weights.

Property 3 (Learned mean–max aggregator)**.** *$\beta_k \to 0$ gives $w_i^{j,k} \to 1/t$, i.e. class-wise pooling; $\beta_k \to \infty$ gives $\hat{Y}_i^k \to \max_j e_i^{j,k}$. As $\hat{Y}_i^k$ is always a convex combination of the $e_i^{j,k}$, it stays inside their range, which keeps training stable.*

We initialize β_k at 0.3, so that training starts close to the class-wise mean and becomes sharper only where this lowers the loss.

C.6 Attention-max Pooling

This is the simplest hard alternative to the previous head, and it serves as a control: the soft-argmax still gives every time point some weight, and this head tests whether that is useful. The aggregation is a learnable per-class blend of the mean and the maximum (Figure 11), again with C extra weights:

$$\hat{Y}_i^k = (1 - \lambda_k)\, \frac{1}{t} \sum_{j=1}^{t} e_i^{j,k} \;+\; \lambda_k \max_j e_i^{j,k}, \qquad \lambda_k = \sigma(\theta_k) \in [0,1]. \tag{17}$$

$\lambda_k \to 0$ recovers class-wise pooling and $\lambda_k \to 1$ gives max pooling, so each class learns on its own whether its evidence is sustained or a short event. Equation (17) and Equation (16) use the same

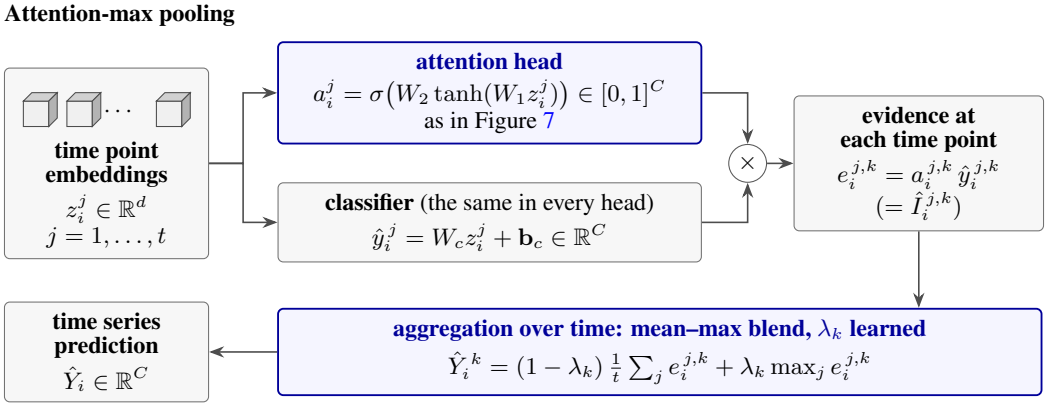


Figure 11: Attention-max pooling (Equation (17)), the hard counterpart of Figure 10: one learned number per class blends the mean with the maximum directly.

gate and differ only in how they aggregate it, so comparing them isolates soft selection from hard selection.

C.7 Per-class Gated Attention

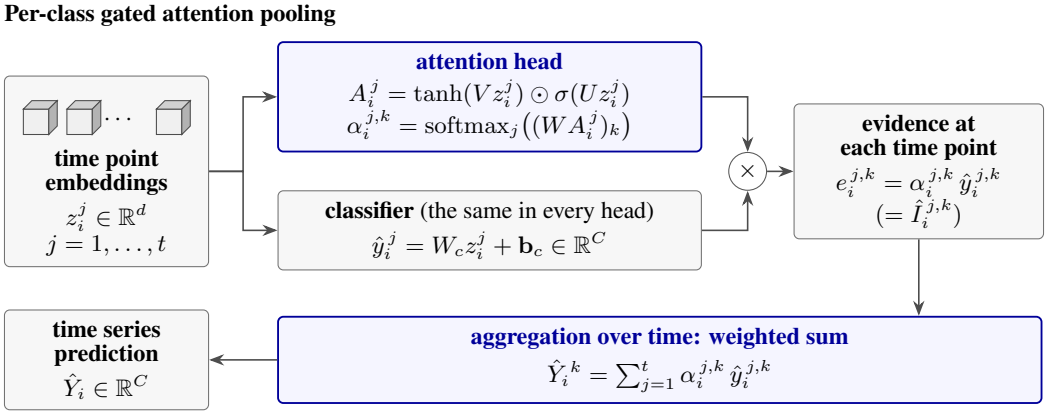


Figure 12: Per-class gated attention (Equation (18) and Equation (19)). The attention head has two layers instead of one, and the second of them decides which attention features are admitted.

Gated attention was proposed by Ilse et al. (2018) and is also used by Hashimoto et al. (2020), where a bag is a whole-slide image and an instance is a patch. It targets **L1** and **L3** together with a stronger attention head, because the baseline gate is a single $\tanh$ layer, which limits how selective it can be. We apply it to time series and make it per class, which the original is not (Figure 12):

$$A_i^j = \tanh(V z_i^j) \odot \sigma(U z_i^j) \in \mathbb{R}^{d_a}, \qquad V, U \in \mathbb{R}^{d_a \times d}, \tag{18}$$

$$\alpha_i^{j,k} = \mathrm{softmax}_j\left((W A_i^j)_k\right), \qquad \hat{Y}_i^k = \sum_{j=1}^{t} \alpha_i^{j,k}\, \hat{y}_i^{j,k}. \tag{19}$$

The $\tanh$ branch proposes attention features and the σ branch decides which of them to let through, so a feature is used only where the second branch judges it reliable. The cost is a second $d \to d_a$ layer, which makes this the most expensive head here. Unlike the other heads, it has no blend parameter that recovers conjunctive pooling, so no setting of its weights returns the baseline.

C.8 Dual-stream Conjunctive Pooling

The two-stream mechanism was proposed by Li et al. (2021), again for whole-slide images: one stream aggregates all the instances, and a second one focuses on the most critical instance and on the instances similar to it. It targets **L3** when the evidence is scattered, i.e. one decisive time point should

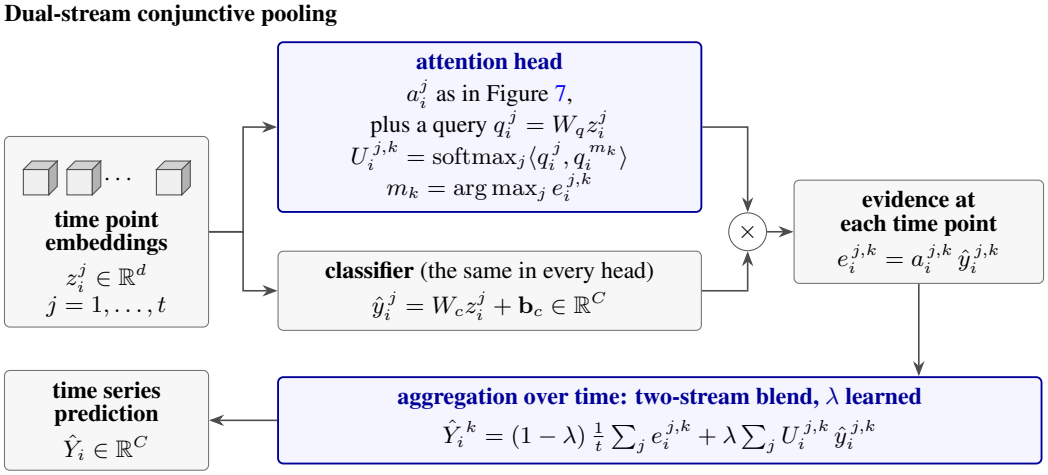


Figure 13: Dual-stream conjunctive pooling (Equation (20) and Equation (21)). A second stream, built around the most supporting time point of each class, is blended into the class-wise stream of Figure 7 by one learned scalar.

be able to dominate without the rest being discarded. We removed the contrastive pre-training of Li et al. (2021), made the critical instance and the query per class, and blended the two streams with a learnable scalar that starts close to the baseline. Stream A is the class-wise mean of Equation (8); stream B finds the most supporting time point of each class and re-weights the time points by their similarity to its query (Figure 13):

$$m_k = \arg\max_j \; e_i^{j,k}, \qquad q_i^{\,j} = W_q z_i^j \in \mathbb{R}^{d_a}, \qquad U_i^{j,k} = \mathrm{softmax}_j\big(\langle q_i^j, q_i^{m_k} \rangle\big), \tag{20}$$

$$\hat{Y}_i{}^k = (1-\lambda)\, \frac{1}{t} \sum_{j=1}^{t} e_i^{j,k} + \lambda \sum_{j=1}^{t} U_i^{j,k}\, \hat{y}_i^{j,k}, \qquad \lambda = \sigma(\theta), \;\; \lambda_0 = 0.05. \tag{21}$$

This head is also the clearest negative result of the study: being able to return to the baseline is **not** a safety guarantee. Five of the seven heads contain conjunctive pooling as a special case, and this is one of them, yet it is the worst head we tested; its four pairings score 0.860, 0.824, 0.736 and 0.556 (Table 24). The training runs show why: λ starts at 0.05 and nothing in the loss pushes it back towards the baseline, so being *able* to return there does not mean that training will go there. This is why the main text argues for Top-k from the measured NDCG@n and not from Property 2 alone.

C.9 The Eight Heads Together

Table 5 lists the eight heads by the two choices that define them, the attention head and the aggregation over time. Table 6 to Table 13 then give the eight heads layer by layer, one table per head, with the input and the output shape of each operation. In these tables, b is the batch size, t the series length, d the embedding size (64 for SEA-Net, 128 for MILLET), $a = 8$ the attention width and c the number of classes.

Table 5: The eight heads as two separate choices. *Reduces at* gives the setting at which the head becomes conjunctive pooling exactly, and *Targets* gives the limitations of Appendix C.1 that it addresses.

#	Head	Attention	Reduction over time	Reduces at	Targets
1	Conjunctive (baseline)	shared gate	mean	—	—
2	Class-wise conjunctive	per-class gate	mean	$a_i^{j,k} = a_i^j$	L1
3	Soft-Max conjunctive	per-class score	softmax, learnable τ	$\tau \to \infty$	L2
4	Top-k conjunctive	per-class gate	mean of the m largest	$\kappa = 1$	L3
5	Adaptive class-wise	per-class gate	soft-argmax, learnable β_k	$\beta_k \to 0$	L3
6	Attention-max	per-class gate	blend of mean and max	$\lambda_k \to 0$	L3
7	Per-class gated attention	gated $\tanh \odot \sigma$	softmax	**none**	L1, L3
8	Dual-stream conjunctive	gate + critical query	blend of two streams	$\lambda \to 0$	L3

Table 6: Conjunctive pooling, the MILLET baseline head (their Table A.5; $d = 128$). One scalar gate per time point, shared by every class. Registered as `mil_conjunctive`.

Process	Layer	Input	Output
Input	Positional encoding + dropout	$b \times t \times d$	$b \times t \times d$ (time-point embs.)
Attention	Linear + tanh	$b \times t \times d$	$b \times t \times a$
	Linear + sigmoid	$b \times t \times a$	$b \times t \times 1$ (attention scores)
Classifier	Linear	$b \times t \times d$	$b \times t \times c$ (time-point logits)
Pooling	Attention weighting	$b \times t \times c$	$b \times t \times c$ (interpretation)
	Mean over t	$b \times t \times c$	$b \times c$ (series logits)

Table 7: Class-wise conjunctive pooling: conjunctive pooling with one gate per class instead of one shared gate. Registered as `sea_classwise_conjunctive`.

Process	Layer	Input	Output
Input	Positional encoding + dropout	$b \times t \times d$	$b \times t \times d$ (time-point embs.)
Attention	Linear + tanh	$b \times t \times d$	$b \times t \times a$
	Linear + sigmoid	$b \times t \times a$	$b \times t \times c$ (one gate per class)
Classifier	Linear	$b \times t \times d$	$b \times t \times c$ (time-point logits)
Evidence	Gate $\times$ logit (element-wise)	$b \times t \times c$	$b \times t \times c$ (interpretation)
Pooling	Mean over t	$b \times t \times c$	$b \times c$ (series logits)

Table 8: Top-k conjunctive pooling, the head of every short-listed model: class-wise conjunctive pooling, but the series logit averages only the $k = \lceil \kappa t \rceil$ largest evidence values of each class ($\kappa = 0.1$). The interpretation is the full evidence map, so it is exactly zero wherever no evidence was used. Registered as `sea_topk_conjunctive`.

Process	Layer	Input	Output
Input	Positional encoding + dropout	$b \times t \times d$	$b \times t \times d$ (time-point embs.)
Attention	Linear + tanh	$b \times t \times d$	$b \times t \times a$
	Linear + sigmoid	$b \times t \times a$	$b \times t \times c$ (one gate per class)
Classifier	Linear	$b \times t \times d$	$b \times t \times c$ (time-point logits)
Evidence	Gate $\times$ logit (element-wise)	$b \times t \times c$	$b \times t \times c$ (interpretation)
Pooling	Top-k over t (per class)	$b \times t \times c$	$b \times k \times c$
	Mean over k	$b \times k \times c$	$b \times c$ (series logits)

Table 9: Soft-Max conjunctive pooling: the attention is a distribution over time (a softmax with one learned temperature τ), so the series logit is a true weighted average and long and short series are on one scale. Registered as `sea_softmax_conjunctive`.

Process	Layer	Input	Output
Input	Positional encoding + dropout	$b \times t \times d$	$b \times t \times d$ (time-point embs.)
Attention	Linear + tanh	$b \times t \times d$	$b \times t \times a$
	Linear (raw scores)	$b \times t \times a$	$b \times t \times c$
	$\div \tau$, softmax over t	$b \times t \times c$	$b \times t \times c$ (weights, sum to 1)
Classifier	Linear	$b \times t \times d$	$b \times t \times c$ (time-point logits)
Pooling	Weight $\times$ logit (element-wise)	$b \times t \times c$	$b \times t \times c$ (interpretation)
	Sum over t	$b \times t \times c$	$b \times c$ (series logits)

Table 10: Adaptive class-wise pooling: class-wise conjunctive pooling with a learned mean-to-max aggregator, i.e. a soft-argmax with one sharpness $\beta_k = \mathrm{softplus}(\theta_k) \geq 0$ per class (c extra parameters; $\beta \to 0$ gives the mean and $\beta \to \infty$ the max). Registered as `sea_adaptive_classwise`.

Process	Layer	Input	Output
Input	Positional encoding + dropout	$b \times t \times d$	$b \times t \times d$ (time-point embs.)
Attention	Linear + tanh	$b \times t \times d$	$b \times t \times a$
	Linear + sigmoid	$b \times t \times a$	$b \times t \times c$ (one gate per class)
Classifier	Linear	$b \times t \times d$	$b \times t \times c$ (time-point logits)
Evidence	Gate $\times$ logit (element-wise)	$b \times t \times c$	$b \times t \times c$ (interpretation)
Pooling	softmax over t of $\beta_k \cdot$ evidence	$b \times t \times c$	$b \times t \times c$ (weights)
	Weighted sum over t	$b \times t \times c$	$b \times c$ (series logits)

Table 11: Attention-max pooling: class-wise conjunctive pooling with a learned per-class blend $\lambda_k = \sigma(\theta_k)$ of the mean and the max over time (c extra parameters). Registered as `sea_attention_max`.

Process	Layer	Input	Output
Input	Positional encoding + dropout	$b \times t \times d$	$b \times t \times d$ (time-point embs.)
Attention	Linear + tanh	$b \times t \times d$	$b \times t \times a$
	Linear + sigmoid	$b \times t \times a$	$b \times t \times c$ (one gate per class)
Classifier	Linear	$b \times t \times d$	$b \times t \times c$ (time-point logits)
Evidence	Gate $\times$ logit (element-wise)	$b \times t \times c$	$b \times t \times c$ (interpretation)
Pooling	Mean over t	$b \times t \times c$	$b \times c$
	Max over t	$b \times t \times c$	$b \times c$
	$(1 - \lambda_k)$ mean $+\lambda_k$ max	$b \times c$	$b \times c$ (series logits)

Table 12: Per-class gated attention pooling (Ilse et al. 2018, made class-specific): a tanh branch multiplied by a sigmoid branch scores every time point, and a softmax over time turns the scores into weights. Registered as `sea_gated_attention`.

Process	Layer	Input	Output
Input	Positional encoding + dropout	$b \times t \times d$	$b \times t \times d$ (time-point embs.)
Attention	Linear + tanh (V)	$b \times t \times d$	$b \times t \times a$
	Linear + sigmoid (U)	$b \times t \times d$	$b \times t \times a$
	Element-wise product	$b \times t \times a$	$b \times t \times a$ (gated features)
	Linear (scores), softmax over t	$b \times t \times a$	$b \times t \times c$ (weights, sum to 1)
Classifier	Linear	$b \times t \times d$	$b \times t \times c$ (time-point logits)
Pooling	Weight $\times$ logit (element-wise)	$b \times t \times c$	$b \times t \times c$ (interpretation)
	Sum over t	$b \times t \times c$	$b \times c$ (series logits)

Table 13: Dual-stream conjunctive pooling (DSMIL, Li et al. 2021, wrapped around class-wise conjunctive pooling): stream A is the plain mean, and stream B re-weights the time points by their similarity to the most supporting one (the critical time point) of each class; a learned scalar $\lambda = \sigma(\theta)$ blends the two and starts near stream A. Registered as `sea_dualstream_conjunctive`.

Process	Layer	Input	Output
Input	Positional encoding + dropout	$b \times t \times d$	$b \times t \times d$ (time-point embs.)
Attention	Linear + tanh	$b \times t \times d$	$b \times t \times a$
	Linear + sigmoid	$b \times t \times a$	$b \times t \times c$ (one gate per class)
Classifier	Linear	$b \times t \times d$	$b \times t \times c$ (time-point logits)
Evidence	Gate $\times$ logit (element-wise)	$b \times t \times c$	$b \times t \times c$ (interpretation)
Stream A	Mean over t	$b \times t \times c$	$b \times c$
Stream B	Linear (query)	$b \times t \times d$	$b \times t \times a$
	argmax over t of the evidence	$b \times t \times c$	$b \times c$ (critical index)
	Gather the critical query	$b \times t \times a$	$b \times c \times a$
	Dot product with every query, softmax over t	$b \times t \times a$	$b \times t \times c$ (weights)
	Weight $\times$ logit, sum over t	$b \times t \times c$	$b \times c$
Blend	$(1-\lambda)$ A $+\lambda$ B	$b \times c$	$b \times c$ (series logits)

D Reproducing the Experiments

Section 4 gives the experimental setup in short: the two datasets, WebTraffic and the 85 UCR datasets used by MILLET, the metrics, i.e. the test accuracy for the classification and AOPCR and NDCG@n for the interpretation, and the MILLET baseline. This appendix gives the details needed to reproduce the experiments. Appendix D.1 shows the WebTraffic classes and lists every dataset, Appendix D.2 describes the baselines and how they were trained, Appendix D.3 gives the training configuration shared by all models, Appendix D.4 defines the two interpretation metrics, and Appendix D.5 lists every hyperparameter and the size of every trained model.

D.1 Datasets

WebTraffic is the only dataset in this work with a ground truth for each time point, so it is the only one on which the interpretation can be scored with NDCG@n. Figure 14 shows one example of each of its classes. Table 14 lists WebTraffic and the 85 UCR datasets with their train and test sizes, lengths, numbers of classes and class imbalance.

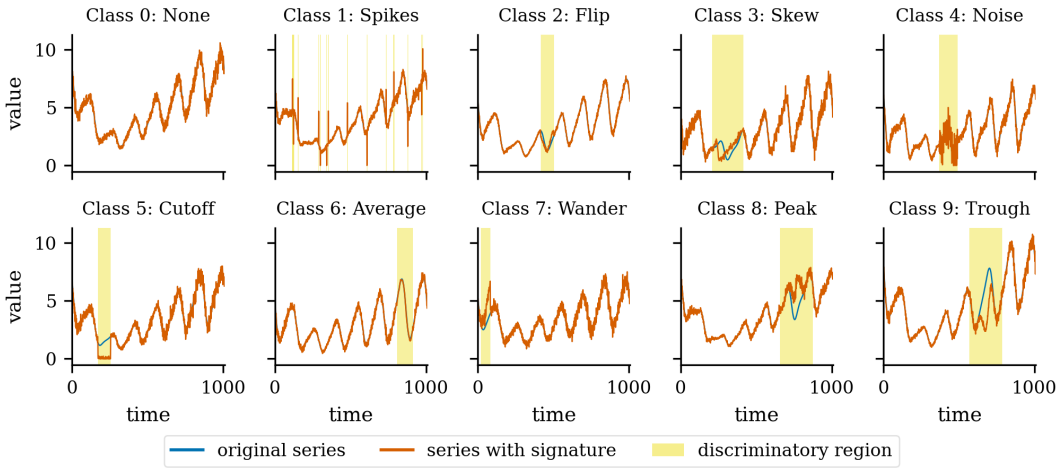


Figure 14: One example of each class of the WebTraffic dataset. Each series is a daily seasonal signal with a weekly trend and noise, and the signature that defines the class is injected inside a single window, shown shaded; the exception is class 0 (None), which has no signature. These windows are the per-time-point ground truth against which NDCG@n is scored, and they are never used during training.

Table 14: The datasets: WebTraffic (the only one with per-time-point labels, hence the only one with NDCG@n) and the 85 UCR datasets that MILLET publishes, in the order of MILLET. *Imbalance* is the largest training-class share divided by the smallest (1.00 = balanced). The sizes are the original train / test splits; our training holds out 20 % of the training set for validation when it has at least 100 series.

#	Dataset	Train	Test	Length	Classes	Imbalance
	WebTraffic	500	500	1,008	10	1.00
1	Adiac	390	391	176	37	3.75
2	ArrowHead	36	175	251	3	1.00
3	Beef	30	30	470	5	1.00
4	BeetleFly	20	20	512	2	1.00
5	BirdChicken	20	20	512	2	1.00
6	Car	60	60	577	4	1.55
7	CBF	30	900	128	3	1.50
8	ChlorineConcentration	467	3,840	166	3	2.88
9	CinCECGTorso	40	1,380	1,639	4	2.60
10	Coffee	28	28	286	2	1.00
11	Computers	250	250	720	2	1.00
12	CricketX	390	390	300	12	1.56
13	CricketY	390	390	300	12	1.29
14	CricketZ	390	390	300	12	1.64
15	DiatomSizeReduction	16	306	345	4	6.00
16	DistalPhalanxOutlineAgeGroup	400	139	80	3	8.57
17	DistalPhalanxOutlineCorrect	600	276	80	2	1.70
18	DistalPhalanxTW	400	139	80	6	11.83
19	Earthquakes	322	139	512	2	4.55
20	ECG200	100	100	96	2	2.23
21	ECG5000	500	4,500	140	5	146.00
22	ECGFiveDays	23	861	136	2	1.56
23	ElectricDevices	8,926	7,711	96	7	4.73
24	FaceAll	560	1,690	131	14	1.00
25	FaceFour	24	88	350	4	2.67
26	FacesUCR	200	2,050	131	14	8.25
27	FiftyWords	450	455	270	50	52.00
28	Fish	175	175	463	7	1.33
29	FordA	3,601	1,320	500	2	1.05
30	FordB	3,636	810	500	2	1.05
31	GunPoint	50	150	150	2	1.08
32	Ham	109	105	431	2	1.10
33	HandOutlines	1,000	370	2,709	2	1.76
34	Haptics	155	308	1,092	5	2.00
35	Herring	64	64	512	2	1.56
36	InlineSkate	100	550	1,882	7	2.00
37	InsectWingbeatSound	220	1,980	256	11	1.00
38	ItalyPowerDemand	67	1,029	24	2	1.03
39	LargeKitchenAppliances	375	375	720	3	1.00
40	Lightning2	60	61	637	2	2.00
41	Lightning7	70	73	319	7	3.80
42	Mallat	55	2,345	1,024	8	5.50
43	Meat	60	60	448	3	1.00
44	MedicalImages	381	760	99	10	33.83
45	MiddlePhalanxOutlineAgeGroup	400	154	80	3	4.31
46	MiddlePhalanxOutlineCorrect	600	291	80	2	1.83
47	MiddlePhalanxTW	399	154	80	6	6.40
48	MoteStrain	20	1,252	84	2	1.00
49	NonInvasiveFetalECGThorax1	1,800	1,965	750	42	1.60

continued on the next page

Table 14 (continued)

#	Dataset	Train	Test	Length	Classes	Imbalance
50	NonInvasiveFetalECGThorax2	1,800	1,965	750	42	1.60
51	OliveOil	30	30	570	4	3.25
52	OSULeaf	200	242	427	6	3.53
53	PhalangesOutlinesCorrect	1,800	858	80	2	1.87
54	Phoneme	214	1,896	1,024	39	24.00
55	Plane	105	105	144	7	2.22
56	ProximalPhalanxOutlineAgeGroup	400	205	80	3	2.62
57	ProximalPhalanxOutlineCorrect	600	291	80	2	2.09
58	ProximalPhalanxTW	400	205	80	6	11.25
59	RefrigerationDevices	375	375	720	3	1.00
60	ScreenType	375	375	720	3	1.00
61	ShapeletSim	20	180	500	2	1.00
62	ShapesAll	600	600	512	60	1.00
63	SmallKitchenAppliances	375	375	720	3	1.00
64	SonyAIBORobotSurface1	20	601	70	2	2.33
65	SonyAIBORobotSurface2	27	953	65	2	1.45
66	StarLightCurves	1,000	8,236	1,024	3	3.77
67	Strawberry	613	370	235	2	1.80
68	SwedishLeaf	500	625	128	15	1.62
69	Symbols	25	995	398	6	2.67
70	SyntheticControl	300	300	60	6	1.00
71	ToeSegmentation1	40	228	277	2	1.00
72	ToeSegmentation2	36	130	343	2	1.00
73	Trace	100	100	275	4	1.48
74	TwoLeadECG	23	1,139	82	2	1.09
75	TwoPatterns	1,000	4,000	128	4	1.14
76	UWaveGestureLibraryAll	896	3,582	945	8	1.27
77	UWaveGestureLibraryX	896	3,582	315	8	1.27
78	UWaveGestureLibraryY	896	3,582	315	8	1.27
79	UWaveGestureLibraryZ	896	3,582	315	8	1.27
80	Wafer	1,000	6,164	152	2	9.31
81	Wine	57	54	234	2	1.11
82	WordSynonyms	267	638	270	25	30.00
83	Worms	181	77	900	5	4.47
84	WormsTwoClass	181	77	900	2	1.38
85	Yoga	300	3,000	426	2	1.19
	UCR-85: min / median / max	16 / 267 / 8926	20 / 375 / 8236	24 / 300 / 2709	2 / 3 / 60	1.00 / 1.60 / 146.00

D.2 BASELINES

MILLET (`InceptionTime` encoder + `Conjunctive` pooling) is the baseline of this work. We also train FCN and ResNet (Wang et al., 2017) with the same pooling head, to have two more reference points. The faithfulness numbers of two different training recipes cannot be compared (Section 4.1), so we retrain every baseline ourselves with our own training budget, which is the only fair comparison. We also list the published numbers of MILLET (Early et al., 2024) for reference (Table 34).

Two things differ between the recipe behind the published numbers and ours. First, they keep no validation split and use no early stopping, so over-fitting cannot be observed during training. Second, they train with hard targets, i.e. the whole probability mass is put on one class, whereas we add label smoothing.

D.3 Training Configuration

We use Optuna to search for the hyperparameters, and the values it selects are then used for every model of this work. Hence, a difference between two rows of a result table is a difference between two models, and not between two training setups. Table 15 gives the shared setting, and Table 16 gives the remaining hyperparameters, with one column per model.

Table 15: Training configuration, identical for every model of this study. The complete hyperparameter list, including the head-specific initial values, is given in Table 16.

Setting	**Value**
Encoder size (every SEA-Net model)	width $d = 64$, 4 blocks; 41,324–69,768 parameters
Optimizer	Adam (Kingma & Ba, 2015), lr $= 1.25 \times 10^{-3}$, weight decay $= 1.2 \times 10^{-4}$
Epochs / early stopping	max 400 epochs; stop after 60 epochs with no improvement
Batch size	$\min(\lfloor n_{\text{train}}/10 \rfloor, 16)$, floor of 2
Validation split	stratified 80/20 when $n_{\text{train}} \geq 100$; else train-loss early stopping
Label smoothing	0.13
Dropout	0.2
Attention width d_a	8
Top-k fraction κ	0.1 (Equation (10))
Head initial values	$\tau_0 = 1.0$, $\beta_0 = 0.3$, $\lambda_0 = 0.5$ (attention-max) or 0.05 (dual-stream)
Focus penalty λ_{focus}	0.01; 0 for the MILLET baselines, since the term is ours
Seeds	3 for the models of Table 28; 1 for the rest
Hardware	GPU nodes of a shared academic computing cluster

The wide encoder uses the same setting, with $d = 128$ and 6 blocks. WebTraffic is generated with the released default settings, and each series is standardized on its own; nothing else is done to the data.

D.4 Interpretation Metrics

Section 4 describes the two interpretation metrics in words: AOPCR removes the time points in the order of their estimated importance and measures how much the score of the predicted class drops, and NDCG@n compares the ranking of the time points with the ground truth. Here we give their exact definitions.

AOPCR. Let $\hat{Y}_i^{\,c}(X_i)$ be the logit of the predicted class c on the unchanged time series, and let $X_i^{(m)}$ be the same series in which the m time points that the model itself ranks as the most important for class c are replaced by the mean of that series. AOPCR adds up the drop of the logit along this order,

$$\text{AOPCR} = \frac{1}{M+1} \sum_{m=0}^{M} \left(\hat{Y}_i^{\,c}(X_i) - \hat{Y}_i^{\,c}\big(X_i^{(m)}\big) \right), \tag{22}$$

following Samek et al. (2017) as adapted to time series by Early et al. (2024), with M set to $10\,\%$ of the series length in steps of $1\,\%$. A higher value is better. Note that Equation (22) is a difference of logits, and logits have no fixed scale. This causes the comparison problem of Section 4.1: two models whose logits lie on different scales give AOPCR values on different scales, however good or bad their ranking of the time points is.

NDCG@n. When the ground truth of each time point is available, as in WebTraffic, we rank the t time points of the series by the importance $\hat{I}_i^{\,j,c}$ that the model gives them for the predicted class c, and score this ranking with the normalized discounted cumulative gain (Järvelin & Kekäläinen, 2002) at the cut-off n, where n is the number of time points that the generator marked as carrying the class signature,

$$\text{DCG@}n = \sum_{p=1}^{n} \frac{r_p}{\log_2(p+1)}, \qquad \text{NDCG@}n = \frac{\text{DCG@}n}{\text{IDCG@}n} \in [0, 1], \tag{23}$$

where $r_p \in \{0, 1\}$ tells whether the time point placed at rank p by the model belongs to the true signature window, and IDCG@n is the same sum for the best possible ranking of the same series. Since the score is divided by the best possible score, NDCG@n lies in $[0, 1]$ and can be compared between models, datasets and papers. This is why the interpretation claims of this paper rest on NDCG@n and not on AOPCR.

D.5 Full Hyperparameters

Table 16 lists every hyperparameter, with one row per hyperparameter and one column per model, read directly from the configuration files. The two SEA-Net columns and the MILLET column with our recipe share the same training block, so any difference between them is in the architecture. Table 17, set on landscape pages, then gives the number of parameters of every encoder $\times$ pooling head pair that was trained, measured on WebTraffic with its 10 classes, together with the change relative to the same encoder with the class-wise conjunctive head.

Table 16: Every hyperparameter, read from the configuration files under `configs/models/`. The two SEA-Net columns and MILLET (our recipe) share one training block, so any difference between them is the architecture. – = not applicable (the InceptionTime backbone is the code of MILLET with its published defaults, so it has no settings here).

Parameter	**SEA-Net bottleneck + Top-k (the paper's model)**	**SEA-Net gated (mean) + Top-k**	**MILLET, our recipe**
Training			
max epochs	400	400	400
early-stopping patience	60	60	60
batch size (max)	16	16	16
learning rate	0.00125	0.00125	0.00125
weight decay	0.00012	0.00012	0.00012
label smoothing	0.13	0.13	0.13
λ_{focus} (attention entropy)	0.01	0.01	0.0
min. train size for a val. split	100	100	100
validation fraction	0.2	0.2	0.2
optimizer	adam	adam	adam
Encoder			
type (registry name)	`sea_mstcn_sep_bottleneck`	`sea_mstcn_sep_gated`	`mil_inceptiontime`
width d	64	64	–
blocks	4	4	–
dropout	0.2	0.2	–
dilation cap	16	16	–
kernel sizes	{5, 11, 23}	{5, 11, 23}	–
bottleneck ratio (d/r)	4	–	–
gate summary statistic	–	mean	–
Pooling head			
type (registry name)	`sea_topk_conjunctive`	`sea_topk_conjunctive`	`mil_conjunctive`
attention width a	8	8	8
dropout	0.2	0.2	0.1
positional encoding	True	True	True
Top-k fraction κ	0.1	0.1	–

Table 17: Number of parameters of every encoder × pooling head pair that was trained, measured on WebTraffic (10 classes; the size of a head grows with the number of classes). The percentages give the change relative to the same encoder with class-wise conjunctive pooling, in the manner of Table A.13 of MILLET: every attention head costs well under 1 % of the encoder, so the size of a model is set by its encoder. The bottom rows are the baseline encoders with the conjunctive head of MILLET.

#	Encoder \ head	class-wise conj.	Top-k conj.	softmax conj.	adaptive cw.	attn-max	gated attn.	dual-stream	conj.^{M}	additive^{M}	attention^{M}
1	bottleneck (shallow)	–	21,548	–	–	–	–	–	–	–	–
2	multi-scale pyramid (bottleneck base, over channels)	–	28,441	–	–	–	–	–	–	–	–
3	bottleneck	41,324	41,324 (+0.00%)	41,325 (+0.00%)	41,334 (+0.02%)	41,334 (+0.02%)	41,844 (+1.26%)	41,845 (+1.26%)	–	–	–
4	multi-scale pyramid (bottleneck base)	–	42,365	–	–	–	–	–	–	–	–
5	multi-scale channels (bottleneck base, lean)	–	43,576	–	–	–	–	–	–	–	–
6	multi-scale channels (bottleneck base)	–	47,176	–	–	–	–	–	–	–	–
7	base	57,580	57,580 (+0.00%)	57,581 (+0.00%)	57,590 (+0.02%)	57,590 (+0.02%)	58,100 (+0.90%)	58,101 (+0.90%)	–	–	–
8	input-gated	58,092	58,092 (+0.00%)	58,093 (+0.00%)	58,102 (+0.02%)	58,102 (+0.02%)	–	–	–	–	–
9	gated	61,740	–	–	–	–	–	–	–	–	58,100 (-5.90%)
10	gated (mean)	61,740	61,740 (+0.00%)	61,741 (+0.00%)	61,750 (+0.02%)	61,750 (+0.02%)	–	–	–	–	–
11	gated (max)	61,740	61,740 (+0.00%)	61,741 (+0.00%)	61,750 (+0.02%)	61,750 (+0.02%)	–	–	–	–	–
12	gated (last)	61,740	61,740 (+0.00%)	61,741 (+0.00%)	61,750 (+0.02%)	61,750 (+0.02%)	62,260 (+0.84%)	62,261 (+0.84%)	–	–	–
13	multi-scale pyramid (gated base)	–	62,781	–	–	–	–	–	–	–	–
14	reconstruction-residual	62,925	62,925 (+0.00%)	62,926 (+0.00%)	62,935 (+0.02%)	62,935 (+0.02%)	–	–	–	–	–
15	spike/trend	67,020	67,020 (+0.00%)	67,021 (+0.00%)	67,030 (+0.01%)	67,030 (+0.01%)	67,540 (+0.78%)	67,541 (+0.78%)	–	–	–

continued on the next page

Table 17 (continued)

#	Encoder \ head	class-wise conj.	Top-k conj.	softmax conj.	adaptive cw.	attn-max	gated attn.	dual-stream	conj.^{M}	additive^{M}	attention^{M}
16	multi-scale channels (gated base)	–	67,592	–	–	–	–	–	–	–	–
17	multi-scale channels (input-gated base)	–	69,768	–	–	–	–	–	–	–	–
18	base (wide, $d = 128$)	269,164	–	269,165 (+0.00%)	269,174 (+0.00%)	–	–	–	269,083 (-0.03%)	269,083 (-0.03%)	–
19	InceptionTime (MILLET)	–	–	–	–	–	–	–	423,707	–	–
20	ResNet	–	–	–	–	–	–	–	506,331	–	–
21	FCN	–	–	–	–	–	–	–	267,035	–	–

E Complete Results

Section 4.1 reports the main results in short. On WebTraffic, our SEA-Net bottleneck + Top-k model is more accurate than MILLET and gives a better NDCG@n with about ten times fewer parameters. On the 85 UCR datasets it stays slightly behind, and this gap comes from the reduced capacity and not from the design. The main text also answers RQ4 (the bottleneck) and RQ5 (Top-k pooling), and shows that AOPCR can only be compared within one training recipe.

This appendix gives the complete results behind these statements, in the same order as the main text. Appendix E.1 gives every model trained on WebTraffic, Appendix E.2 the cost of the models, Appendix E.3 the ablation studies, Appendix E.4 the results of every seed, Appendix E.5 the complete UCR results, and Appendix E.6 compares our MILLET baseline with the published numbers of MILLET (Early et al., 2024).

E.1 Complete WebTraffic Results

Figure 15 first compares the explanations of MILLET and of our model on four WebTraffic classes. It shows, on single series, the difference between the two interpretation metrics: MILLET wins AOPCR on three of the four classes, while our heatmaps are the more concentrated.

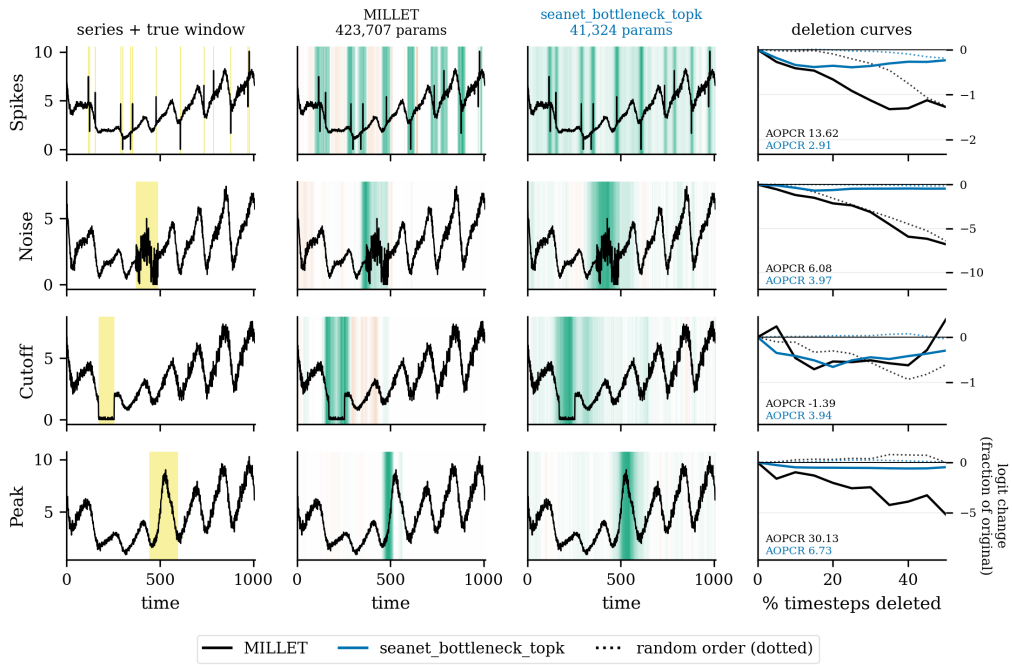


Figure 15: Head-to-head comparison of the explanations of MILLET (published recipe) and of our SEA-Net bottleneck + Top-k model on four WebTraffic classes. The panels show, left to right, the series with its true signature window, the interpretation heatmap for the true class, and the perturbation curves under each model's own deletion order (solid) against a random one (dotted). The vertical axis is the change as a fraction of the original logit, because the two models produce logits on very different scales (Appendix E.6). The AOPCR of each panel is the area between the two curves; MILLET wins it on three of the four classes while our heatmaps are the more concentrated — the difference between the two metrics is analysed in Section 4.1.

The tables below give every trained model. Table 21 is the complete WebTraffic leaderboard of the 71-model sweep, ranked by accuracy, where one row is one encoder + pooling head pair; the rows are grouped into five accuracy bands, so that the table can be read one band at a time. Table 18 is a later round in which every new head was trained on the bottleneck encoder, so the row of our model is the one to beat there. Table 19 and Table 20 give the same sweep as a grid, with one encoder per row and one pooling head per column, for the accuracy and for AOPCR / NDCG@n respectively; a dash means that the pair was not trained. These three tables are wide and are set on landscape pages.

Table 18: The 2026-09 pooling-head round on WebTraffic (results/top_results, one seed each): every new head is trained on the bottleneck encoder (one on the gated encoder), so row 1 (the model of the paper, trained again) is the number to beat. The MILLET row is the baseline retrained with our recipe. *Epochs run / best* gives the epoch at which early stopping fired and the epoch whose weights were kept; the voting heads that fail to fit are visible in their NLL.

#	Encoder	Pooling head	Acc.	AUROC	NLL	AOPCR	NDCG@n	Params	Epochs run / best	Train (s)
1	bottleneck	Top-k conjunctive (paper model, re-run)	0.942	0.996	0.294	2.951	0.786	41,324	180 / 120	54
2	bottleneck	Top-k conjunctive, thresholded	0.930	0.996	0.330	2.603	0.744	41,324	172 / 112	52
3	bottleneck	softmax conjunctive, per-class τ	0.914	0.991	0.377	2.076	0.732	41,334	151 / 91	65
4	bottleneck	deviation-max (MLP form)	0.912	0.990	0.369	2.394	0.742	42,544	164 / 104	276
5	bottleneck	DSMIL (batched rewrite)	0.900	0.991	0.477	1.462	0.667	71,956	379 / 319	611
6	InceptionTime	conjunctive (MILLET, our recipe)	0.890	0.985	0.531	2.120	0.692	423,707	184 / 124	443
7	bottleneck	DSMIL (original code)	0.888	0.991	0.488	1.242	0.536	71,956	319 / 259	2152
8	bottleneck	Top-k voting, thresholded	0.886	0.987	0.468	2.195	0.604	41,325	247 / 187	71
9	bottleneck	deviation-max (dual form)	0.878	0.988	0.451	2.356	0.723	41,944	176 / 116	69
10	bottleneck	Top-k voting	0.870	0.983	0.516	2.212	0.601	41,325	218 / 158	61
11	bottleneck	adaptive class-wise + MLP	0.836	0.982	0.653	1.528	0.618	41,375	195 / 135	78
12	bottleneck	DSMIL (batched rewrite, no pos.)	0.832	0.976	0.632	1.312	0.639	71,956	205 / 145	318
13	gated (mean)	DSMIL (original code)	0.810	0.971	0.692	0.996	0.341	92,372	126 / 66	1043
14	bottleneck	Top-k voting, hard	0.672	0.902	1.173	1.770	0.697	41,325	168 / 108	50
15	bottleneck	simple voting	0.100	0.531	4.511	0.150	0.222	41,324	73 / 13	22

Table 19: WebTraffic accuracy for every encoder × pooling head pair that was trained (71-model sweep; the κ / λ ablation copies are left out). – = pair not trained; a superscript count means that several configurations share the cell and the most accurate one is shown. ^{M} = a MILLET head. The best value of each column is in bold.

#	Encoder \ head	class-wise conj.	Top-k conj.	softmax conj.	adaptive cw.	attn-max	gated attn.	dual-stream	conj.^{M}	additive^{M}	attention^{M}
1	gated (mean)	0.902	**0.955**	0.912	0.860	0.832	–	–	–	–	–
2	base (wide, $d = 128$)	**0.946**	–	0.880	0.918	–	–	–	**0.954**	**0.942**	–
3	gated (max)	0.944	0.952	0.914	0.906	0.886	–	–	–	–	–
4	spike/trend	0.904	0.950	0.878	0.934	0.854	**0.930**	0.556	–	–	–
5	multi-scale channels (input-gated base)	–	0.948	–	–	–	–	–	–	–	–
6	multi-scale channels (gated base)	–	0.948	–	–	–	–	–	–	–	–
7	reconstruction-residual	0.910	0.940	0.932	**0.946**	**0.892**	–	–	–	–	–
8	input-gated	0.914	0.946	0.894	0.905	0.858	–	–	–	–	–
9	gated (last)	0.942	0.942	0.920	0.938	0.854	0.918	0.736	–	–	–
10	bottleneck	0.902	0.931	**0.940**	0.868	0.816	**0.930**	0.824	–	–	–
11	base	0.916	0.932	0.890	0.930	0.714	0.894	**0.860**	–	–	–
12	gated	0.930	–	–	–	–	–	–	–	–	**0.888**
13	multi-scale channels (bottleneck base)	–	0.926	–	–	–	–	–	–	–	–
14	multi-scale channels (bottleneck base, lean)	–	0.924	–	–	–	–	–	–	–	–
15	InceptionTime	–	–	–	–	–	–	–	0.887	–	–
16	bottleneck (shallow)	–	0.846	–	–	–	–	–	–	–	–
17	multi-scale pyramid (gated base)	–	0.832	–	–	–	–	–	–	–	–
18	multi-scale pyramid (bottleneck base)	–	0.786	–	–	–	–	–	–	–	–
19	ResNet	–	–	–	–	–	–	–	0.772	–	–
20	FCN	–	–	–	–	–	–	–	0.742	–	–
21	multi-scale pyramid (bottleneck base, over channels)	–	0.696	–	–	–	–	–	–	–	–

Table 20: WebTraffic AOPCR / NDCG@n for every encoder $\times$ pooling head pair that was trained (71-model sweep; the κ / λ ablation copies are left out). – = pair not trained; a superscript count means that several configurations share the cell and the most accurate one is shown. ^{M} = a MILLET head. AOPCR is unnormalised (the same recipe is used everywhere here).

#	Encoder \ head	class-wise conj.	Top-k conj.	softmax conj.	adaptive cw.	attn-max	gated attn.	dual-stream	conj.^{M}	additive^{M}	attention^{M}
1	gated (mean)	2.49 / 0.714	2.23 / 0.750	2.09 / 0.765	2.24 / 0.731	1.67 / 0.704	–	–	–	–	–
2	base (wide, $d = 128$)	1.72 / 0.686	–	1.02 / 0.605	1.61 / 0.619	–	–	–	1.50 / 0.698	1.58 / 0.729	–
3	gated (max)	1.89 / 0.592	2.27 / 0.719	1.88 / 0.776	2.07 / 0.729	1.12 / 0.575	–	–	–	–	–
4	spike/trend	2.15 / 0.699	2.32 / 0.756	1.43 / 0.665	1.91 / 0.747	1.65 / 0.644	1.59 / 0.721	1.64 / 0.510	–	–	–
5	multi-scale channels (input-gated base)	–	2.32 / 0.757	–	–	–	–	–	–	–	–
6	multi-scale channels (gated base)	–	2.22 / 0.717	–	–	–	–	–	–	–	–
7	reconstruction-residual	2.50 / 0.680	2.51 / 0.698	1.64 / 0.723	2.60 / 0.768	2.32 / 0.725	–	–	–	–	–
8	input-gated	2.48 / 0.715	2.80 / 0.748	1.64 / 0.696	2.65 / 0.732	2.15 / 0.700	–	–	–	–	–
9	gated (last)	2.52 / 0.742	2.31 / 0.748	1.99 / 0.738	2.39 / 0.793	1.78 / 0.668	1.35 / 0.757	1.95 / 0.637	–	–	–
10	bottleneck	2.57 / 0.733	2.50 / 0.765	1.68 / 0.740	1.97 / 0.680	1.74 / 0.610	2.23 / 0.741	2.04 / 0.652	–	–	–
11	base	2.48 / 0.719	2.69 / 0.778	1.83 / 0.705	2.45 / 0.793	1.63 / 0.532	2.30 / 0.714	2.40 / 0.745	–	–	–
12	gated	2.41 / 0.637	–	–	–	–	–	–	–	–	2.40 / 0.710
13	multi-scale channels (bottleneck base)	–	2.24 / 0.777	–	–	–	–	–	–	–	–
14	multi-scale channels (bottleneck base, lean)	–	2.76 / 0.735	–	–	–	–	–	–	–	–
15	InceptionTime	–	–	–	–	–	–	–	2.57 / 0.677	–	–
16	bottleneck (shallow)	–	2.82 / 0.624	–	–	–	–	–	–	–	–
17	multi-scale pyramid (gated base)	–	0.98 / 0.591	–	–	–	–	–	–	–	–
18	multi-scale pyramid (bottleneck base)	–	1.03 / 0.569	–	–	–	–	–	–	–	–
19	ResNet	–	–	–	–	–	–	–	2.95 / 0.554	–	–
20	FCN	–	–	–	–	–	–	–	3.83 / 0.533	–	–
21	multi-scale pyramid (bottleneck base, over channels)	–	2.17 / 0.653	–	–	–	–	–	–	–	–

Table 21: The complete WebTraffic leaderboard of the 71-model sweep (results/old_results), ranked by accuracy and grouped into accuracy bands. * marks the mean over three seeds (seeds 0–2, the runs of Table 28); every other row is a single run with seed 0, so a comparison between a * row and an unmarked row sets a three-run mean against one run. *UCR-85 acc.* is the mean accuracy over the 85 UCR datasets for the models that completed the archive, and – for the models that were only screened on WebTraffic. *Encoder* drops the SEA-Net prefix (MS = multi-scale); *wide* is the $d = 128$ encoder; ^{M} = a MILLET head. AUROC is given in the csv file next to this table.

#	Encoder	Pooling head	Acc.	NLL	AOPCR	NDCG@n	Params	UCR-85 acc.
Accuracy ≥ 0.94								
1	gated (mean)	Top-k conj.*	0.955	0.254	2.225	0.750	61,740	0.8083
2	base (wide, $d = 128$)	conj.^{M}	0.954	0.262	1.502	0.698	269,083	0.8238
3	gated (max)	Top-k conj.	0.952	0.308	2.268	0.719	61,740	0.8108
4	bottleneck	Top-k conj. (λ_{focus}=0)	0.950	0.263	2.303	0.765	41,324	–
5	spike/trend	Top-k conj.	0.950	0.288	2.316	0.756	67,020	0.8089
6	MS channels (input-gated base)	Top-k conj.	0.948	0.318	2.316	0.757	69,768	–
7	MS channels (gated base)	Top-k conj.	0.948	0.289	2.223	0.717	67,592	–
8	base (wide, $d = 128$)	class-wise conj.	0.946	0.292	1.722	0.686	269,164	0.8292
9	input-gated	Top-k conj.	0.946	0.261	2.803	0.748	58,092	0.8131
10	reconstruction-residual	adaptive cw.	0.946	0.296	2.601	0.768	62,935	0.8140
11	gated (max)	class-wise conj.	0.944	0.306	1.889	0.592	61,740	0.8237
12	gated (last)	class-wise conj.	0.942	0.327	2.520	0.742	61,740	0.7888
13	gated (last)	Top-k conj.	0.942	0.328	2.312	0.748	61,740	0.8015
14	base (wide, $d = 128$)	additive^{M}	0.942	0.260	1.579	0.729	269,083	0.8298
15	bottleneck	Top-k conj. (κ=0.25)	0.940	0.277	2.648	0.733	41,324	–
16	reconstruction-residual	Top-k conj.	0.940	0.321	2.505	0.698	62,925	0.8063
17	bottleneck	softmax conj.	0.940	0.345	1.685	0.740	41,325	0.8106
Accuracy in $[0.92, 0.94)$								
18	gated (last)	adaptive cw.	0.938	0.289	2.387	0.793	61,750	0.8029
19	spike/trend	adaptive cw.	0.934	0.301	1.913	0.747	67,030	0.8010
20	base	Top-k conj.	0.932	0.306	2.691	0.778	57,580	0.8093
21	reconstruction-residual	softmax conj.	0.932	0.341	1.642	0.723	62,926	–
22	bottleneck	Top-k conj.*	0.931	0.312	2.498	0.765	41,324	0.8097
23	gated	class-wise conj.	0.930	0.359	2.408	0.637	61,740	0.7900
24	base	adaptive cw.	0.930	0.336	2.448	0.793	57,590	–
25	bottleneck	gated attn.	0.930	0.313	2.232	0.741	41,844	0.8146
26	spike/trend	gated attn.	0.930	0.350	1.593	0.721	67,540	0.8039
27	MS channels (bottleneck base)	Top-k conj.	0.926	0.357	2.238	0.777	47,176	–
28	MS channels (bottleneck base, lean)	Top-k conj.	0.924	0.389	2.762	0.735	43,576	–
29	gated (last)	softmax conj.	0.920	0.381	1.990	0.738	61,741	–
Accuracy in $[0.90, 0.92)$								

continued on the next page

Table 21 (continued)

#	Encoder	Pooling head	Acc.	NLL	AOPCR	NDCG@n	Params	UCR-85 acc.
30	base (wide, $d = 128$)	adaptive cw.	0.918	0.360	1.608	0.619	269,174	0.8134
31	gated (last)	gated attn.	0.918	0.365	1.354	0.757	62,260	0.8048
32	base	class-wise conj.	0.916	0.383	2.478	0.719	57,580	–
33	gated (max)	softmax conj.	0.914	0.372	1.884	0.776	61,741	–
34	input-gated	class-wise conj.	0.914	0.398	2.481	0.715	58,092	–
35	gated (mean)	softmax conj.	0.912	0.368	2.093	0.765	61,741	–
36	reconstruction-residual	class-wise conj.	0.910	0.417	2.504	0.680	62,925	–
37	bottleneck	Top-k conj. (κ=1.00)	0.908	0.395	2.436	0.722	41,324	–
38	gated (max)	adaptive cw.	0.906	0.444	2.071	0.729	61,750	–
39	input-gated	adaptive cw.*	0.905	0.390	2.651	0.732	58,102	0.8153
40	spike/trend	class-wise conj.	0.904	0.395	2.155	0.699	67,020	–
41	bottleneck	class-wise conj.	0.902	0.395	2.572	0.733	41,324	–
42	gated (mean)	class-wise conj.	0.902	0.393	2.493	0.714	61,740	–
Accuracy in $[0.85, 0.90)$								
43	input-gated	softmax conj.	0.894	0.454	1.636	0.696	58,093	–
44	base	gated attn.	0.894	0.403	2.296	0.714	58,100	–
45	reconstruction-residual	attn-max	0.892	0.499	2.324	0.725	62,935	–
46	base	softmax conj.	0.890	0.419	1.832	0.705	57,581	–
47	gated	attention^{M}	0.888	0.426	2.403	0.710	58,100	–
48	bottleneck	Top-k conj. (κ=0.05)	0.888	0.401	2.288	0.715	41,324	–
49	InceptionTime	conj.$^{\mathrm{M}*}$	0.887	0.507	2.569	0.677	423,707	0.8274
50	gated (max)	attn-max	0.886	0.588	1.124	0.575	61,750	–
51	bottleneck	Top-k conj. (κ=0.50)	0.882	0.418	3.114	0.732	41,324	–
52	base (wide, $d = 128$)	softmax conj.	0.880	0.455	1.015	0.605	269,165	0.8131
53	spike/trend	softmax conj.	0.878	0.488	1.435	0.665	67,021	–
54	bottleneck	adaptive cw.	0.868	0.559	1.969	0.680	41,334	–
55	base	dual-stream	0.860	0.498	2.399	0.745	58,101	–
56	gated (mean)	adaptive cw.	0.860	0.549	2.238	0.731	61,750	–
57	input-gated	attn-max	0.858	0.625	2.153	0.700	58,102	–
58	spike/trend	attn-max	0.854	0.637	1.646	0.644	67,030	–
59	gated (last)	attn-max	0.854	0.598	1.779	0.668	61,750	–
Accuracy < 0.85								
60	bottleneck (shallow)	Top-k conj.	0.846	0.579	2.817	0.624	21,548	–
61	gated (mean)	attn-max	0.832	0.665	1.672	0.704	61,750	–
62	MS pyramid (gated base)	Top-k conj.	0.832	0.602	0.980	0.591	62,781	–
63	bottleneck	dual-stream	0.824	0.638	2.037	0.652	41,845	–
64	bottleneck	attn-max	0.816	0.732	1.737	0.610	41,334	–
65	MS pyramid (bottleneck base)	Top-k conj.	0.786	0.759	1.035	0.569	42,365	–
66	ResNet	conj.^{M}	0.772	0.824	2.952	0.554	506,331	0.8146
67	FCN	conj.^{M}	0.742	0.910	3.826	0.533	267,035	0.8141

continued on the next page

Table 22: Cost on the WebTraffic input (length 1,008, 10 classes), one GPU, batch 32. *Params*× and *FLOPs*× are size and compute relative to the re-trained MILLET baseline; column definitions in Appendix D.4; lower is better throughout.

Model	Params	Size	FLOPs	Latency	Params×	FLOPs×
SEA-Net (ours)						
bottleneck + Top-k conj.	**41,324**	**1.43**	**76.7**	0.131	0.10×	0.09×
bottleneck + Top-k conj. ($\lambda_{\text{focus}} = 0$)	**41,324**	**1.43**	**76.7**	0.130	0.10×	0.09×
input-gated + Top-k conj.	58,092	1.49	110.6	0.130	0.14×	0.13×
gated (mean) + Top-k conj.	61,740	1.50	109.7	0.128	0.15×	0.13×
spike/trend + Top-k conj.	67,020	1.53	120.1	0.132	0.16×	0.14×
bottleneck + Top-k conj., 2026-09 re-run	**41,324**	**1.43**	**76.7**	0.131	0.10×	0.09×
Baselines						
MILLET (InceptionTime + conjunctive), our recipe	423,707	4.11	847.6	0.203	1.00×	1.00×
ResNet + conjunctive	506,331	4.42	1013.2	0.139	1.20×	1.20×
FCN + conjunctive	267,035	3.47	535.2	**0.068**	0.63×	0.63×

Table 21 (continued)

#	Encoder	Pooling head	Acc.	NLL	AOPCR	NDCG@n	Params	UCR-85 acc.
68	gated (last)	dual-stream	0.736	0.804	1.947	0.637	62,261	–
69	base	attn-max	0.714	0.855	1.630	0.532	57,590	–
70	MS pyramid (bottleneck base, over channels)	Top-k conj.	0.696	0.992	2.167	0.653	28,441	–
71	spike/trend	dual-stream	0.556	1.260	1.643	0.510	67,541	–

E.2 Model Cost

The main text states that our model is $10.3\times$ smaller and needs $11.1\times$ less computation than MILLET. Table 22 gives the full cost and separates what improved from what did not. At 32-bit precision the weights go from about 1.6 MB to 161 KB, which moves the model from above to well below the roughly 1 MB flash budget of the micro-controller class described by Lin et al. (2020) and Banbury et al. (2021b), and standard *int8* quantization (Jacob et al., 2018) would reduce it to about 40 KB. In this sense, the model becomes a candidate for such a device. The other two columns do not improve. Latency improves by only $1.6\times$ even though the FLOPs fall $11.1\times$, because depthwise convolutions are limited by memory speed, and a GPU at batch 32 is the wrong tool for a claim about small hardware; the 269 K wide model is in fact the *slowest* row. Peak memory is 179.0 MB against 112.3 MB for the baseline, which is worse, and this comes from the design: the encoder keeps the length unchanged, so the activations have the shape (d, t) at every stage, however few weights produce them, and Equation (3) reduces the weights, not the activations. On micro-controllers, the SRAM needed for the activations is the limit at least as often as the flash needed for the weights, so this is a real limitation of the design and not a measurement error (Section 5).

Figure 16 draws the compute column of Table 22 as a function of the size of the problem. The left panel varies the length of the series at 10 classes, and the right panel varies the number of classes at length 463, and the two dashed lines mark the mean length and the mean number of classes over the 85 UCR datasets. The cost of both models grows linearly with the length and hardly depends on the number of classes, and our model needs about ten times fewer operations at every length.

Table 23 completes the picture on the pooling side, by counting the real multiplications of each head as a function of t, C and d. No head adds a term that grows faster than those of `Conjunctive` pooling, so the saving of Table 22 comes from the encoder and is not lost again in the head.

E.3 Ablation Studies

The main text answers RQ4 and RQ5 with matched pairs of models, i.e. two models that differ in one part only. The ablations below give the wider view: the effect of the pooling head and of the encoder

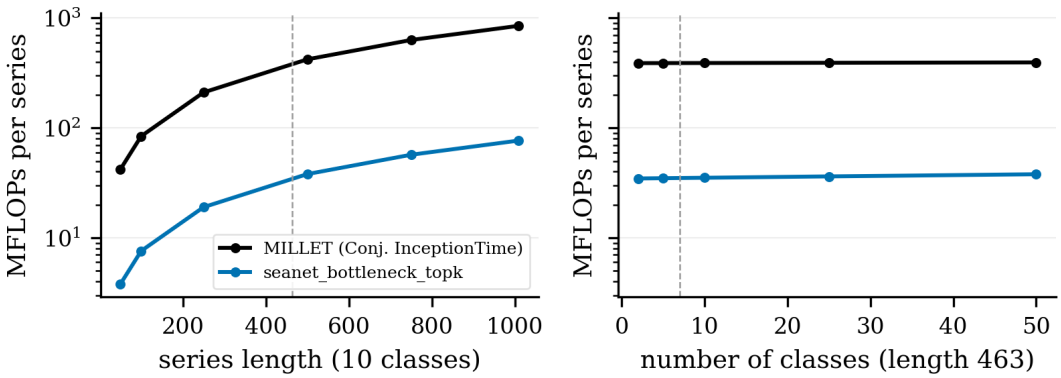


Figure 16: Compute cost of one prediction, in millions of floating point operations, for our SEA-Net bottleneck + Top-k model and for the MILLET baseline (`InceptionTime` + `Conjunctive`). *Left:* as the length of the series grows, at 10 classes. *Right:* as the number of classes grows, at length 463. The dashed lines mark the mean series length and the mean number of classes over the 85 UCR datasets.

Table 23: Time complexity of the pooling heads as the number of real multiplications, counted from the code (matrix products only, as in Table A.15 of MILLET; additions, comparisons, the exponential of a softmax and the gather of the critical time point are not counted). t = series length, c = classes, d = embedding size, a = attention width, $k = \lceil \kappa t \rceil$. The last column evaluates the formula for WebTraffic ($t = 1008$, $c = 10$, $a = 8$, $\kappa = 0.1$, so $k = 101$) with the own d of each head: the SEA-Net heads read $d = 64$ embeddings and the MILLET head $d = 128$, so every SEA-Net head is cheaper than the baseline head in absolute terms, although its formula has more terms. ^{M} = MILLET.

#	Pooling head	Real multiplications	Scale w.r.t. t	Scale w.r.t. c	At WebTraffic (M)
1	conjunctive^{M}	$dtc + dta + (a + 2c)\,t$	$dc + da + a + 2c$	$dt + 2t$	2.351 ($d = 128$)
2	class-wise conjunctive	$dtc + dta + (ac + 2c)\,t$	$dc + da + ac + 2c$	$dt + at + 2t$	1.262 ($d = 64$)
3	Top-k conjunctive	$dtc + dta + (ac + c)\,t + ck$	$dc + da + ac + c + c\kappa$	$dt + at + t + k$	1.253 ($d = 64$)
4	softmax conjunctive	$dtc + dta + (ac + 3c)\,t$	$dc + da + ac + 3c$	$dt + at + 3t$	1.272 ($d = 64$)
5	adaptive class-wise	$dtc + dta + (ac + 4c)\,t$	$dc + da + ac + 4c$	$dt + at + 4t$	1.282 ($d = 64$)
6	attention-max	$dtc + dta + (ac + 2c)\,t + 2c$	$dc + da + ac + 2c$	$dt + at + 2t + 2$	1.262 ($d = 64$)
7	gated attention	$dtc + 2dta + (ac + a + 2c)\,t$	$dc + 2da + ac + a + 2c$	$dt + at + 2t$	1.786 ($d = 64$)
8	dual-stream conjunctive	$dtc + 2dta + (2ac + 4c)\,t + 2c$	$dc + 2da + 2ac + 4c$	$dt + 2at + 4t + 2$	1.879 ($d = 64$)

averaged over the whole sweep (A and B), the effect of the Top-k fraction κ (C), and the effect of the attention entropy term (D). A short look at the wrong predictions of the two models closes the subsection.

A. Effect of the pooling head, averaged over the encoders. In Table 24, one row is one pooling head, and the columns are the accuracy, AOPCR and NDCG@n averaged over every encoder that the head was paired with, where n is the number of pairings. Averaged this way, the pooling head alone changes the WebTraffic accuracy by almost 18 points, which is why we treat the choice of the head as a design decision and not as a detail.

B. Effect of the encoder, averaged over the pooling heads. Table 25 is the same table for the encoders, i.e. one row is one encoder, and the values are averaged over every head that it was paired with. Its spread, from 0.937 down to 0.771, is close to the spread of the pooling table above, so the

Table 24: Effect of the pooling head: WebTraffic accuracy, AOPCR and NDCG@n averaged over every encoder that each head was paired with in the 71-model sweep (n = number of pairings; the κ / λ ablation copies are excluded, so that no pairing is counted twice). The last column is the best single pairing of that head. A head paired once ($n = 1$) is not comparable with one averaged over ten.

#	Pooling head	Mean acc.	Mean AOPCR	Mean NDCG@n	n	Best pairing (acc.)
SEA-Net heads						
1	class-wise conjunctive	0.921	2.322	0.692	10	0.946 (`seanet_classwise`)
2	gated attention	0.918	1.869	**0.733**	4	0.930 (`seanet_bottleneck_gatedattn`)
3	adaptive class-wise	0.912	2.210	0.732	9	0.946 (`seanet_recon_adaptive`)
4	softmax conjunctive	0.907	1.690	0.712	9	0.940 (`seanet_bottleneck_softmax`)
5	Top-k conjunctive	0.903	2.260	0.712	16	0.955 (`seanet_gated_mean_topk`)
6	attention-max	0.838	1.758	0.645	8	0.892 (`seanet_recon_attnmax`)
7	dual-stream conjunctive	0.744	2.007	0.636	4	0.860 (`seanet_slim_dualstream`)
MILLET heads (reference; $n = 1$ rows are one pairing only)						
8	additive	**0.942**	1.579	0.729	1	0.942 (`seanet`)
9	attention	0.888	2.403	0.710	1	0.888 (`seanet_slim_gatedattn`)
10	conjunctive	0.839	**2.712**	0.615	4	0.954 (`seanet_conjunctive`)

two choices are about equally important. Two cautions apply to both tables. First, several groups contain very few models and not every pair was trained, so the grid of Table 19 is the more reliable view. Second, the mean of the bottleneck encoder (0.884) is lower than its best pairing suggests, because it was also paired with the weak heads.

C. Effect of the Top-k fraction κ. Table 26 is the same model trained five times with only the κ of Equation (10) changed, all of them on seed 0, and the column k gives how many time points are actually kept at each fraction. The selection is what helps: $\kappa = 1$, which by Property 2 is exactly the class-wise conjunctive pooling of Equation (8), scores 0.908, clearly below the 0.938–0.940 obtained at $\kappa = 0.1$ and $\kappa = 0.25$.

D. Effect of the attention entropy term. Table 27 is one matched pair, i.e. the same model with only λ_{focus} switched on and off, on seed 0. Switching the term off raises the accuracy from 0.888 to 0.950 and NDCG@n from 0.755 to 0.765, and lowers AOPCR from 2.409 to 2.303. Seed 0 is, however, the weakest of the three runs of the default model (0.888, 0.938 and 0.966; Table 28): against its three-run mean (0.931 ± 0.032 accuracy, 0.765 ± 0.013 NDCG@n), the run without the term lies within one standard deviation in accuracy and matches it exactly in NDCG@n. We therefore keep the value selected by the hyperparameter search and read this pair as showing no measurable effect of the term in either direction.

Wrong predictions. Finally, Figures 17 and 18 look at the wrong predictions of the two models on their hardest WebTraffic class, i.e. the class with the lowest accuracy. The errors are of two kinds: in the first, the model still points at the right window but names the wrong class, so the NDCG@n stays high; in the second, the signature is too weak or too close to the seasonal pattern, and the model never finds it.

E.4 Per-seed Results

The main text compares every difference with the seed-to-seed spread measured on WebTraffic, ± 0.016 for the accuracy and ± 0.013 for NDCG@n. Table 28 gives the runs behind these numbers:

Table 25: Effect of the encoder: WebTraffic accuracy, AOPCR and NDCG@n averaged over every pooling head that each encoder was paired with (n = number of pairings, with the same exclusions as in Table 24). The gated encoder has the largest n because it was paired with the most heads, which pulls its mean towards the weak heads; the last column shows what each encoder reaches with its best head.

#	Encoder	Mean acc.	Mean AOPCR	Mean NDCG@n	n	Best pairing (acc.)
SEA-Net encoders († = a wrapper around another encoder)						
1	SEA-Net multi-scale channels†	**0.936**	2.385	**0.747**	4	0.948 (`seanet_inputgate_mschan`)
2	SEA-Net reconstruction-residual	0.924	2.315	0.719	5	0.946 (`seanet_recon_adaptive`)
3	SEA-Net input-gated	0.903	2.345	0.718	5	0.946 (`seanet_inputgate_topk`)
4	SEA-Net gated	0.902	2.056	0.710	19	0.955 (`seanet_gated_mean_topk`)
5	SEA-Net base	0.898	1.933	0.693	12	0.954 (`seanet_conjunctive`)
6	SEA-Net bottleneck	0.882	2.193	0.693	8	0.940 (`seanet_bottleneck_softmax`)
7	SEA-Net spike/trend	0.858	1.814	0.677	7	0.950 (`seanet_spiketrend_topk`)
8	SEA-Net multi-scale pyramid†	0.771	1.394	0.604	3	0.832 (`seanet_gated_pyramid`)
Baseline encoders (each paired with conjunctive pooling only)						
9	InceptionTime	0.887	2.569	0.677	1	0.887 (`millet`)
10	ResNet	0.772	2.952	0.554	1	0.772 (`resnet`)
11	FCN	0.742	**3.826**	0.533	1	0.742 (`fcn`)

Table 26: Top-k fraction ablation on WebTraffic: the bottleneck + Top-k model trained five times with only κ changed (seed 0 everywhere, so the $\kappa = 0.10$ row is the seed-0 run of that model, and the last row is its mean over every run). $k = \lceil \kappa t \rceil$ is the number of time points that the series logit averages; $\kappa = 1$ recovers class-wise conjunctive pooling exactly. The best single-seed value of each column is in bold.

#	κ	k at $t = 1008$	Accuracy	NLL	AOPCR	NDCG@n
1	0.05	51	0.888	0.401	2.288	0.715
2	0.10 *(default)*	101	0.888	0.391	2.409	**0.755**
3	0.25	252	**0.940**	**0.277**	2.648	0.733
4	0.50	504	0.882	0.418	**3.114**	0.732
5	1.00 *(= class-wise conjunctive)*	1008	0.908	0.395	2.436	0.722
	0.10, mean $\pm$ std over 3 runs	101	0.931 ± 0.032	0.312 ± 0.063	2.498 ± 0.149	0.765 ± 0.013

Table 27: The attention-entropy term of the loss, switched on and off, on WebTraffic (seed 0): the bottleneck + Top-k model with only λ_{focus} changed. This is one matched pair on one seed, so it indicates a direction rather than an effect size.

λ_{focus}	Accuracy	NLL	AOPCR	NDCG@n
0.01 *(on, the default)*	0.888	0.391	**2.409**	0.755
0.00 *(off)*	**0.950**	**0.263**	2.303	**0.765**

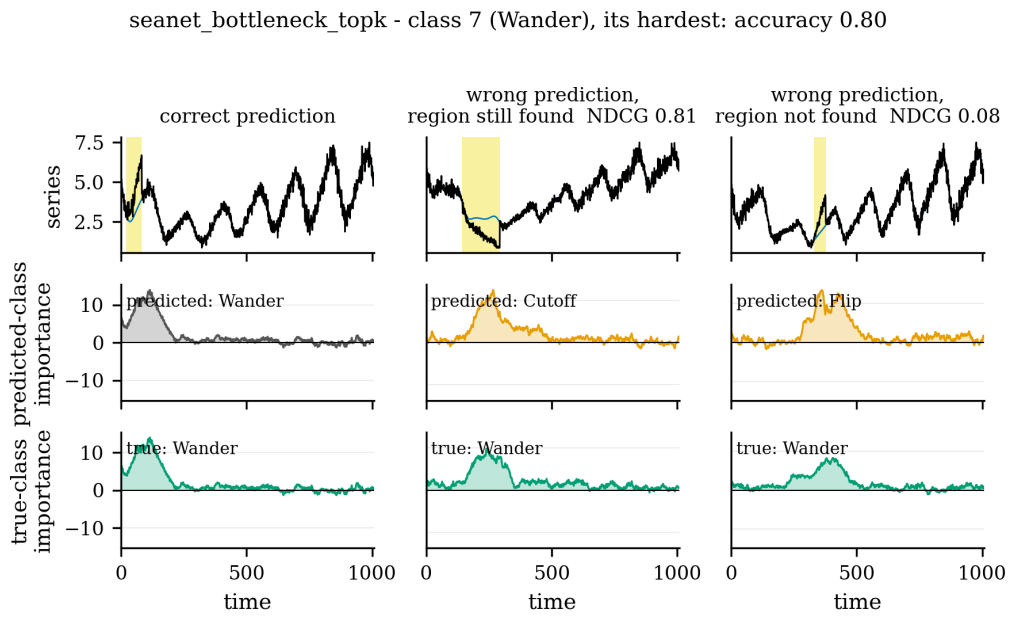


Figure 17: SEA-Net bottleneck + Top-k (ours). Wrong predictions on the hardest WebTraffic class, i.e. the class with the lowest accuracy. *Top:* the series with its true signature window in yellow. *Middle:* the interpretation for the class that the model predicted. *Bottom:* the interpretation for the true class, with its NDCG@n.

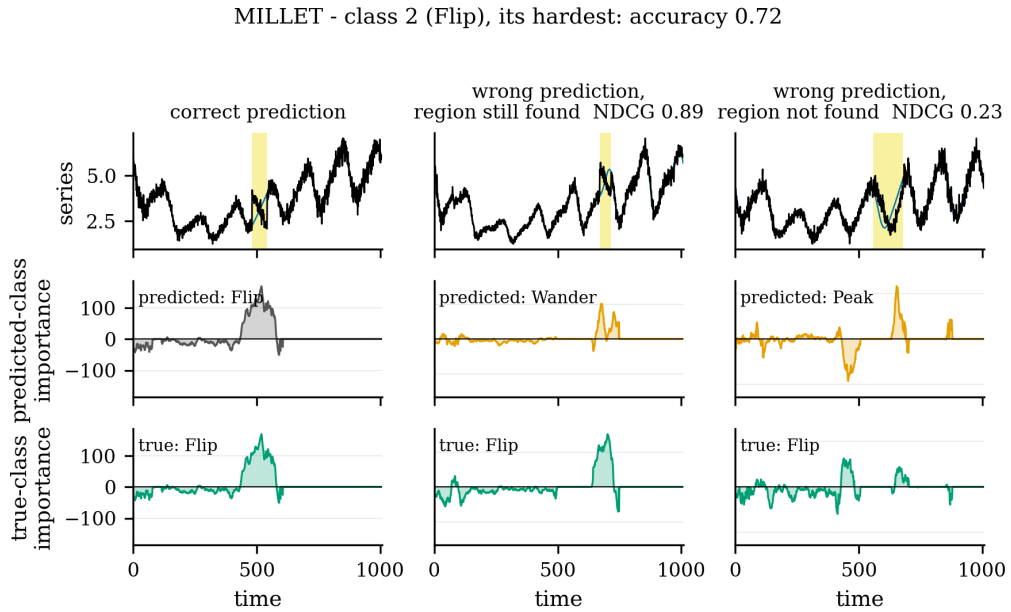


Figure 18: MILLET (published recipe). Wrong predictions on the hardest WebTraffic class, i.e. the class with the lowest accuracy. *Top:* the series with its true signature window in yellow. *Middle:* the interpretation for the class that the model predicted. *Bottom:* the interpretation for the true class, with its NDCG@n.

every WebTraffic run of the models that were trained more than once, with one row per model and per metric, one column per run, and the mean $\pm$ the standard deviation in the last column. These are the numbers behind every $\pm$ of the main tables and the noise level used throughout the paper: we never report a difference smaller than this spread as a result. For every model, the spread of AOPCR is far larger than that of the accuracy, which is why we never read an AOPCR difference of a few tenths as a win. Figure 19 asks the same question on the 85 UCR datasets.

Table 28: Every WebTraffic run of the models that were trained more than once, run by run (*archive* = the 71-model sweep in results/old_results, *2026-09* = results/top_results; a seed run in both folders is two different trainings). These are the numbers behind every $\pm$ in the main tables. The spread on AOPCR is far larger than on accuracy for every model, which is why AOPCR differences of a few tenths are not read as wins.

#	Model	Metric	Run 1	Run 2	Run 3	Run 4	Mean $\pm$ std
1	SEA-Net gated (mean) + Top-k conjunctive	accuracy	0.954	0.958	0.952	–	0.955 ± 0.002
		AOPCR	2.293	2.074	2.308	–	2.225 ± 0.107
		NDCG@n	0.772	0.749	0.728	–	0.750 ± 0.018
		run	*archive, seed 0*	*archive, seed 1*	*archive, seed 2*		
2	SEA-Net bottleneck + Top-k conjunctive	accuracy	0.888	0.938	0.966	–	0.931 ± 0.032
		AOPCR	2.409	2.708	2.378	–	2.498 ± 0.149
		NDCG@n	0.755	0.783	0.758	–	0.765 ± 0.013
		run	*archive, seed 0*	*archive, seed 1*	*archive, seed 2*		
3	SEA-Net input-gated + adaptive class-wise	accuracy	0.938	0.880	0.898	–	0.905 ± 0.024
		AOPCR	3.126	2.266	2.560	–	2.651 ± 0.357
		NDCG@n	0.777	0.692	0.728	–	0.732 ± 0.035
		run	*archive, seed 0*	*archive, seed 1*	*archive, seed 2*		
4	MILLET (InceptionTime + conjunctive), our recipe	accuracy	0.894	0.876	0.892	0.890	0.888 ± 0.007
		AOPCR	1.594	2.795	3.319	2.120	2.457 ± 0.655
		NDCG@n	0.671	0.695	0.665	0.692	0.681 ± 0.013
		run	*archive, seed 0*	*archive, seed 1*	*archive, seed 2*	*2026-09, seed 0*	

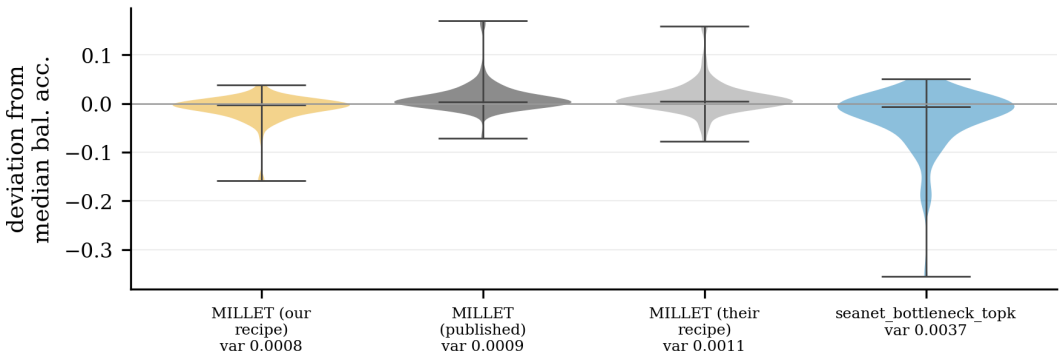


Figure 19: Model variance over the 85 UCR datasets, in the style of Early et al. (2024). For each dataset and each seed, the balanced accuracy minus the median balanced accuracy of that model on that dataset, pooled over the datasets; the models are ordered from left to right by their total variance, which is printed under each of them. Our SEA-Net model is the least stable of the four and has a long lower tail, i.e. on a few datasets one seed falls far below the other two, which is the same effect that Table 28 shows on WebTraffic.

E.5 Complete UCR Results

On the UCR archive, the main text reports that our model reaches 0.810 mean accuracy, behind MILLET with the matched budget (0.827) and behind the published MILLET accuracy (0.8445, Early et al. 2024), and that the wide variant of the same design closes this gap. Table 29 is the evidence behind these statements (Section 4.1). With the training budget fixed, changing the backbone at a similar size moves the mean rank from 12.60 to 11.53, about 1.1 places, so our design is not what loses accuracy; the reduced capacity is.

Table 29: UCR-85 accuracy and mean rank. Mean rank is over the 84 datasets finished by all 28 models we ran on the archive; lower is better. Win/tie/loss is counted per dataset against MILLET's *published* accuracies. ‡84 of 85 datasets finished.

Model	Params	Acc.	Mean rank	W/T/L
MILLET, published (Early et al., 2024)	423,707	0.8445	—	—
SEA-Net wide + class-wise conjunctive	269,164	0.8292	**11.53**	26/19/39
SEA-Net wide + MILLET additive	269,083	0.8298	11.98	23/19/42
SEA-Net wide + MILLET conjunctive	269,083	0.8238	12.43	22/20/42
MILLET, matched budget	423,707	0.8274‡	12.60	13/25/46
ResNet + conjunctive	506,331	0.8146	13.54	25/11/49
FCN + conjunctive	267,035	0.8141	14.48	20/12/52
SEA-Net gated + Top-k	61,740	0.8083	15.29	19/13/53
SEA-Net bottleneck + Top-k	**41,324**	0.8097	15.40	20/15/50

The remaining tables give the full detail. Table 30 lists every model that completed the archive, sorted by mean accuracy, where n is the number of datasets it finished out of the 85, and Table 31 counts the wins, ties and losses of the same models against the published per-dataset numbers of MILLET. Table 32 and Table 33 then give one row per dataset and one column per model, for the accuracy and for AOPCR respectively, so that a single dataset can be looked up instead of a mean.

A Wilcoxon–Holm test over the same datasets finds that most of the differences in the middle of Table 30 are not significant at $\alpha = 0.05$. This is one more reason why the main text argues from WebTraffic and from the cost rather than from the UCR rank, and why we give the win / tie / loss counts instead of relying on the rank order alone.

Figure 20 shows the accuracy comparison with one point per dataset, our model against the published MILLET accuracy (Early et al., 2024). Figure 21 then checks, on the UCR datasets whose normalized class entropy is below one, whether an imbalanced training set hurts the smaller model more, and Figure 22 checks the same for the length of the series and for the number of training series. In all three cases the answer is the same: the ranking of the models is stable, so our model does not fall further behind on the imbalanced, the long or the small datasets.

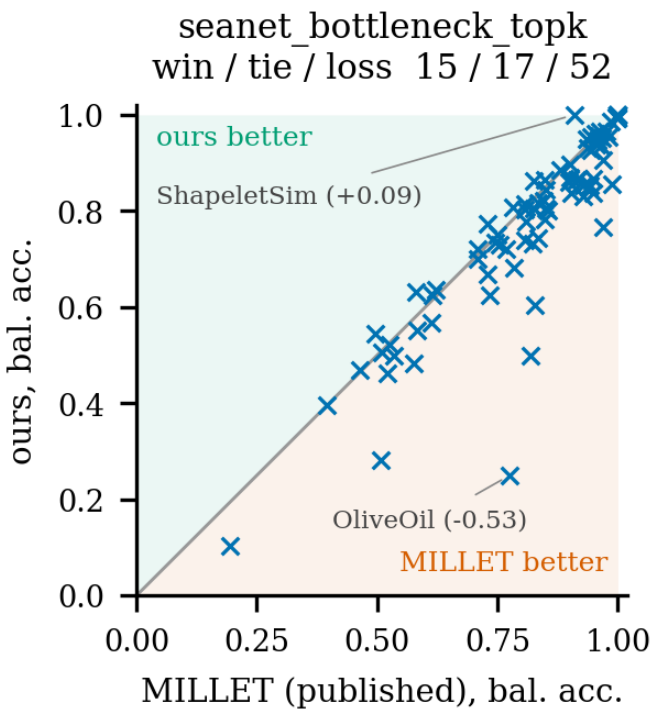


Figure 20: Direct comparison of our SEA-Net bottleneck + Top-k model against the published MILLET accuracies on the 85 UCR datasets, one point per dataset. The points above the diagonal are the datasets where our model is the more accurate one. The title gives the number of wins / ties / losses (15 / 17 / 52, where a tie is a difference below 0.005), and the datasets with the largest positive and negative differences are named.

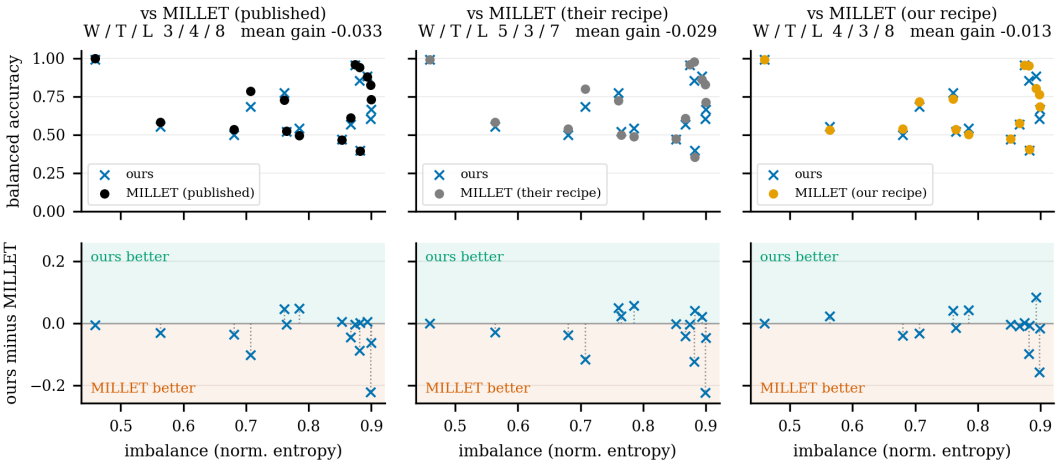


Figure 21: The effect of the class imbalance, on the UCR datasets whose training set is imbalanced (normalized class entropy below one). *Top:* the balanced accuracy of our model and of MILLET measured three ways, i.e. published, re-trained with the published recipe, and re-trained with the configuration of Table 15. *Bottom:* the difference, ours minus MILLET, where green means that our model is the better one. Each title gives the wins / ties / losses and the mean gain. The imbalance does not hurt the smaller model more than the baseline: the differences are small on both sides, and they do not grow with the imbalance.

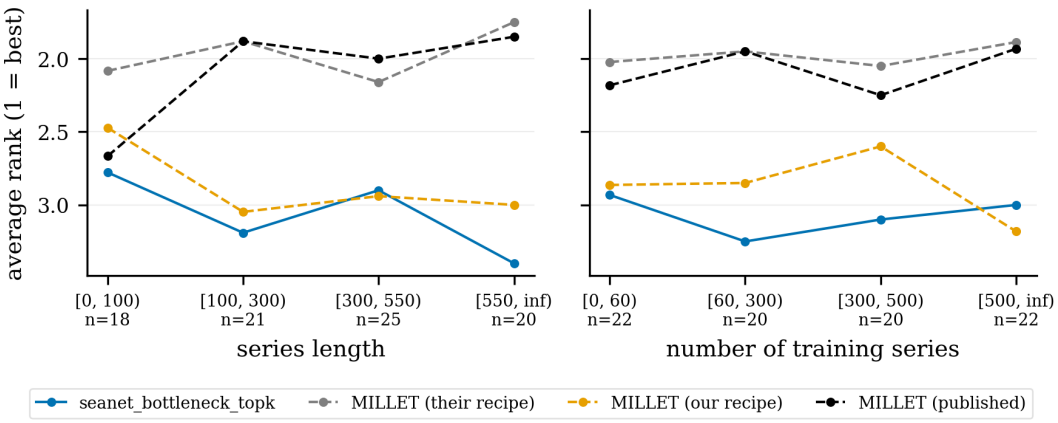


Figure 22: Average accuracy rank (1 = best, lower is better) of our model and of the three MILLET measurements, binned by two properties of the dataset: the length of the series (*left*) and the number of training series (*right*). The bin edges are chosen to give bins of roughly equal size, and n is the number of datasets in each bin. The ranking is stable across the bins, i.e. our SEA-Net model does not fall further behind on the long series or on the small training sets.

Table 30: The 85 UCR datasets: every model that completed the archive, sorted by mean accuracy (n = number of datasets finished, out of the 85 that MILLET publishes). The means are over the datasets that each model finished; the win / tie / loss counts of the same rows are given in Table 31. AOPCR is unnormalised: the rows trained with our recipe can be compared with each other, whereas the † row (the published numbers of MILLET) is on another scale. The best value of each column is in bold (the † row is excluded for AOPCR).

#	Model	n	Acc.	Bal. acc.	AUROC	NLL	AOPCR	Params
1	SEA-Net base (wide, $d = 128$) + additive (MILLET)	85	0.8298	0.8077	0.9295	**0.5088**	0.551	269,083
2	SEA-Net base (wide, $d = 128$) + class-wise conjunctive	85	0.8292	0.8039	0.9255	0.5113	0.564	269,164
3	MILLET (InceptionTime + conjunctive), our recipe	84	0.8274	0.7997	0.9275	0.5297	0.812	423,707
4	SEA-Net base (wide, $d = 128$) + conjunctive (MILLET)	85	0.8238	0.7993	0.9281	0.5180	0.626	269,083
5	SEA-Net gated (max) + class-wise conjunctive	85	0.8237	0.7979	0.9288	0.5214	0.619	61,740
6	SEA-Net input-gated + adaptive class-wise	85	0.8153	0.7905	0.9208	0.5545	0.964	58,102
7	SEA-Net bottleneck + gated attention	85	0.8146	0.7879	0.9254	0.5330	0.688	41,844
8	ResNet + conjunctive	85	0.8146	0.7839	0.9232	0.5709	1.330	506,331
9	FCN + conjunctive	85	0.8141	0.7846	0.9216	0.5727	**1.378**	267,035
10	SEA-Net reconstruction-residual + adaptive class-wise	85	0.8140	0.7859	0.9232	0.5502	1.091	62,935
11	SEA-Net base (wide, $d = 128$) + adaptive class-wise	85	0.8134	0.7849	0.9199	0.5354	0.891	269,174
12	SEA-Net base (wide, $d = 128$) + softmax conjunctive	85	0.8131	0.7858	0.9212	0.5436	1.247	269,165
13	SEA-Net input-gated + Top-k conjunctive	85	0.8131	0.7877	0.9223	0.5365	0.964	58,092
14	SEA-Net gated (max) + Top-k conjunctive	85	0.8108	0.7850	0.9172	0.5464	0.936	61,740
15	SEA-Net bottleneck + softmax conjunctive	85	0.8106	0.7856	0.9167	0.5509	0.759	41,325

continued on the next page

Table 30 (continued)

#	Model	n	Acc.	Bal. acc.	AUROC	NLL	AOPCR	Params
16	SEA-Net bottleneck + Top-k conjunctive	85	0.8097	0.7827	0.9179	0.5506	1.091	41,324
17	SEA-Net base + Top-k conjunctive	85	0.8093	0.7833	0.9182	0.5525	1.018	57,580
18	SEA-Net spike/trend + Top-k conjunctive	85	0.8089	0.7832	0.9137	0.5460	1.108	67,020
19	SEA-Net gated (mean) + Top-k conjunctive	85	0.8083	0.7803	0.9168	0.5560	1.064	61,740
20	SEA-Net reconstruction-residual + Top-k conjunctive	85	0.8063	0.7793	0.9156	0.5530	1.222	62,925
21	SEA-Net gated (last) + gated attention	85	0.8048	0.7804	0.9157	0.5564	0.500	62,260
22	SEA-Net spike/trend + gated attention	85	0.8039	0.7793	0.9112	0.5537	0.682	67,540
23	SEA-Net gated (last) + adaptive class-wise	85	0.8029	0.7797	0.9139	0.5633	0.796	61,750
24	SEA-Net gated (last) + Top-k conjunctive	85	0.8015	0.7749	0.9131	0.5531	0.967	61,740
25	SEA-Net spike/trend + adaptive class-wise	85	0.8010	0.7766	0.9135	0.5641	0.677	67,030
26	SEA-Net gated + class-wise conjunctive	85	0.7900	0.7657	0.9055	0.5848	0.486	61,740
27	SEA-Net gated (last) + class-wise conjunctive	85	0.7888	0.7658	0.9027	0.5922	0.535	61,740
28	MILLET conjunctive InceptionTime, published (mean of 5 seeds)	85	**0.8445**	**0.8235**	**0.9329**	1.2241	4.553[†]	423,300

Table 31: Wins / ties / losses (W/T/L) of the models of Table 30, with the same rows and the same numbering, against the published per-dataset numbers of MILLET, with tie bands of 0.005 (accuracy), 0.01 (NLL) and 0.1 (AOPCR). The published AOPCR values come from MILLET's recipe and are on another scale, which is why every model "loses" the AOPCR W/T/L against them.

#	Model	W/T/L acc.	W/T/L NLL	W/T/L AOPCR
1	SEA-Net base (wide, $d = 128$) + additive (MILLET)	24/19/42	60/2/23	3/0/82
2	SEA-Net base (wide, $d = 128$) + class-wise conjunctive	27/19/39	60/1/24	3/0/82
3	MILLET (InceptionTime + conjunctive), our recipe	13/25/46	56/2/26	3/0/81
4	SEA-Net base (wide, $d = 128$) + conjunctive (MILLET)	23/20/42	57/5/23	4/0/81
5	SEA-Net gated (max) + class-wise conjunctive	20/17/48	61/0/24	4/0/81
6	SEA-Net input-gated + adaptive class-wise	20/15/50	60/0/25	4/0/81
7	SEA-Net bottleneck + gated attention	23/14/48	59/1/25	5/0/80
8	ResNet + conjunctive	25/11/49	55/0/30	8/0/77
9	FCN + conjunctive	21/12/52	55/1/29	5/3/77
10	SEA-Net reconstruction-residual + adaptive class-wise	19/18/48	58/1/26	4/0/81
11	SEA-Net base (wide, $d = 128$) + adaptive class-wise	25/17/43	57/1/27	4/1/80
12	SEA-Net base (wide, $d = 128$) + softmax conjunctive	17/13/55	60/7/18	4/0/81

continued on the next page

Table 31 (continued)

#	Model	W/T/L acc.	W/T/L NLL	W/T/L AOPCR
13	SEA-Net input-gated + Top-k conjunctive	23/16/46	60/1/24	3/1/81
14	SEA-Net gated (max) + Top-k conjunctive	22/16/47	59/0/26	5/1/79
15	SEA-Net bottleneck + softmax conjunctive	22/17/46	58/1/26	3/0/82
16	SEA-Net bottleneck + Top-k conjunctive	20/15/50	58/1/26	4/0/81
17	SEA-Net base + Top-k conjunctive	20/11/54	60/1/24	5/0/80
18	SEA-Net spike/trend + Top-k conjunctive	19/23/43	56/1/28	6/0/79
19	SEA-Net gated (mean) + Top-k conjunctive	19/13/53	58/1/26	4/0/81
20	SEA-Net reconstruction-residual + Top-k conjunctive	19/19/47	59/0/26	5/0/80
21	SEA-Net gated (last) + gated attention	17/15/53	58/0/27	3/0/82
22	SEA-Net spike/trend + gated attention	15/14/56	58/1/26	4/0/81
23	SEA-Net gated (last) + adaptive class-wise	18/18/49	57/0/28	4/1/80
24	SEA-Net gated (last) + Top-k conjunctive	17/17/51	59/0/26	3/1/81
25	SEA-Net spike/trend + adaptive class-wise	24/12/49	57/0/28	4/0/81
26	SEA-Net gated + class-wise conjunctive	16/15/54	55/1/29	3/0/82
27	SEA-Net gated (last) + class-wise conjunctive	17/12/56	55/0/30	3/0/82
28	MILLET conjunctive InceptionTime, published (mean of 5 seeds)	–	–	–

Table 32: Per-dataset accuracy on the 85 UCR datasets that MILLET publishes. *MILLET pub.* is the published mean of 5 seeds of MILLET; the next four columns are SEA-Net encoders with the Top-k conjunctive head (the model of the paper is *bottleneck*); *MILLET (ours)*, *ResNet* and *FCN* are the baselines re-trained under our recipe, each with conjunctive pooling. The best value of each row is in bold (the published column included).

#	Dataset	MILLET pub.	bottleneck	gated (mean)	input-gated	spike/trend	MILLET (ours)	ResNet	FCN
1	Adiac	**0.832**	0.733	0.788	0.749	0.772	0.812	0.785	0.788
2	ArrowHead	0.838	0.851	0.829	0.840	0.869	0.827	**0.886**	0.840
3	Beef	0.847	0.822	**0.867**	0.833	0.767	0.800	0.800	0.733
4	BeetleFly	**0.920**	0.867	0.750	0.850	0.900	0.883	0.850	0.700
5	BirdChicken	0.940	0.850	0.800	0.950	0.850	0.983	**1.000**	0.900
6	Car	0.920	0.867	0.883	0.917	0.883	0.911	**0.933**	0.883
7	CBF	1.000	0.993	0.947	0.999	**1.000**	0.997	0.989	0.997
8	ChlorineConcentration	**0.858**	0.681	0.700	0.598	0.616	0.803	0.590	0.780
9	CinCECGTorso	0.824	0.732	0.749	0.704	0.768	0.844	0.860	**0.897**
10	Coffee	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
11	Computers	0.769	0.721	0.640	0.688	0.596	0.741	0.732	**0.828**
12	CricketX	0.827	0.811	0.833	0.818	**0.836**	0.808	0.718	0.710
13	CricketY	0.823	0.805	0.772	**0.836**	0.813	0.818	0.679	0.715
14	CricketZ	**0.857**	0.816	0.800	0.836	0.823	0.817	0.774	0.764
15	DiatomSizeReduction	**0.972**	0.966	0.935	0.954	0.971	0.948	0.951	0.922
16	DistalPhalanxOutlineAgeGroup	0.715	0.755	0.770	0.770	0.770	0.734	**0.784**	0.734
17	DistalPhalanxOutlineCorrect	0.770	0.775	0.768	0.750	0.772	**0.777**	0.768	0.732
18	DistalPhalanxTW	0.660	0.703	0.698	0.698	**0.712**	0.700	0.698	0.705
19	Earthquakes	0.725	0.748	0.748	0.748	**0.777**	0.755	0.741	0.748
20	ECG200	0.896	0.887	0.890	**0.900**	0.850	0.830	0.830	0.890
21	ECG5000	0.942	0.944	0.933	**0.945**	0.942	0.940	0.941	0.942
22	ECGFiveDays	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.957	0.978

continued on the next page

Table 32 (continued)

#	Dataset	MILLET pub.	bottleneck	gated (mean)	input-gated	spike/trend	MILLET (ours)	ResNet	FCN
23	ElectricDevices	0.712	0.719	0.696	0.740	0.712	0.708	**0.750**	0.729
24	FaceAll	**0.953**	0.805	0.822	0.807	0.834	0.925	0.922	0.919
25	FaceFour	0.959	0.962	**0.977**	0.920	0.955	0.958	0.920	0.932
26	FacesUCR	0.960	0.946	0.960	**0.961**	**0.961**	0.955	0.949	0.910
27	FiftyWords	**0.835**	0.784	0.809	0.811	0.789	0.807	0.692	0.637
28	Fish	**0.981**	0.952	0.954	0.971	0.954	0.977	0.960	0.971
29	FordA	0.954	0.962	0.967	0.963	0.961	0.957	**0.968**	0.921
30	FordB	0.850	0.845	0.837	0.826	**0.858**	0.840	0.836	0.785
31	GunPoint	0.985	0.987	0.987	0.993	0.993	0.987	**1.000**	**1.000**
32	Ham	0.709	0.721	0.733	0.657	0.686	0.711	0.686	**0.752**
33	HandOutlines	0.948	0.943	**0.951**	0.938	0.897	0.939	0.916	0.889
34	Haptics	**0.524**	0.464	0.393	0.448	0.432	0.497	0.442	0.471
35	Herring	0.578	0.641	0.641	0.656	0.594	0.583	**0.672**	**0.672**
36	InlineSkate	**0.491**	0.276	0.293	0.304	0.309	0.418	0.364	0.293
37	InsectWingbeatSound	0.615	**0.625**	0.610	0.607	0.616	0.597	0.444	0.337
38	ItalyPowerDemand	0.951	0.965	**0.969**	0.966	0.967	–	0.932	0.962
39	LargeKitchenAppliances	0.901	0.898	0.904	**0.920**	0.901	0.904	0.912	0.893
40	Lightning2	0.856	0.803	**0.885**	0.770	0.853	0.803	0.738	0.672
41	Lightning7	0.827	0.772	0.822	**0.836**	0.781	0.804	0.822	0.794
42	Mallat	0.969	0.765	0.683	0.768	0.963	0.960	0.934	**0.969**
43	Meat	**0.987**	0.856	0.933	0.817	0.883	0.983	0.817	0.800
44	MedicalImages	**0.802**	0.741	0.712	0.730	0.720	0.754	0.762	0.746
45	MiddlePhalanxOutlineAgeGroup	0.522	0.574	0.610	0.591	0.623	0.606	**0.656**	0.649
46	MiddlePhalanxOutlineCorrect	0.810	0.817	0.790	**0.828**	**0.828**	0.801	0.804	0.808
47	MiddlePhalanxTW	0.510	0.574	0.558	0.506	0.571	0.589	0.571	**0.591**
48	MoteStrain	0.902	0.864	0.881	0.865	0.891	0.904	**0.906**	0.893
49	NonInvasiveFetalECGThorax1	**0.951**	0.869	0.908	0.882	0.922	0.943	0.939	0.945
50	NonInvasiveFetalECGThorax2	**0.953**	0.840	0.930	0.903	0.928	0.938	0.927	0.951
51	OliveOil	**0.847**	0.400	0.400	0.400	0.400	0.556	0.400	0.800
52	OSULeaf	0.947	0.843	0.872	0.876	0.835	0.950	**0.975**	0.955
53	PhalangesOutlinesCorrect	**0.830**	0.809	0.814	0.823	0.822	0.823	0.803	0.780
54	Phoneme	**0.301**	0.222	0.208	0.196	0.207	0.268	0.240	0.260
55	Plane	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
56	ProximalPhalanxOutlineAgeGroup	0.854	0.842	0.829	0.820	0.859	0.839	**0.868**	0.844
57	ProximalPhalanxOutlineCorrect	**0.923**	0.884	0.900	0.880	0.873	0.901	0.883	0.869
58	ProximalPhalanxTW	0.760	0.780	**0.800**	0.785	0.795	0.768	0.795	0.795
59	RefrigerationDevices	0.510	0.506	0.469	**0.517**	**0.517**	0.506	0.491	0.501
60	ScreenType	0.576	0.484	0.491	0.539	0.424	0.541	0.560	**0.637**
61	ShapeletSim	0.909	**1.000**	**1.000**	**1.000**	0.483	0.998	**1.000**	**1.000**
62	ShapesAll	**0.928**	0.854	0.855	0.857	0.875	0.907	0.855	0.870
63	SmallKitchenAppliances	0.782	**0.808**	0.803	0.800	0.773	0.772	0.797	0.800
64	SonyAIBORobotSurface1	0.934	0.836	0.787	0.887	0.809	0.946	0.977	**0.987**
65	SonyAIBORobotSurface2	0.827	0.871	0.854	0.864	0.877	0.816	**0.959**	0.932
66	StarLightCurves	**0.977**	0.970	0.970	0.974	0.972	0.975	0.975	0.969
67	Strawberry	**0.975**	0.964	0.960	0.954	0.946	0.960	0.941	0.965
68	SwedishLeaf	0.967	0.949	0.933	0.947	0.941	0.967	**0.973**	0.971
69	Symbols	0.958	0.948	0.944	0.957	**0.959**	0.926	0.922	0.931
70	SyntheticControl	0.997	0.996	0.993	0.997	0.993	0.999	**1.000**	0.983
71	ToeSegmentation1	**0.963**	0.936	0.934	0.952	0.908	0.958	0.952	0.952

continued on the next page

Table 32 (continued)

#	Dataset	MILLET pub.	bottleneck	gated (mean)	input-gated	spike/trend	MILLET (ours)	ResNet	FCN
72	ToeSegmentation2	0.909	0.944	0.954	0.923	0.946	0.879	**0.962**	0.939
73	Trace	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
74	TwoLeadECG	0.999	0.995	0.972	0.990	0.972	0.999	0.998	**1.000**
75	TwoPatterns	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	1.000	0.995
76	UWaveGestureLibraryAll	0.945	0.954	**0.965**	0.959	0.958	0.945	0.895	0.913
77	UWaveGestureLibraryX	**0.810**	0.804	0.805	0.789	0.806	0.788	0.786	0.792
78	UWaveGestureLibraryY	**0.753**	0.728	0.731	0.719	0.751	0.728	0.723	0.702
79	UWaveGestureLibraryZ	0.743	0.733	0.711	**0.763**	0.738	0.732	0.753	0.709
80	Wafer	0.998	0.997	0.998	**0.999**	0.996	0.997	0.998	0.997
81	Wine	**0.819**	0.500	0.500	0.500	0.593	0.500	0.500	0.500
82	WordSynonyms	0.742	0.706	**0.745**	0.718	0.730	0.725	0.538	0.578
83	Worms	**0.766**	0.697	0.610	0.701	0.714	0.697	0.740	0.727
84	WormsTwoClass	0.706	0.697	0.701	**0.766**	0.740	0.714	0.753	0.714
85	Yoga	**0.903**	0.872	0.884	0.893	0.873	0.848	0.720	0.749
	Mean	0.845	0.810	0.808	0.813	0.809	0.827	0.815	0.814

Table 33: Per-dataset AOPCR on the 85 UCR datasets that MILLET publishes. *MILLET pub.* is the published mean of 5 seeds of MILLET; the next four columns are SEA-Net encoders with the Top-k conjunctive head (the model of the paper is *bottleneck*); *MILLET (ours)*, *ResNet* and *FCN* are the baselines re-trained under our recipe, each with conjunctive pooling. The best value of each row is in bold among the columns trained with our recipe; the published column (†) was trained with MILLET's recipe and its AOPCR is on another scale, so it is shown but not compared.

#	Dataset	MILLET pub.†	bottleneck	gated (mean)	input-gated	spike/trend	MILLET (ours)	ResNet	FCN
1	Adiac	15.89	-0.83	-0.02	-0.27	-0.64	1.60	-2.05	**1.73**
2	ArrowHead	3.20	0.24	0.45	**1.37**	0.67	0.60	0.84	1.31
3	Beef	5.49	1.33	1.95	0.80	-0.30	1.34	**2.07**	1.63
4	BeetleFly	1.24	0.58	**2.35**	1.18	1.59	0.24	2.29	1.35
5	BirdChicken	3.37	1.96	**2.77**	1.30	1.58	0.96	1.15	1.36
6	Car	6.62	0.40	1.81	-0.03	2.80	1.02	2.55	**3.66**
7	CBF	3.58	**1.97**	1.58	0.92	1.71	0.87	0.97	0.85
8	ChlorineConcentration	2.14	0.49	0.31	0.80	**1.79**	0.26	0.20	1.13
9	CinCECGTorso	4.77	2.31	2.43	**2.50**	0.01	1.48	1.80	1.97
10	Coffee	3.49	0.60	0.22	1.19	**1.69**	0.89	1.05	0.64
11	Computers	3.44	**0.89**	0.65	0.53	0.45	0.53	0.74	0.47
12	CricketX	4.51	**2.03**	1.40	1.36	1.34	1.27	1.78	1.87
13	CricketY	4.89	1.75	1.80	1.45	1.21	1.22	**1.95**	1.94
14	CricketZ	4.77	1.83	1.14	1.28	1.71	1.36	**2.13**	1.93
15	DiatomSizeReduction	1.03	2.41	-0.02	0.92	**2.44**	-0.15	0.67	0.99
16	DistalPhalanxOutlineAgeGroup	2.43	0.44	0.17	0.48	0.02	0.66	**0.88**	0.81
17	DistalPhalanxOutlineCorrect	0.64	0.52	0.54	0.27	-0.29	0.30	**0.86**	0.73
18	DistalPhalanxTW	3.12	1.05	0.92	**1.61**	-0.71	0.71	0.71	0.85
19	Earthquakes	3.49	0.20	0.11	0.15	0.48	**0.73**	0.35	0.13
20	ECG200	2.86	0.38	0.74	0.10	**1.02**	0.44	0.49	0.45

continued on the next page

Table 33 (continued)

#	Dataset	MILLET pub.†	bottleneck	gated (mean)	input-gated	spike/trend	MILLET (ours)	ResNet	FCN
21	ECG5000	6.70	1.01	1.32	0.49	1.21	1.22	1.26	**1.46**
22	ECGFiveDays	4.76	0.68	-0.51	-0.01	0.31	0.19	0.27	**1.31**
23	ElectricDevices	4.64	0.87	1.06	0.75	1.19	0.72	1.22	**1.39**
24	FaceAll	4.00	0.78	0.94	0.25	0.41	0.20	1.23	**1.41**
25	FaceFour	6.47	1.23	0.62	1.13	**1.59**	0.47	1.21	0.97
26	FacesUCR	3.18	1.07	**1.20**	1.06	0.90	0.03	0.95	0.93
27	FiftyWords	3.31	2.19	1.78	1.73	1.77	0.84	**2.39**	2.38
28	Fish	5.48	0.71	1.09	1.00	1.84	1.51	2.33	**3.45**
29	FordA	2.71	**0.87**	0.73	-0.11	0.70	0.28	-0.31	0.75
30	FordB	1.64	0.51	0.03	0.51	**0.93**	0.37	0.18	0.25
31	GunPoint	4.01	**1.05**	0.49	0.70	0.70	0.91	0.48	0.74
32	Ham	3.93	1.11	1.51	1.10	**1.82**	0.75	1.59	1.62
33	HandOutlines	12.71	1.48	2.90	2.02	**5.52**	1.85	1.34	3.51
34	Haptics	5.99	1.63	1.37	2.13	1.44	1.25	2.01	**2.47**
35	Herring	4.61	1.44	1.80	0.95	**5.24**	2.70	3.18	2.34
36	InlineSkate	8.16	1.52	1.88	**2.70**	0.80	2.05	1.98	1.75
37	InsectWingbeatSound	2.73	1.18	**1.43**	0.69	1.02	0.61	0.82	0.97
38	ItalyPowerDemand	4.16	0.43	**0.92**	0.42	0.30	–	0.54	0.49
39	LargeKitchenAppliances	9.51	**2.20**	1.22	1.70	1.63	1.68	1.80	1.57
40	Lightning2	5.38	0.98	1.09	1.62	1.00	0.96	**1.67**	1.45
41	Lightning7	4.25	**1.86**	1.16	1.38	1.16	1.30	1.48	1.36
42	Mallat	6.74	1.87	2.28	2.56	1.54	1.76	**2.99**	2.52
43	Meat	6.19	1.62	1.59	0.68	0.95	**2.08**	1.74	2.01
44	MedicalImages	2.89	0.91	0.96	0.52	0.69	0.92	0.99	**1.07**
45	MiddlePhalanxOutlineAgeGroup	0.99	0.35	0.35	0.25	0.41	0.86	**1.25**	0.78
46	MiddlePhalanxOutlineCorrect	2.45	0.72	0.58	-0.18	-0.36	0.13	2.69	**2.84**
47	MiddlePhalanxTW	3.40	0.66	0.62	0.66	0.05	0.57	**1.04**	0.47
48	MoteStrain	3.56	0.52	1.39	0.75	**2.12**	0.55	0.46	0.58
49	NonInvasiveFetalECGThorax1	-2.26	1.73	1.46	1.29	1.23	0.30	**8.51**	5.18
50	NonInvasiveFetalECGThorax2	2.99	1.73	1.41	1.23	1.00	0.50	**3.77**	2.93
51	OliveOil	-3.63	0.12	0.08	0.12	0.37	**3.79**	0.37	-1.05
52	OSULeaf	6.53	1.26	0.73	1.16	0.58	0.84	**2.56**	1.90
53	PhalangesOutlinesCorrect	2.04	0.04	0.45	0.36	0.31	0.33	-0.32	**0.48**
54	Phoneme	4.78	**1.71**	1.16	1.16	1.26	0.50	0.64	0.53
55	Plane	3.39	**1.13**	1.05	1.07	1.07	0.31	0.26	0.46
56	ProximalPhalanxOutlineAgeGroup	5.65	0.07	-0.25	0.56	**1.05**	1.04	0.92	0.11
57	ProximalPhalanxOutlineCorrect	-3.54	0.08	0.31	-1.67	-0.12	**0.69**	-1.35	0.68
58	ProximalPhalanxTW	5.70	-0.10	-0.88	-1.46	**0.57**	-2.17	-0.64	-0.86
59	RefrigerationDevices	4.25	0.32	0.28	**1.41**	0.28	0.63	0.86	0.71
60	ScreenType	4.22	0.83	0.37	0.53	0.56	0.69	1.07	**1.15**
61	ShapeletSim	3.43	1.92	**2.46**	1.15	0.23	0.74	1.18	0.85
62	ShapesAll	9.56	2.36	2.23	2.28	1.44	1.64	**3.82**	3.15
63	SmallKitchenAppliances	3.93	0.90	**1.09**	0.50	0.32	0.50	0.90	1.04
64	SonyAIBORobotSurface1	3.26	0.58	-0.48	0.07	0.22	0.31	**0.63**	0.50
65	SonyAIBORobotSurface2	2.96	0.10	-0.13	0.40	**0.45**	0.17	0.35	0.15
66	StarLightCurves	8.42	2.65	1.69	1.89	**3.22**	1.01	2.83	1.86
67	Strawberry	6.22	0.61	0.16	0.78	0.75	0.51	1.25	**2.58**
68	SwedishLeaf	4.16	**1.32**	1.17	0.98	0.84	0.47	0.89	0.94
69	Symbols	2.56	1.29	1.17	1.87	**2.04**	1.20	1.31	1.71

continued on the next page

Table 33 (continued)

#	Dataset	MILLET pub.†	bottleneck	gated (mean)	input-gated	spike/trend	MILLET (ours)	ResNet	FCN
70	SyntheticControl	3.30	**0.95**	0.40	0.57	0.74	0.66	0.75	0.54
71	ToeSegmentation1	2.18	0.53	0.24	0.65	0.63	0.33	0.64	**0.86**
72	ToeSegmentation2	3.21	0.91	0.42	0.83	0.31	0.41	**1.14**	0.62
73	Trace	4.92	2.05	1.40	1.82	**2.46**	1.76	2.24	2.21
74	TwoLeadECG	4.76	**1.49**	0.94	1.02	0.12	-0.22	0.83	0.29
75	TwoPatterns	12.16	2.42	**3.07**	2.22	1.89	1.35	2.88	1.29
76	UWaveGestureLibraryAll	8.08	1.95	2.03	2.43	1.46	1.27	2.45	**2.96**
77	UWaveGestureLibraryX	4.30	1.21	1.70	1.09	0.81	0.85	1.78	**1.99**
78	UWaveGestureLibraryY	4.10	1.23	1.55	1.12	0.93	0.87	**1.66**	1.61
79	UWaveGestureLibraryZ	4.65	1.44	1.62	1.13	1.22	0.92	**1.70**	1.57
80	Wafer	1.14	0.37	0.50	**0.84**	0.27	-0.27	0.75	0.77
81	Wine	26.96	0.61	2.27	1.12	**3.31**	0.03	0.05	0.03
82	WordSynonyms	3.04	1.49	1.28	1.33	0.94	0.69	2.09	**2.20**
83	Worms	3.11	1.22	1.46	1.89	1.59	0.89	1.63	**2.15**
84	WormsTwoClass	4.30	0.95	1.19	1.01	1.13	0.90	1.29	**1.74**
85	Yoga	6.62	1.28	1.72	1.80	**3.22**	0.64	1.14	2.62
	Mean	4.55	1.09	1.06	0.96	1.11	0.81	1.33	1.38

E.6 Relation to the Published MILLET Results

The main comparisons of the paper use MILLET retrained with our own budget, and not its published numbers. Table 34 sets our retrained baseline next to the published numbers of MILLET (Early et al., 2024), with one row per way of obtaining the number; the WebTraffic columns are means over the runs and the UCR columns are means over the 85 published datasets. Early et al. (2024) report 0.924 over five seeds on WebTraffic, and our retrained baseline reaches 0.888 with the same architecture. The two differ in the training recipe (Appendix D.2).

Table 34: The MILLET baseline (InceptionTime + conjunctive pooling): the published numbers of Early et al. (2024) and our retrained baseline. The WebTraffic columns are means (± std) over the runs in both results folders, and the UCR columns are means over the 85 published datasets (the number in brackets if fewer were finished). Only the last row is used in the main comparisons. †: the published AOPCR comes from MILLET's own recipe and is on another scale, so AOPCR must only be compared within one recipe.

			WebTraffic			UCR-85 mean	
#	How the number was obtained	Runs	Acc.	AOPCR	NDCG@n	Acc.	AOPCR
1	Published, mean of 5 seeds	5	0.924	12.768†	0.674	0.8445	4.553†
2	Published, 5-model ensemble	1	0.940	11.100†	0.694	–	–
3	**Re-trained here, our recipe (400 epochs, label smoothing, early stopping): the baseline of every comparison**	4	0.888 ±0.007	2.457 ±0.655	0.681 ±0.013	0.8274 (84)	0.812

The same table also shows the problem of Section 4.1. The published WebTraffic AOPCR of MILLET is 12.768, against 2.457 for our retrained baseline, although the two have a similar NDCG@n (0.674 and 0.681). A factor of about five on a metric that is supposed to measure the quality of an explanation therefore does not reflect a better ranking of the time points. This is why the AOPCR column of Table 1 is only read across its own rows, and never against an AOPCR reported in another paper, including the published 4.55 of MILLET.