information

MDPI

Article

ProtoPGTN: A Scalable Prototype-Based Gated Transformer Network for Interpretable Time Series Classification

Jinjin Huang, Ce Guo * and Wayne Luk

Department of Computing, Imperial College London, South Kensington Campus, London SW7 2AZ, UK; jinjin.huang24@imperial.ac.uk (J.H.); w.luk@imperial.ac.uk (W.L.)
* Correspondence: c.guo@imperial.ac.uk

Abstract

Time Series Classification (TSC) plays a crucial role in machine learning applications across domains such as healthcare, finance, and industrial systems. In these domains, TSC requires accurate predictions and reliable explanations, as misclassifications may lead to severe consequences. In addition, scalability issues, including training time and memory consumption, are critical for practice usage. To address these challenges, we propose ProtoPGTN, a prototype-based interpretable framework that unifies gated transformers with prototype reasoning for scalable time series classification. Unlike existing prototype-based interpretable TSC models which rely on recurrent structure for sequence processing and Euclidean distance for similarity computation, ProtoPGTN adapts Gated Transformer Networks (GTN), which uses an attention mechanism to capture both temporal and spatial long-range dependencies in time series data and integrates the prototype learning framework from ProtoPNet with cosine similarity to enhance metric consistency and interpretability. Extensive experiments are conducted on 165 publicly available datasets from the UCR and UEA repositories, covering both univariate and multivariate tasks. Results show that ProtoPGTN obtains at least the same performance as existing prototype-based interpretable models on both multivariate and univariate datasets. ProtoPGTN achieves up to 20× faster training and up to 200× lower memory consumption than existing prototype-based interpretable models.

Keywords: explainable AI; time series classification; prototype-based interpretability; scalability; memory; training time

check for updates

Academic Editor: Firstname Lastname

Received:
Revised:
Accepted:
Published:

Citation: Huang, J.; Guo, C.; Luk, W. ProtoPGTN: A Scalable Prototype-Based Gated Transformer Network for Interpretable Time Series Classification. *Information* **2025**, *1*, 0. https://doi.org/

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Time series data contain a series of observations indexed in time order, and have become increasingly critical across various domains such as healthcare, industry, finance, and environmental monitoring. Applications of time series data range from patient monitoring to fault detection and stock price prediction. These domains require precise prediction results, and interpretations can help professionals reason about the results. As a result, interpretable Time Series Classification (TSC) has emerged as an important task in machine learning. TSC refers to the task of assigning a label to a sequence of data points ordered in time along with proper explanations.

Interpretability is crucial in TSC, since it is closely linked to user trust [1] and model safety [2]. For example, in healthcare monitoring doctors must understand why the predictions are made before they act on them. An explanation of how different time segments on prediction contribution can build confidence in whether the model is relying on valid and

domain-specific patterns. Moreover, the reasoning process can help users detect potentially dangerous failure modes and biases. This prevents incorrect decisions from leading to severe consequences such as misdiagnosing a medical condition. However, many deep learning models do not provide interpretations of how the decisions are made, although they have very high accuracy. These models function as black boxes with limited transparency. This lack of interpretability prevents users from understanding why predictions are made, reducing trust and increasing risks in high-stakes domains such as healthcare.

In addition, existing prototype-based interpretable models suffer from fundamental architectural limitations. Time series data often exhibit long-range temporal dependencies and complex interactions across variables. However, recurrent and convolutional backbones used by existing models restrict representations to local and sequential contexts and have limited ability to capture global temporal relationships. In addition, most prior approaches rely on Euclidean distance for measuring similarity between prototypes and latent feature; however, Euclidean distance can be sensitive to scale and less effective in high-dimensional feature spaces. As summarized in Table 1, nearly all prior prototype models adopt CNN, LSTM, Tree-based backbones combined with Euclidean distance across domains ranging from computer vision to time series. These design choices weaken the representativeness of learned prototypes and limit the interpretability quality. To address these limitations, we propose ProtoPGTN, which revisits the backbone architecture and similarity design.

Table 1. Comparison of prototype-based interpretable models and their applicability across domains.

Model	Backbone	Similarity Metric	Domain
ProtoPNet [3]	CNN	Euclidean distance	Computer Vision
Deformable ProtoPNet [4]	CNN	Euclidean distance	Computer Vision
NP-ProtoPNet [5]	CNN	Euclidean distance	Computer Vision
ProtoLNet [6]	CNN	Euclidean distance	Earth Systems
ProtoTree [7]	CNN + Decision Tree	Euclidean distance	Computer Vision
PGIB [8]	GNN + Information Bottleneck	Euclidean distance	Graph Data
ProtoPLSTM [9]	CNN + LSTM	Euclidean distance	Time Series (SisFall)
ProtoPGTN	Gated Transformer	Cosine similarity	General TSC

Another problem with interpretable TSC relates to scalability. Many interpretation methods can be computationally expensive for long sequences or high-dimensional datasets. Specifically, prototype-based methods learn a large number of prototypes during training. This process leads to increased training time, since repeated distance calculations between prototypes and latent features must be performed to measure similarity. Moreover, prototypes need to be maintained and refined throughout training, leading to additional storage and update costs. For example, the `DuckDuckGeese` dataset involves classifying bird species based on short audio clips of their calls. It has 1345 variables, and each variable represents the energy at a fixed frequency bin in the spectrogram over time. This task reflects broader bioacoustic applications such as biodiversity monitoring and ecological conservation. The existing prototype-based ProtoPLSTM model is estimated to consume over 500 GB of memory. This is infeasible on existing GPUs. This problem still exists when number of variable is moderate, around 30 to 50. This scalability issue is challenging since interpretable models must ensure that the explanations capture the fine-grained information; however, reducing the number of prototypes often sacrifices interpretability, creating a tradeoff between scalability and efficiency.

The key contributions of this work are as follows:

1. Prototype-Driven Explanations: ProtoPNet [3] was originally designed for image classification. It learns a set of prototypes to identify which part of the image contributes the most to the classification decision. This concept is adapted by ProtoPGTN for TSC, where prototypes are representative subsequences. Similar to ProtoPNet, classification in ProtoPGTN is based on the similarity between input samples and learned prototypes. Therefore, the decision process can be visualized through their corresponding similarity scores. The interpretability analysis demonstrates that the learned prototypes are meaningful and representative of classes. Prototypes belonging to the same class exhibit similar patterns, while those from different classes show clear distinctions. In addition, the learned prototypes are shown to be effective for prediction, as samples from a given class tend to exhibit high similarity scores with prototypes of the same class while showing consistently low similarity with prototypes associated to other classes.
2. Backbone and Similarity Innovations with ProtoPGTN: ProtoPGTN introduces two key architectural innovations to enhance both representation expressiveness and interpretability. First, it employs a Gated Transformer Network (GTN) [10] as the backbone to jointly capture temporal and spatial features. The GTN backbone uses two transformer encoders to capture long-range dependencies in two dimensions and uses a gate mechanism to combine results from two encoders. This overcomes the locality and sequential bottlenecks from LSTM and CNN-based designs. Second, it replaces Euclidean distance used in the majority of prototype-based models with cosine similarity for more robust prototype matching. This offers a scale-invariant and stable similarity metric for high-dimensional representations. These innovations are integrated with a prototype learning framework adapted from ProtoPNet [3], where prototypes correspond to representative subsequences in time series.
3. Scalable Interpretable TSC: ProtoPGTN demonstrates high scalability in terms of training efficiency and GPU memory usage, addressing the key limitations of prototype-based interpretable models. Compared to existing prototype-based interpretable classifiers, our model achieves up to 20× faster training on large-scale tasks. In addition to faster training, ProtoPGTN also significantly improves the memory consumption. The GPU memory usage on datasets with thousands of variables is under 2 GB compared to other models with estimated over 500 GB memory usage.
4. Comprehensive Evaluation Against State-of-the-Art Methods: Comprehensive experiments are conducted on 35 multivariate and 130 univariate datasets from UCR and UEA archives. We compare the accuracy against existing prototype-based model ProtoPLSTM [9], its simplified ProtoPConv version that uses only convolutional layers for feature extraction, non-interpretable models such as TimesNet [11] and Zerveas [12], and a simple MLP model [13] for a performance lower bound. Results show that ProtoPGTN achieves good performance on both multivariate and univariate time series datasets. The performance of ProtoPGTN is nearly the same as that of the existing ProtoPLSTM prototype-based model while avoiding the severe GPU memory issues that ProtoPLSTM suffers from. On univariate datasets, ProtoPGTN achieves 76.99% accuracy, which is only 1.5% lower than TimesNet [11] and 3.5% lower than Zerveas's transformer-based model [12]. TimesNet and Zerveas do not trade accuracy for interpretability, and are expected to represent the upper bound performance.

The implementation of ProtoPGTN is publicly available. (https://github.com/jinjin-huang/interpretable-time-series-classification (accessed on 30 November 2025)).

2. Background and Related Work

2.1. Time Series Classification: Problem Definition

Time series data consist of sequences collected over time, and time series classification refers to the task of assigning categorical labels to such data. In contrast to traditional data, where inputs are independent, time series data have an inherent temporal order. Therefore, classification techniques should capture dependencies within temporal order.

Formally, suppose that time series data are denoted as:

$$\mathbf{X} = \{x_1, x_2, \ldots, x_T\},$$

where T represents the sequence length and each $x_t \in \mathbb{R}^d$ represents a d-dimensional observation at time t. Time series data can be classified into univariate ($d = 1$) and multivariate ($d > 1$) types depending on the dimensionality of the observations. Multivariate time series may also have inherent correlations between different channels. The entire dataset containing N time series with labels is represented as

$$\mathcal{D} = \left\{ \left(\mathbf{X}^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N},$$

where $y^{(i)}$ represents the class label corresponding to the i-th time series data. This formulation specifically applies to time series classification. In contrast, other tasks such as forecasting or unsupervised learning do not require class labels. The goal of time series classification is to learn a function that takes the time series data as input and outputs the corresponding class from class set $\mathcal{C}$:

$$f : \mathbb{R}^{T \times d} \to \mathcal{C}.$$

2.2. Time Series Classification Methods

One classical approach to Time Series Classification (TSC) is distance-based. As its name suggests, this method quantifies the distance between two time sequences; a smaller distance indicates the two sequences are more similar. Common distance measures include lock-step measures such as L_p norms, elastic measures such as Dynamic Time Warping (DTW), and edit distance-based measures as well as threshold-based measures and pattern-based measures [14]. The K-Nearest Neighbors (KNN) algorithm is a representative distance-based technique. In TSC, the most widely used variant is 1NN with DTW [15]. An early study verified the effectiveness of three variants of 1NN (early 1NN, fixed-length 1NN, and full-length 1NN) on seven datasets from the UCR Time Series Archive [16]. A recent study compared 1NN effectiveness using different distance measures on all 128 datasets [17]. The results showed that the error rates were relatively low and that the DTW-based distance was more suitable for 1NN TSC.

Another important paradigm is shapelet-based TSC. Instead of measuring the distance over entire sequences, this approach aims to identify discriminative subsequences that are maximally representative of a particular class in time series data [18]. Generalized Random Shapelet Forests (gRSF) [19] generates several shapelet-based trees and uses the distance between the shapelet and full-length data as the splitting criterion at each tree node. The shapelets are randomly selected, and this method can be applied to both univariate and multivariate TSC. Other research has paid more attention to finding representative shapelets, since it is very time-consuming. Yang et al. [20] proposed a neural-network-based shapelet candidate searching method, which improves search speed and accuracy significantly. Moreover, the single-scan shapelet algorithm introduced by Hills [21] finds the best k shapelets to generate a transformed dataset. This dataset can be used with any classifier, allowing higher accuracy to be achieved.

In recent years, deep learning-based methods have gained increasing popularity in TSC. A widely adopted architecture is the Convolutional Neural Network (CNN). Wang et al. [13] explored Fully Convolutional Networks (FCNs), which remove the fully connected layers at the end of conventional CNNs. They used three convolution blocks with filter sizes of 128, 256, and 128, respectively. Each block was followed by batch normalization and ReLU activation. Finally, global pooling and softmax were used to make decisions. Another variant of CNN designed for TSC was introduced by Zhao et al. [22] in 2017. Compared to traditional CNNs, it used the Mean Squared Error (MSE) instead of cross-entropy as a loss function. Thus, the final softmax was replaced by a sigmoid activation. The max pooling was changed to average pooling. In addition to these CNN architectures, recent studies have integrated CNNs with transformers to capture local and global contexts. For example, He et al. proposed three closely related CNN-transformer architectures for real-world railway inspection: M2CTFNet for U-shaped bolt and nut defects (uses multiscale cross-attention and a shuffling technique in the transformer branch) [23], C2T-HR3D for dropper defects (uses cross-fusion between CNN and transformer features and enhances long-range dependencies and small targets) [24], and CTBM-DAHD for arcing-horn defects (uses a bridge mode with dual attention to balance local and global features) [25]. The models achieved high precision and recall with reduced computational cost and model parameters.

Recurrent Neural Networks (RNNs) can also be applied to TSC for modeling temporal dependencies. Hüsken and Stagge first proposed the use of RNNs for TSC in 2003 [26]. Their proposed approach performed sequence-to-sequence classification. Each subseries of input was classified, and the neuron with the highest weight after applying argmax was regarded as the classification result. However, naive RNNs suffer from the vanishing gradient problem. Long Short-Term Memory (LSTM) is used to address this issue. Sequence-to-Sequence with Attention (S2SwA) [27] contains two LSTMs as the encoder and decoder. The role of the LSTM encoder is to handle arbitrary input lengths and extract information, based on which the LSTM decoder identifies discriminatory features.

2.3. Prototype-Based Interpretability

Prototype-based interpretability lies between model-specific and model-agnostic interpretability. The explanation is generated by comparing an input instance with a set of representative examples from the training set, referred to as prototypes [28]. One of the most famous prototype-based interpretable models is ProtoPNet [3]. It employs image patches as prototypes, in contrast to whole-image prototypes used in earlier approaches [29]. The architecture of ProtoPNet begins with a convolutional backbone f such as ResNet or LeNet used to extract key features in the image. The convolutional feature map $f(x)$ has dimensionality $H \times W \times D$. This is followed by a prototype layer g_p of shape $H_1 \times W_1 \times D$, where each prototype corresponds to a patch in the latent space. From the input feature map, patches of the same size as the prototypes are extracted and their distances to the prototypes are computed. The L^2 distance to each prototype is calculated and converted into similarity scores. Next, global max pooling is used to retain only the maximum similarity in order to indicate how strongly a prototypical part is present within a patch of the input image. Mathematically, for a convolutional output $z = f(x)$, the i-th prototype unit g_{p_i} computes

$$\mathbf{g}_{p_i}(z) = \max_{\tilde{z} \in \text{patches}(z)} \log\left(\frac{\|\tilde{z} - \mathbf{p}_i\|_2^2 + 1}{\|\tilde{z} - \mathbf{p}_i\|_2^2 + \epsilon}\right),$$

where $g_{p_i}(z)$ decreases as $\|\tilde{z} - \mathbf{p}_i\|_2^2$ increases. The prototype similarity scores are then fed into a fully connected layer h, where each score is weighted to produce the output logits. A softmax activation is finally applied to obtain class probabilities. In fact, the majority of

prototype-based interpretable models follow a similar architecture containing convolutional layers, prototype layers, and a fully connected layer. The training process for ProtoPNet is also well defined, involving three stages. The model is first trained to produce several representative prototypes. The layers before the last layer are trained using Stochastic Gradient Descent (SGD) for a finite number of epochs. It is used to learn a meaningful latent space where patches that belong to the same class are clustered around similar prototypes, and clusters from different classes are well-separated. Second, a prototype projection step pushes each prototype toward its nearest training patch to enhance visual interpretability. Finally, the weights of the last fully connected layer are optimized via convex optimization.

In addition to the original ProtoPNet mentioned above, there are a wide range of ProtoPNet variants that expand the original concept of ProtoPNet. Donnelly et al. [4] presented a deformable prototypical part network (Deformable ProtoPNet). It addresses the limitation that ProtoPNet only uses spatially rigid prototypes and allows spatially flexible prototypes. Interestingly, the title of the paper introducing ProtoPNet is "This looks like that", and Singh and Yow [5] introduced a negative-positive prototypical part network (NP-ProtoPNet) with the title "These do not look like those". The original ProtoPNet classifies the input based on the similarity to each class. Conversely, NP-ProtoPNet incorporates both positive and negative prototypes, enabling the model not only to confirm classes but also to explicitly reject them. In addition to ProtoPNet and its variants, there are other models that use prototypes; for example, the Neural Prototype Tree (ProtoTree) [7] combines prototype learning with decision trees, providing global interpretability. Each node corresponds to a trainable prototype, and the decision path queries the presence or absence of prototypes in the input.

2.4. Interpretable Models for TSC

Gao et al. [9] proposed ProtoPLSTM, an adaptation of ProtoPNet for TSC. It was evaluated on the SisFall dataset for fine-grained fall detection. The ProtoPLSTM model has three components: a CNN-LSTM encoder backbone network, a contextual enhancement module, and a ProtoPNet-based network. The first and second modules are used to extract features and produce meaningful feature representations. The CNN–LSTM backbone captures both local and long-range temporal dependencies, which are then linked to prototypes for interpretable decision-making. The last ProtoPNet-based network provides interpretability through prototype-based reasoning.

In addition to prototype-based methods, other works have focused on identifying representative patterns. Senin and Malinchik [30] combined Symbolic Aggregate approXimation and Vector Space Model (SAX-VSM) to discover characteristic patterns in a time series and rank those patterns to support interpretable class generalization. Similarly, Xing et al. [31] extracted interpretable features from time series for early classification; they considered those features to be local shapelets, since they were segments in the input and highly interpretable. However, finding shapelets is computationally expensive if all subsequences of the series are examined. Liang and Wang [32] introduced a method to speed up the feature learning process. This was achieved by learning a minimal set of class-specific shapelets, possibly of variable lengths. Instead of finding shapelets in time series, Lee et al. [33] introduced Z-time to construct interpretable features by leveraging temporal abstraction and interval-based event relations.

2.5. Literature Summary

As shown in Table 2, interpretability techniques span three major categories: prototype-based, shapelet/feature-based, and model-agnostic. Prototype-based approaches, while highly intuitive in computer vision, remain largely unexplored for TSC, with ProtoPLSTM

being the only notable adaptation. Shapelet and feature-based methods are naturally aligned with TSC, since they directly identify discriminative temporal patterns. However, they often suffer from high computational cost in practice. Model-agnostic methods such as LIME and attention-based mechanisms provide flexibility across model families, yet they are limited in capturing long-range temporal dependencies. Overall, the table highlights a clear research gap: although several interpretability techniques exist, only a small subset directly supports TSC, and there is a lack of generalizable prototype-based interpretable models for diverse time series applications.

Table 2. Comparison of interpretability techniques across categories and their applicability to TSC.

Category	Technique	Interpretation	TSC
Prototype-based	ProtoPNet	Image-patch prototypes	No
	Def. ProtoPNet	Flexible prototypes	No
	NP-ProtoPNet	Pos/Neg prototypes	No
	ProtoLNet	Location-aware prototypes	No
	ProtoTree	Prototype + Decision Tree	No
	ProtoPLSTM	Prototype + CNN/LSTM	Yes
Shapelet/Feature-based	SAX-VSM	Symbolic pattern extraction	Yes
	Shapelet-based	Discriminative subsequences	Yes
	Interpretable Feature Extraction	Local shapelets for early classification	Yes
	Z-time	Temporal abstraction	Yes
Model-agnostic / Other	Linear Regression	Weight-based	No
	Decision Tree	Rule/Path-based	No
	Attention-based	Saliency/attention weights	Yes
	BETA	Rule-based approximation	No
	LIME	Local surrogate models	Yes

3. Proposed Approach

3.1. Overview

Figure 1 depicts the architecture of the ProtoPGTN. Our model consists of a Gate Transformer Network [10] f_{GTN}, a prototype-based layer g_p, and a fully connected layer h with softmax activation. The Gated Transformer Network (GTN) [10] is specially designed to capture the features in two time series dimensions (spatial and temporal) using two separate transformer encoders. A gating mechanism is then employed to combine the outputs of two separate transformer encoders. The features $f_{GTN}(x)$ learned by the GTN are as input to prototype layer g_p adapted from ProtoPNet [3]. The network learns class-specific prototypes by iteratively updating prototypes during training, capturing representations of each class and projecting the prototypes to the nearest training sample for interpretability. The similarity score between encoded input sequences and learned prototypes guides the model's decision-making process, since each prototype corresponds to a class. The final fully connected layer h is followed by a softmax activation function. They are used to produce the model's classification output by converting the similarity scores into probabilities. The model is trained end-to-end using cross-entropy loss.

In particular, Table 3 compares ProtoPGTN and ProtoPLSTM. In ProtoPLSTM, a 2D convolutional layer followed by an LSTM is employed for feature extraction, and the Euclidean distance is used to compute the distance between the prototypes and input samples to determine similarity. On the other hand, ProtoPGTN utilizes a Gated Transformer Network (GTN) for feature extraction and implements cosine similarity for prototype matching, which results in better and more discriminative prototypes. In addition, ProtoPGTN utilizes several mechanisms to reduce memory usage and training time. While

both methods achieve similar accuracy, ProtoPGTN is more efficient in terms of training time and memory.

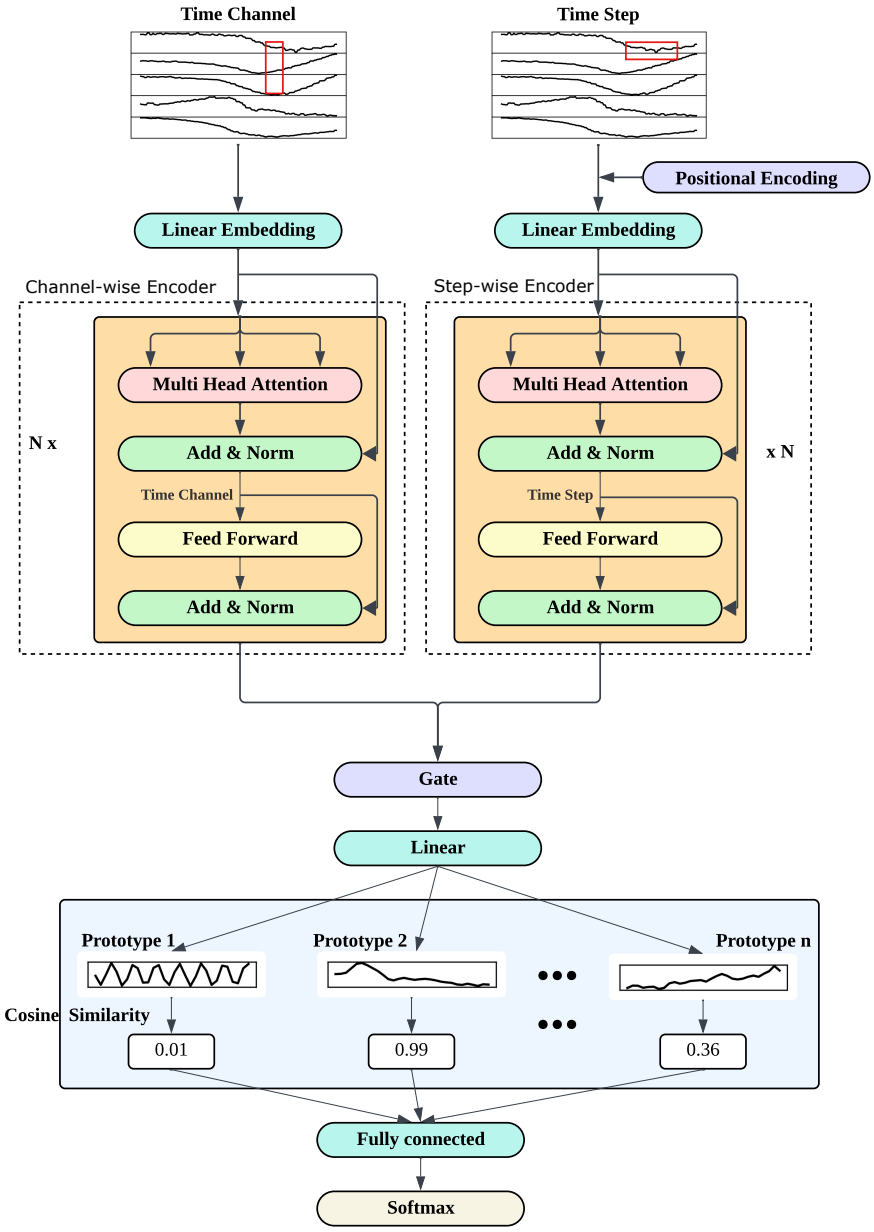

Figure 1. The architecture of ProtoPGTN. The inputs are passed to two separate transformer encoders to capture temporal and spatial features, with positional encoding added to the step-wise encoder. The prototype layer learns k prototypes per class to explain the decision-making process resulting in a total of n prototypes. The similarity score between learned prototypes and encodings is calculated. This similarity score is passed to a fully connected layer followed by a softmax activation for final prediction output.

3.1.1. Gated Transformer Network

Multivariate time series datasets consist of multiple channels, and each dimension records a series of data points in temporal order. Cross-channel information is also crucial for decision-making, since they are collected simultaneously and often exhibit interdependencies. Gated Transformer Network (GTN) [10] was motivated by this concept, including the design of a two-tower transformer architecture consisting of two encoders (step-wise f_{st} and channel-wise f_{ch}) to capture spatial and temporal features of the dataset.

Table 3. Comparison of the existing prototype-based interpretable model (ProtoPLSTM) and the proposed ProtoPGTN. ✓indicates the presence of the feature and ✗indicates absence.

Aspect	**ProtoPLSTM [9]**	**ProtoPGTN**
Linear layer before feature extraction	✗	✓, reduces memory usage
Feature extraction	2D Conv + LSTM	GTN
Gate mechanism	✗	✓, fuses temporal and spatial results
Similarity measure	Euclidean distance	Cosine similarity
Prototype dimension	2D	1D
Efficient prototype matching	✗	✓, direct projection to prototype shape
Evaluation scope	One dataset (SisFall)	165 datasets (UCR/UEA)
Accuracy	Comparable on UCR/UEA	Comparable to ProtoPLSTM
Training time	Slow	$\sim 8\times$ faster on average
GPU memory usage	Extremely high	Highly efficient ($\sim 379\times$ reduction on average)

Step-Wise Encoder

The step-wise encoder f_{st} captures temporal dependencies within each channel by processing the sequence of data points in temporal order. Given the raw time series dataset $X \in \mathbb{R}^{B\times L\times C}$, where B is the batch size, L is the sequence length, and C refers to the number of channels, the data are first projected into d_{model}-dimensional embedding space so that the dimension changed to $X \in \mathbb{R}^{B\times L\times d_{\text{model}}}$. Here, d_{model} refers to the dimension of the embedding space, which determines the size of the feature vectors used by the transformer model for encoding temporal dependencies. Because the dataset is collected sequentially over time, positional encoding is applied to preserve to preserve temporal information. This is followed by a stack of N identical multi-head attention layers. These layers compute scaled dot-product attention across time steps, with a mask applied during training to prevent the model from attending to future time steps. Each multi-head attention layer consists of h parallel attention heads, where each head computes attention independently using different query, key, and value matrices, and their outputs are concatenated and linearly transformed. As in traditional transformer encoders, the multi-head attention layer is followed by a position-wise feedforward network, and layer normalization and residual connections are applied after each multi-head attention layer and feedforward network.

Channel-Wise Encoder

The channel-wise encoder has the same architecture as the step-wise encoder but without positional encodings and attention mask, as there is no inherent order across channels. The input is reshaped by swapping sequence length and number of channels to be $X \in \mathbb{R}^{B\times C\times L}$, where C represents the number of channels and L is the sequence length. This reshaping allows attention to be computed across channels for all time steps, enabling the model to capture dependencies across channels throughout the sequence.

Gate Layer

After obtaining the encoded representations, the gate layer integrates the two resulting encodings. The encoded outputs are flattened to have dimensions step-wise encoding $\mathbf{E}_t \in \mathbb{R}^{B\times(L\cdot d_{\text{model}})}$ and channel-wise encoding $\mathbf{E}_c \in \mathbb{R}^{B\times(C\cdot d_{\text{model}})}$, respectively. Rather than simply concatenating the encodings, Liu et al. [10] proposed the gating mechanism. The gating mechanism passes the concatenated encodings through a linear layer that produces two scalar logits, each corresponding to one encoding type (step-wise and channel-

wise). Softmax is then applied to these two logits, producing two probabilities, g_t and g_c, which serve as the gating coefficients, with the constraint that $g_t + g_c = 1$. Simple concatenation treats spatial and temporal encodings equally. However, some samples may exhibit stronger connections on temporal sequences and others may rely more on cross-channel dependencies. The gating mechanism learns weights from input sequences that reflect the importance of each encoding type. This allows GTN to adaptively balance temporal and cross-channel information based on input characteristics rather than relying on a fixed combination. Thus, the resulting encodings are more representative of temporal-spatial correlations. The final fused representation is obtained by weighting each encoding by its corresponding gating coefficient to each encoding and concatenating the results, as shown in the following formula:

$$\mathbf{E}_{\text{fused}} = [g_t \cdot \mathbf{E}_t \parallel g_c \cdot \mathbf{E}_c].$$

The memory consumption of another prototype-based model, ProtoPLSTM, increases more rapidly as the number of variables increases. This is mainly because in ProtoPLSTM, the LSTM feature extractor flattens both the channel and variable dimensions before passing the resulting representations to a GRU. As a result, the GRU input and hidden dimensions scale linearly in the number of variables, leading to quadratic growth in parameter size and intermediate activations. In contrast, ProtoPGTN projects each input sequence into a fixed embedding dimension before applying attention, which decouples the memory requirement from the number of variables. This design makes ProtoPGTN significantly more memory-efficient and scalable to high-dimensional datasets.

3.2. ProtoPNet-Based Network

The previous gating layer results in a fused encoding of dimension $E_{\text{fused}} \in \mathbb{R}^{B\times(L\cdot d_{\text{model}}+C\cdot d_{\text{model}})}$, implying that each multivariate sequence is represented as a single flattened vector. Therefore, the prototypes need to be 1D vectors, each with a fixed length of 128. The number of prototypes per class in ProtoPGTN is fixed at 5. Rather than extracting subsequences from E_{fused} to align with the prototype shape, we directly project E_{fused} into $\mathbb{R}^{B\times 128}$. This means that each prototype in the prototype layer g_p computes a similarity score with the entire projected fused encoding. This operation can save a significant amount of training time, since it avoids computationally expensive sliding-window operations over the fused encoding, which requires repeated similarity calculations at each possible position. While the original ProtoPNet and its variants use Euclidean distance, we use the cosine similarity, since it better suits TSC. This is because time series data emphasize similarity in patterns, trends, and cycles rather than absolute value differences. For example, some signals can differ in magnitude and still share some similarities in their temporal dynamics. In this case, the Euclidean distance is significant, but the cosine similarity remains invariant to magnitude differences. In addition, cosine similarity combined with high-dimensional encodings produced by the transformer provides additional numerical stability. This is because when both $\mathbf{x}$ and $\mathbf{y}$ are normalized, the similarity is bounded in $[-1, 1]$. This avoids the issue of unbounded growth in the Euclidean distance. For visualization, the prototypes are periodically projected onto their nearest training samples during training. All training samples are passed through the two transformer encoders to obtain their latent feature representations. In this way, each prototype and training sample lies in the same latent feature space. Each prototype then computes the cosine similarity between itself and all training samples' latent feature representations, then is replaced by

the latent feature representation with the highest similarity. This projection is performed every five epochs. The update rule is defined as follows:

$$p_j \leftarrow x_{i^*}, \quad \text{where} \quad i^* = \arg\max_{i \in \{1,\ldots,N\}} \frac{x_i^\top p_j}{\|x_i\|_2 \, \|p_j\|_2}, \quad j = 1, \ldots, k \tag{1}$$

where p_j denotes to the j-th prototype, k is the total number of prototypes, and x_i denotes the latent feature representation of the i-th training sample. The prototypes are fixed after training. During inference, the similarity scores between input samples and prototypes in the latent feature space are passed to a fully-connected layer followed by a softmax activation to produce class probabilities.

3.3. Training and Inference

The proposed architecture can be trained end-to-end with prototype projection. The loss function is the standard classification cross-entropy loss. The Adam optimizer is employed due to its adaptive learning rate. During training, an attention mask is applied to prevent the model from attending to future time steps. This mask is disabled during inference. Dropout is applied inside the transformer encoder, specifically after the self-attention mechanism and the feed-forward networks. This regularization technique helps to reduce overfitting and improve generalization by randomly dropping units during training. To prevent test data leakage, the model is saved based on the highest validation accuracy achieved during training. The validation dataset is separated from the training dataset and used solely for model selection. Importantly, the test dataset is never used during training to maintain its role as a final evaluation set. The complete training and inference procedure is outlined in Algorithm 1.

Algorithm 1 Training and Inference Procedure for ProtoPGTN

Training process
Input: The time series dataset $X \in R^{B \times C \times L}$ along with its ground truth y
Output: Trained ProtoPGTN model parameters and prototypes.

Reshape the training dataset into $R^{B \times L \times C}$
Randomly initialize parameters w;
for $epoch = 1$ to 100 **do**
    Learn temporal and spatial feature representations through step-wise and channel-wise transformer encoders.
    Calculate similarity scores with prototypes using cosine similarity.
    Pass the output through a fully connected layer followed by a softmax activation to obtain class probabilities.
    Update w by minimizing the cross-entropy loss function in one batch.
    if $epoch$ mod 5 == 0 **then**
        For each prototype in P, project it onto its nearest training feature representation.
    end if
end for
Return trained parameters w;

Inference process
Input: The test dataset X_{test} and the trained model.
Output: Predicted class labels for all test samples.

for all $X^{(i)}$ **do**
    Compute output logits using the trained model.
    Output the class of the $X^{(i)}$.
end for

4. Evaluation

4.1. Experimental Settings

4.1.1. Dataset

To comprehensively evaluate the performance of our model, we evaluated it on datasets with varying dimensionalities, sequence lengths, and domains. Middlehurst et al. [34] maintain several repositories of benchmark datasets for time series machine learning. They provide a unified format for all Time Series Classification (TSC) datasets hosted on the website (https://www.timeseriesclassification.com/index.php (accessed on 30 November 2025)). The TSC dataset collection includes the UCR Time Series Archive [35], the UEA Multivariate Time Series Dataset [36], and several additional datasets contributed by individual researchers. Originally introduced in 2002 with only 16 datasets, the UCR Archive has since expanded to 128 datasets as of 2019. However, because the UCR Archive only contains univariate datasets, the UEA collection was later introduced to include 30 multivariate datasets. In total, the collection comprises 189 datasets, with only one lacking documentation of its original data source. Each dataset is divided into training and test sets, and datasets with unequal series lengths are padded accordingly. The datasets originate from diverse sources, with sequence lengths ranging from 8 to 236,784. This enables a comprehensive evaluation of model performance. While the majority are univariate datasets with only one channel, some multivariate datasets contain as many as 1345 channels. Among the 190 datasets, 165 were selected for evaluating ProtoPGTN's performance, comprising 35 multivariate and 130 univariate datasets. The remaining 25 datasets were excluded due to missing data or very small/large sample sizes that make them impractical to process. This provides a comprehensive and standardized benchmark for assessing TSC models.

4.1.2. Hyperparameter Tuning

ProtoPGTN involves several key hyperparameters. Given the scale of over 150 datasets, exhaustive tuning of all hyperparameters was computationally infeasible. Therefore, the majority of hyperparameters were set to their default values, as shown in Table 4, while two hyperparameters—d_{model} and the random seed—were tuned. These default settings are a common setup in machine learning experiments and are sufficient to achieve reasonable performance across all datasets. The choice of tuning only d_{model} and the random seed was motivated by both sensitivity and computational feasibility: d_{model} directly controls the model's capacity to learn expressive representations, while the random seed significantly influences reproducibility. K-fold cross-validation was employed to evaluate different hyperparameter configurations, with each dataset partitioned into k folds. In this study, 5-fold cross-validation was used, with the dataset split into five equal parts. In each iteration, $k-1$ folds were used for training, while the remaining fold was used for validation. This process was repeated k times, ensuring that each sample serves as validation exactly once. The final model performance was then obtained by averaging the results over all k folds. Candidate values for d_{model} and the random seed were predefined to determine their optimal configurations. Specifically, d_{model} was chosen from $\{64, 128, 256\}$, while the random seed was set to either 0 or 21. For each dataset, k-fold cross-validation was applied to every combination of hyperparameter settings and the combination yielding the highest accuracy was selected for final testing. The predefined search space and fixed random seeds ensured the reproducibility of the hyperparameter tuning process.

Table 4. Hyperparameters used in ProtoPGTN and their default configurations.

Parameter	**Default Value**
Model Architecture Parameters	
d_{model}	tuned
Number of layers	4
Number of heads	4
Hidden units	128
Query/Value dimension	8
Dropout rate	0.1
Positional encoding	Yes
Mask	Yes
Number of prototypes per class	5
Feature dimension	128
Training Parameters	
Batch size	32
Epochs	100
Projection interval	5
Random seed	tuned
Learning rate	0.001

4.1.3. Evaluation Metric

Interpretability

Interpretability is considered an important evaluation criterion. Although interpretability is inherently subjective and difficult to quantify, it is evaluated by the extent to which the model provides meaningful human-understandable explanations of its decision-making process. Thus, we adopt the same methods that ProtoPNet uses to interpret the decision-making process. Prototypes from different classes are visualized to compare inter-class dissimilarity and intra-class similarity. The decision process is based on the similarity between input sequences and learned prototypes.

Performance

Accuracy is used as the primary evaluation metric for time series classification. We compared our model with five other methods, including two prototype-based interpretable models. The interpretable models serve as comparable baselines that balance accuracy and interpretability. In addition, the MLP baseline defines a lower bound on accuracy due to its relatively simple architecture. The other two models serve as the upper bound, as they do not trade accuracy for interpretability. All experiments were conducted on either an NVIDIA Tesla A30 (24 GB) GPU or an NVIDIA L4 Tensor Core (24 GB) GPU. Models used for comparison were as follows:

- TimesNet [11] (non-interpretable): TimesNet represents the current state of the art in TSC, achieving the highest accuracy on the UEA benchmark. Motivated by the multi-periodicity of time series, it transforms 1D series into 2D tensors: columns capture intra-period variations, while rows capture inter-period variations, enabling the use of 2D kernels.
- Zerveas [12] (non-interpretable): This method employs a transformer encoder architecture to learn representations of multivariate time series, which can then be used in downstream tasks such as classification and regression.

- MLP [13] (non-interpretable): The MLP baseline consists of three fully connected layers, each with 500 neurons, and employs Rectified Linear Unit (ReLU) activation functions.
- ProtoPLSTM [9]: ProtoPLSTM integrates an LSTM encoder with prototype-based learning, offering interpretability alongside predictive performance. It combines three key modules: a CNN-LSTM encoder for feature extraction, a contextual enhancement module for richer representations, and a ProtoPNet-based interpretability layer. However, the original work evaluated the model only on the SisFall dataset, and the released implementation was hard-coded for that dataset. To extend its applicability, we modified the codebase by making layer dimensions dynamically adjustable to fit different input datasets.
- ProtoPConv: ProtoPConv is a simplified variant of ProtoPLSTM that removes the LSTM component. It employs three convolutional layers as the feature extractor before the prototype layer.

Scalability

Training efficiency is another important evaluation aspect. Training time per epoch was recorded under the same hardware conditions. The hardware used to obtain training time is shown in Table 5. In addition, GPU memory usage is employed as an additional metric to assess model scalability. ProtoPLSTM exhibits extremely high GPU memory consumption on high-dimensional datasets, making it impractical to record real-time memory usage. Therefore, we estimated the memory usage of ProtoPLSTM based on its architecture. The memory is the sum of parameters, activations, gradients, and optimizer states. The recurrent module (GRU) is the main contributor to the memory usage, since its parameter count grows with the input channels d. Thus, as the input channels increase, the memory usage of ProtoPLSTM increases significantly. The full derivation of the ProtoPLSTM memory estimation is provided in Appendix A.

Table 5. Hardware configuration used for recording the training time.

Category	**Specification**
Hardware	
GPU	NVIDIA A30 (24 GB)
Number of GPUs	1
CPU	AMD EPYC 7702P (32 cores)
System Memory (RAM)	111GB
Storage	180 GB SSD (root: 35 GB, var: 20 GB, data: 121 GB)
Software Environment	
Operating System	Ubuntu 24.04.2 LTS (Noble)
CUDA/cuDNN	CUDA 12.8

4.2. Interpretability

4.2.1. Prototypes Learned for Classes

During training, ProtoPGTN learns a set of five prototypes per class ($k = 5$). For visualization, each prototype is projected onto its nearest training sample in the latent space. Figure 2 illustrates the learned prototypes on the multivariate `ArticularyWord Recognition` dataset. The dataset contains 25 classes, showing prototypes for classes 0, 1, 6, and 17. The subcaption of each panel indicates the prototype ID. The legend specifies the source, including the training sample index, channel, and temporal window from which the matched segment is extracted. For example, prototype 62 is matched to training sample 9, channel 5, window [30, 60). The learned prototypes appear well-formed and semantically

meaningful, as those within the same class share similar temporal patterns. For example, class 0 prototypes exhibit a monotonically increasing pattern, whereas class 1 prototypes display a parabolic pattern. Moreover, prototypes from different classes exhibit marked differences. This suggests that the model captures class-specific structure, facilitating accurate classification.

Figure 2. Prototypes learned for representative classes (0, 1, 6, and 17) from `ArticularyWordRecognition` dataset. Each colored row corresponds to one class, and each row contains three of five learned prototypes. Each prototype is visualized using nearest training segments and the segment information is shown on the legend indicating sample index, channel and temporal window. Prototypes within the same class exhibit similar patterns, whereas prototypes across different classes demonstrate distinct temporal dynamics. This suggests that ProtoPGTN can learn representative prototypes, contributing to both interpretability and accurate final decision-making.

4.2.2. Explaining Decision-Making Process

After training, the prototypes remain fixed. During inference, each test sample is compared against all prototypes and its class is determined by the resulting similarity scores. Figure 3 illustrates the reasoning process for a test sample from the `ArticularyWordRecognition` dataset belonging to class 0. The top panel shows an example test sample from class 0, which consists of nine channels. The GTN converts the sample to latent features, and these features are compared against learned prototypes. Specifically, for each class, the model seeks evidence that the input belongs to that class by comparing latent features with prototypes that belong to that class. For example, as shown in Figure 3 (left), the network demonstrates that the test sample belongs to class 0 by comparing its latent features with class-0 prototypes. The similarity score is computed as cosine similarity between the sample's latent features and each prototype. The red box highlights the most similar subsequence from a channel to the prototype. The figure clearly shows that

similarity scores for class 0 prototypes are much higher (0.9598, 0.9480, and 0.9679), whereas those for class 1 prototypes are considerably lower (0.1053, 0.0942, and 0.0982) in Figure 3 (right). This indicates that the model's decision is well supported by the learned prototypes.

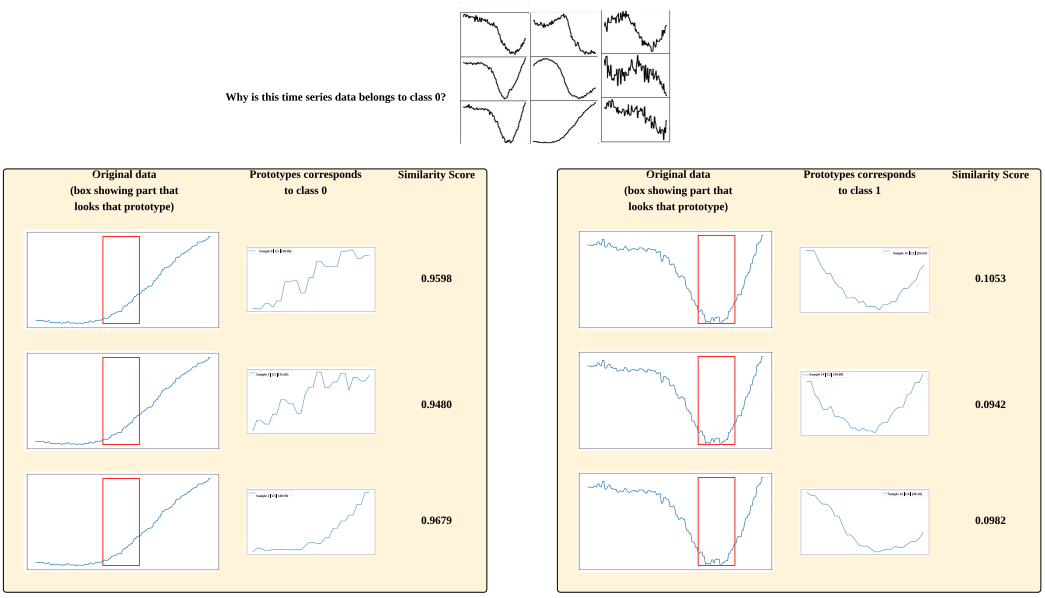

Figure 3. Illustration of reasoning process of the proposed network for a sample from `ArticularyWordRecognition` dataset. The top panel shows the original test sample, which belongs to class 0 and contains nine channels. Two blocks below compare the latent features of this test sample with learned prototypes from class 0 (left) and class 1 (right). Within each block, the first column represents one channel from original sample, with the red box highlighting the subsequence that is most similar to the prototype learned in the second column. The third column displays the cosine similarity score computed in the latent space between them. The class-0 prototypes achieve significantly higher similarity scores (0.9598, 0.9480, 0.9679) than class-1 prototypes (0.1053, 0.0942, 0.0982). This demonstrates that the model's decision-making process is interpretable and evidence-based.

4.3. Accuracy

4.3.1. Multivariate TSC

The classification accuracy results of different models on multivariate TSC datasets are summarized in Table 6. A total of 35 multivariate datasets are included in the benchmark. The MLP baseline serves as the lower bound because of its simplicity. It consistently obtains poor performance across nearly all datasets. However, MLP can still achieve non-trivial performance on a few datasets, such as 13.33% on `StandWalkJump` and 7.98% on `DuckDuckGeese`. Compared with the prototype-based ProtoPLSTM interpretable model, ProtoPGTN achieves higher accuracy on 19 out of 35 datasets (60%), including cases where ProtoPLSTM fails to produce results, while ProtoPLSTM outperforms ProtoPGTN on only ten datasets. Substantial improvements are observed in datasets such as `ERing` and `HandMovementDirection`. In general, ProtoPGTN attains performance that is comparable to or slightly better than, ProtoPLSTM. However, the minimum accuracy drops to 2.56% on `PhonemeSpectra`, suggesting that the performance of ProtoPGTN can be unstable and that its robustness still requires improvement. Compared to ProtoPConv, which relies solely on convolutional layers for feature extraction, ProtoPGTN achieves substantially higher accuracy, indicating that the GTN backbone is more effective at capturing discriminative features. TimesNet and Zerveas, which do not sacrifice accuracy for interpretability, serve as upper-bound baselines. The performance gap between ProtoPGTN and the upper-bound baselines varies across datasets. For example, ProtoPGTN achieves comparable accuracy to TimesNet on datasets such as `EMOPain`, `EthanolConcentration`, and `FaceDetection`. In

addition, ProtoPGTN even matches or slightly surpasses the non-interpretable baselines on certain datasets, such as `MotionSenseHAR`, `PenDigits`, and `SpokenArabicDigits`. On the other hand, larger gaps are observed on datasets such as `Epilepsy` and `MindReading`.

Table 6. Accuracy comparison across multivariate time series datasets. Results in bold mean higher accuracy between ProtoPLSTM and ProtoPGTN. Missing values for ProtoPLSTM and ProtoPConv are due to excessive GPU memory usage or invalid convolution kernel settings on certain datasets. For Zerveas's transformer-based model, missing results also occur when GPU memory is insufficient for large-channel datasets.

Dataset	ProtoPGTN	ProtoPLSTM	ProtoPConv	TimesNet	Zerveas	MLP
AsphaltObstaclesCoordinates	73.66%	**80.36%**	52.69%	78.01%	79.54%	0.00%
AsphaltPavementTypeCoordinates	80.49%	**88.26%**	84.19%	85.89%	80.78%	0.00%
AsphaltRegularityCoordinates	91.61%	**96.54%**	95.74%	95.87%	89.08%	0.13%
ArticularyWordRecognition	93.00%	**96.67%**	81.00%	98.00%	98.33%	1.00%
BasicMotions	**82.50%**	75.00%	62.50%	97.50%	100.00%	0.00%
Blink	55.56%	55.56%	77.33%	95.33%	95.78%	0.67%
CharacterTrajectories	88.65%	**98.47%**	97.21%	98.89%	98.89%	0.07%
Cricket	81.94%	**94.44%**	59.72%	90.28%	0.00%	1.39%
EMOPain	78.31%	78.31%	66.76%	81.13%	86.20%	0.85%
ERing	**68.52%**	13.33%	33.33%	81.48%	94.81%	1.11%
EthanolConcentration	28.14%	**28.90%**	25.10%	29.67%	–	0.76%
EyesOpenShut	50.00%	50.00%	50.00%	66.67%	64.29%	4.76%
FaceDetection	**65.81%**	–	65.32%	67.65%	67.68%	0.03%
DuckDuckGeese	**36.00%**	–	24.00%	58.00%	70.00%	7.98%
JapaneseVowels	**95.41%**	–	–	98.38%	99.46%	0.81%
MotionSenseHAR	92.45%	**98.87%**	95.09%	92.08%	95.85%	1.13%
RacketSports	**81.58%**	–	–	84.21%	88.16%	0.66%
SelfRegulationSCP1	**88.05%**	80.89%	69.28%	90.44%	89.76%	0.00%
Handwriting	**13.88%**	3.41%	6.94%	30.94%	32.12%	0.00%
Epilepsy	74.64%	**91.30%**	72.46%	92.03%	97.83%	1.45%
AtrialFibrillation	33.33%	33.33%	33.33%	46.67%	46.67%	0.00%
FingerMovements	**45.00%**	–	–	57.00%	57.00%	2.00%
HandMovementDirection	**50.00%**	33.78%	17.57%	58.11%	67.57%	0.00%
Heartbeat	72.20%	72.20%	72.20%	78.05%	74.63%	0.00%
Libras	**53.89%**	–	–	75.56%	86.67%	2.78%
LSST	**54.62%**	–	–	37.99%	58.92%	0.04%
MindReading	**23.12%**	–	23.12%	41.50%	45.33%	0.15%
NATOPS	**76.11%**	–	–	83.89%	96.11%	0.06%
PEMS-SF	**52.60%**	–	32.95%	82.08%	84.97%	3.47%
PenDigits	**97.97%**	–	–	98.28%	97.68%	0.06%
PhonemeSpectra	2.56%	**18.43%**	14.52%	13.93%	8.44%	0.06%
SelfRegulationSCP2	**50.56%**	50.00%	50.00%	53.89%	55.00%	1.11%
SpokenArabicDigits	**98.04%**	96.77%	97.68%	98.18%	98.04%	0.09%
StandWalkJump	33.33%	33.33%	33.33%	48.00%	–	13.33%
UWaveGestureLibrary	**83.44%**	75.00%	13.75%	86.88%	86.56%	0.00%
Wins	**19**	10	–	–	–	–

4.3.2. Univariate TSC

A total of 130 univariate datasets are included, and Table 7 presents results for a representative subset of 30 datasets. The complete accuracy results are available in the accompanying code repository. The MLP baseline achieves higher accuracy on univariate datasets compared to its performance on multivariate tasks. The accuracy of MLP is only slightly lower than ProtoPGTN, TimesNet, and Zerveas. Moreover, MLP achieves near-perfect results on certain datasets, such as `Coffee`, `PowerCons` (100%), and `GunPointMaleVersusFemale` (99.68%). ProtoPGTN also performs well in univariate settings; its performance gap against non-interpretable baselines is reduced to only 1.4% (TimesNet) and 3.3% (Zerveas). Compared to ProtoPConv, ProtoPGTN achieves over 20% higher average accuracy. ProtoPGTN outperforms ProtoPLSTM on 24 datasets, with roughly 15% higher average accuracy. ProtoPLSTM underperforms relative to the other models except ProtoPConv, with the lowest average accuracy of 62.75%. Nevertheless, it achieves competitive or even superior results in some cases, such as `GesturePebbleZ`, `LargeKitchenAppliances`, and `Computers`. TimesNet and Zerveas perform very similarly

on average, with Zerveas having the highest accuracy (80.53%). Finally, some datasets are challenging for all models. For example, `Haptics` yields relatively low accuracy across multiple models (40.91% for ProtoPGTN and 48.38% for Zerveas).

Table 7. Accuracy comparison across univariate time series datasets. Results in bold mean higher accuracy between ProtoPLSTM and ProtoPGTN. Missing values from ProtoPLSTM and ProtoPConv are due to the invalid kernel size.

Dataset	ProtoPGTN	ProtoPLSTM	ProtoPConv	TimesNet	Zerveas	MLP
BeetleFly	**90.00%**	65.00%	50.00%	85.00%	95.00%	85.00%
BME	**96.00%**	38.67%	51.33%	64.00%	90.00%	96.67%
Car	**70.00%**	60.00%	31.67%	78.33%	85.00%	81.67%
CBF	**86.44%**	82.67%	33.33%	83.89%	98.33%	86.78%
Coffee	**100.00%**	60.71%	53.57%	89.29%	96.43%	100.00%
Computers	54.80%	**62.00%**	66.80%	65.60%	69.60%	53.60%
Crop	**73.77%**	–	–	75.72%	74.88%	66.92%
DistalPhalanxTW	**58.27%**	30.22%	67.63%	71.94%	71.22%	66.19%
ECG200	**81.00%**	80.00%	68.00%	84.00%	90.00%	90.00%
ECG5000	**92.49%**	91.96%	92.02%	94.09%	94.13%	92.22%
ElectricDeviceDetection	**85.93%**	83.97%	64.30%	86.28%	86.31%	81.82%
FaceAll	**81.42%**	68.99%	63.91%	77.40%	75.50%	81.30%
FaceFour	**73.86%**	25.00%	15.91%	82.95%	84.09%	85.23%
Haptics	**40.91%**	37.66%	19.16%	44.48%	48.38%	42.86%
FreezerSmallTrain	**76.63%**	53.54%	51.16%	76.70%	77.02%	67.61%
GesturePebbleZ2	61.39%	**70.89%**	34.18%	85.44%	75.95%	59.49%
GunPointMaleVersusFemale	**96.52%**	52.53%	47.47%	99.05%	100.00%	99.68%
Ham	**82.86%**	64.76%	67.62%	79.05%	75.24%	71.43%
Lightning2	**70.49%**	60.66%	45.90%	77.05%	77.05%	68.85%
MixedShapesSmallTrain	**72.04%**	56.29%	38.56%	82.39%	83.55%	84.49%
Plane	**93.33%**	88.57%	73.33%	98.10%	99.05%	97.14%
PowerCons	**100.00%**	99.44%	93.33%	99.44%	100.00%	100.00%
ProximalPhalanxOutlineAgeGroup	**85.37%**	48.78%	83.90%	85.37%	86.83%	82.93%
ShapeletSim	47.22%	**50.00%**	53.89%	56.11%	53.89%	48.33%
SonyAIBORobotSurface1	**88.02%**	42.93%	42.93%	77.04%	78.04%	65.56%
SonyAIBORobotSurface2	**87.20%**	61.70%	68.10%	82.27%	83.00%	83.84%
UWaveGestureLibraryX	73.03%	**77.39%**	49.69%	72.39%	72.17%	76.66%
WormsTwoClass	**59.74%**	54.55%	55.84%	64.94%	67.53%	59.74%
LargeKitchenAppliances	56.27%	**76.00%**	79.47%	60.00%	53.07%	48.00%
Earthquakes	74.82%	74.82%	74.82%	76.98%	74.82%	70.50%
Number of Wins	24	5	–	–	–	–

The evaluated datasets contain a wide range of scales, from very small training samples to large-scale datasets. For example, in the multivariate datasets, `ERing` and `BasicMotions` contain only 30 and 40 training samples, respectively. Similarly, univariate datasets such as `BeetleFly` (20), `BMF` (30), `CBF` (30), and `Coffee` (28) also have small training sets. However, ProtoPGTN performs well on these datasets, achieving higher accuracy than ProtoPLSTM. In addition, ProtoPGTN can also outperforms ProtoPLSTM on large datasets such as `PenDigits` and `SpokenArabicDigits`, which have 7494 and 6599 training samples, respectively.

4.4. Training Time

The training time per epoch for ProtoPLSTM and ProtoPGTN was measured under the same hardware configuration. This setup ensured that the comparisons are conducted under equivalent training conditions. TimesNet and Zerveas are non-interpretable and not prototype-based, so their training times are not included in the comparison. Because MLP is a relatively simple model that requires minimal training time, its detailed training time is ignored. The same is the case for ProtoPConv, since its feature extraction is relatively simple.

Table 8 reports the detailed training times for multivariate datasets. The results clearly show that ProtoPGTN consistently outperforms ProtoPLSTM across all datasets. The performance gap is more significant on large-scale datasets. For example, ProtoPLSTM requires approximately 227 s to train one epoch on the `PhonemeSpectra` dataset, whereas

ProtoPGTN only needs approximately 11 s. This corresponds to roughly a 20× reduction in training time. Similarly, ProtoPGTN trains about ten times faster on several datasets, such as `EthanolConcentration`, `MotionSenseHAR`, and `Heartbeat`. ProtoPGTN also maintains a clear advantage on small-scale datasets such as `BasicMotions`, `ERing`, and `EyesOpenShut`.

Table 8. Detailed training time in seconds (s) for ProtoPGTN and ProtoPLSTM on multivariate datasets. Missing values are due to memory limitations and invalid kernel sizes. The bold values indicate faster training time.

Dataset	ProtoPGTN	ProtoPLSTM
AsphaltObstaclesCoordinates	**0.9839**	1.8569
AsphaltPavementTypeCoordinates	**2.1802**	4.3391
AsphaltRegularityCoordinates	**1.4258**	2.3824
ArticularyWordRecognition	**1.0612**	6.5625
BasicMotions	**0.5367**	0.5922
Blink	**1.4250**	2.6275
CharacterTrajectories	**2.5115**	6.4605
Cricket	**2.7315**	9.6463
EMOPain	**1.4805**	14.0935
ERing	**0.5438**	0.6205
EthanolConcentration	**0.5211**	6.1584
EyesOpenShut	**0.5373**	0.7162
FaceDetection	**6.0796**	–
DuckDuckGeese	**1.5141**	–
JapaneseVowels	**0.7107**	–
MotionSenseHAR	**4.0186**	68.0326
RacketSports	**0.5967**	–
SelfRegulationSCP1	**1.5961**	3.6851
Handwriting	**0.8535**	1.9227
Epilepsy	**0.6433**	0.9020
AtrialFibrillation	**0.4843**	1.6728
FingerMovements	**0.6868**	–
HandMovementDirection	**0.7891**	2.9576
Heartbeat	**0.9343**	10.8406
Libras	**0.7334**	–
LSST	**2.4925**	–
MindReading	**2.0731**	–
NATOPS	**0.6762**	–
PEMS-SF	**1.7808**	–
PenDigits	**5.6463**	–
PhonemeSpectra	**11.3748**	227.0682
SelfRegulationSCP2	**0.6953**	3.9527
SpokenArabicDigits	**6.2297**	16.5668
StandWalkJump	**0.9033**	1.0293
UWaveGestureLibrary	**0.7198**	1.5138
Number of Wins	**35**	0

Table 9 reports the per-epoch training times for the univariate datasets. This table only shows those datasets for which both models produced training time results. For the majority of datasets, ProtoPGTN demonstrates a clear advantage in training efficiency. For instance, the training time for ProtoPGTN on the `ECG5000` dataset is only 0.8 s, which is nearly 3× faster than ProtoPLSTM (2.16 s). Smaller but consistent improvements are also observed across several other datasets. For example, ProtoPGTN saves approximately 0.2 s per epoch on `SonyAIBORobotSurface1/2`, `FaceFour`, and `GesturePebbleZ2` datasets. However, there are a few cases cases where ProtoPLSTM exhibits slightly higher efficiency.

For example, on the `Computers` and `LargeKitchenAppliances` datasets, ProtoPLSTM trains slightly faster than ProtoPGTN. There are also datasets on which the two models exhibit nearly identical training times, such as `Coffee` and `Earthquakes`.

Table 9. Detailed training time in seconds (s) for ProtoPGTN and ProtoPLSTM on univariate datasets. Only datasets for which both models produced training time results are shown. The bold values indicate faster training time.

Dataset	ProtoPGTN	ProtoPLSTM
BeetleFly	**0.5328**	0.5660
BME	**0.5385**	0.6015
Car	**0.6966**	0.7052
CBF	**0.5469**	0.6645
Coffee	0.5253	**0.5151**
Computers	1.2655	**0.9348**
DistalPhalanxTW	0.8235	**0.7525**
ECG200	**0.5170**	1.0160
ECG5000	**0.7974**	2.1640
ElectricDeviceDetection	**1.0514**	1.3032
FaceAll	**1.2722**	1.4824
FaceFour	**0.5430**	0.7160
Haptics	1.9317	**1.5354**
FreezerSmallTrain	**0.5461**	0.9043
GesturePebbleZ2	**0.8747**	1.0640
GunPointMaleVersusFemale	**0.6117**	0.7709
Ham	**0.6375**	0.8147
Lightning2	**0.6237**	0.9095
MixedShapesSmallTrain	1.5439	**1.4716**
Plane	**0.6202**	0.8274
PowerCons	**0.6474**	0.8431
ProximalPhalanxOutlineAgeGroup	**0.7405**	0.8287
ShapeletSim	**0.5200**	0.7512
SonyAIBORobotSurface1	**0.5108**	0.7453
SonyAIBORobotSurface2	**0.5182**	0.7689
UWaveGestureLibraryX	**2.3779**	3.0240
WormsTwoClass	1.2762	**1.0682**
LargeKitchenAppliances	1.8021	**1.5819**
Earthquakes	1.1020	**1.0097**
Number of Wins	**21**	8

4.5. Memory Usage

The memory usage of both models was evaluated on selected high-dimensional datasets, as shown in Table 10. ProtoPGTN consumed less GPU memory across all 18 datasets. The gap in memory consumption between ProtoPGTN and ProtoPLSTM becomes increasingly pronounced as the number of variables grows. For example, ProtoPGTN consumed only 1490.77 MB of GPU memory on `DuckDuckGeese` with 1345 variables, whereas ProtoPLSTM was estimated to consume over 5,000,000 MB (≈5 TB), making it impossible to run on modern GPUs. Similarly, on `PEMS-SF` (963 variables) and `MindReading` (204 variables), ProtoPLSTM reaches memory usage in the millions of megabytes, while ProtoPGTN remains below 2000 MB. The same pattern holds for datasets with moderate dimensionality. For instance, `FingerMovements` and `EMOPain` have 28 and 30 variables, respectively. ProtoPLSTM requires over 2000 MB to train, whereas ProtoPGTN requires under 100 MB of memory. The difference is less extreme on lower-dimensional datasets such as `JapaneseVowels`, `RacketSports`, and `PhonemeSpectra`, but still consistently favors

ProtoPGTN. These results highlight ProtoPGTN's robustness on high-dimensional datasets and its scalability.

Table 10. Memory usage (MB) comparison between ProtoPGTN and ProtoPLSTM across selected high-dimensional datasets. The smaller value in each row is highlighted in bold.

Dataset	ProtoPGTN	ProtoPLSTM	No. Variable	Reduction Factor
ArticularyWordRecognition	**103.60**	281.17	9	2.71
BasicMotions	**55.76**	138.78	6	2.49
EMOPain	**85.68**	2857.00	30	33.34
EthanolConcentration	**42.20**	53.40	3	1.27
EyesOpenShut	**25.75**	642.34	14	24.94
FaceDetection	**62.43**	62,267 (est.)	144	997.61
DuckDuckGeese	**1490.77**	5,440,971.19 (est.)	1345	3649.38
JapaneseVowels	**34.88**	435.54 (est.)	12	12.49
MotionSenseHAR	**460.78**	523.58	12	1.14
RacketSports	**35.15**	111.93	6	3.18
FingerMovements	**54.37**	2362.59	28	43.45
Heartbeat	**263.24**	11,776.62	61	44.75
LSST	**37.67**	112.12 (est)	6	2.98
MindReading	**323.94**	125,282.15 (est.)	204	386.72
NATOPS	**45.21**	1737.69 (est.)	24	38.44
PEMS-SF	**1774.83**	2,784,217.19 (est.)	963	1569.33
PhonemeSpectra	**180.03**	431.73	11	2.40
SpokenArabicDigits	**66.35**	574.74	13	8.66
Number of wins	**18**	0	–	–

4.6. Statistical Analysis

To verify whether the performance differences between ProtoPLSTM and ProtoPGTN are statistically significant, we conducted pairwise comparisons using the one-sided and two-sided Wilcoxon signed-rank tests with a significance level of $\alpha = 0.05$. The two-sided test was conducted on classification accuracy, whereas the one-sided test was applied to training time and memory usage. This is because we expected ProtoPGTN to achieve a similar level of accuracy as ProtoPLSTM while showing significantly improved efficiency and scalability. This non-parametric test was chosen because it does not assume normality and is suitable for comparing two related models across multiple datasets. For each test, the corresponding p-value is reported. A smaller p-value ($p < 0.05$) indicates a statistically significant improvement.

The results of the Wilcoxon signed-rank tests are shown in Table 11. For fairness, the test was conducted on datasets where both ProtoPLSTM and ProtoPGTN successfully produced valid results. Under this setting, the performance difference is not statistically significant in terms of multivariate accuracy. This is consistent with our previous conclusion on accuracy results: the accuracy of ProtoPGTN is comparable to that of ProtoPLSTM while achieving higher efficiency and scalability. For univariate accuracy, the difference is statistically significant, indicating that ProtoPGTN achieves better accuracy. Similarly, the one-sided Wilcoxon signed-rank test indicates that ProtoPGTN achieves statistically significantly lower training time and memory usage, as the p-values for both metrics are below 0.05.

Table 11. Results of the Wilcoxon signed-rank test comparing ProtoPGTN and ProtoPLSTM across different metrics. p-values below 0.05 indicate statistical significance. Bold values indicate statistical significance at $p < 0.05$.

Metric	Test Type	p-value
Accuracy (Multivariate)	two-sided	0.32587
Accuracy (Univariate)	two-sided	**0.00069**
Training Time (Multivariate)	one-sided (less)	**0.00000**
Training Time (Univariate)	one-sided (less)	**0.00691**
Memory Usage	one-sided (less)	**0.00000**

5. Ablation Analysis

To analyze the contribution of individual components of ProtoPGTN, we conducted an ablation study. The ablation study focused on two components: the gated transformer and cosine similarity. We isolated the gated transformer and added an additional linear layer on top so that it could be used directly for classification. In addition, we replaced the cosine similarity with the Euclidean distance in order to evaluate its effectiveness. Table 12 illustrates the results for the ablation test evaluated on multivariate datasets. When using the Euclidean distance, the average accuracy drops by approximately 5%. The original ProtoPGTN achieves higher accuracy on the majority of datasets when compared with the Euclidean distance variant. Great improvements are observed on datasets such as `AsphaltObstaclesCoordinates`, `Libras`, and `DuckDuckGeese`, suggesting that the cosine similarity is more suitable for time series classification.

Table 12. Ablation study results comparing ProtoPGTN, ProtoPGTN (Euclidean), and GTN. ProtoPGTN is the original proposed model, ProtoPGTN (Euc.) is the ProtoPGTN using Euclidean distance to replace cosine similarity, and the GTN classifier uses the representation learned by the two-tower transformer directly for classification. Bold values indicate the best performance among the three methods for each dataset.

Dataset	ProtoPGTN	ProtoPGTN (Euc.)	GTN
AsphaltObstaclesCoordinates	**73.66%**	49.10%	55.75%
AsphaltPavementTypeCoordinates	**80.49%**	66.57%	67.05%
AsphaltRegularityCoordinates	**91.61%**	81.76%	80.56%
ArticularyWordRecognition	93.00%	94.67%	**96.33%**
BasicMotions	82.50%	47.50%	**95.00%**
Blink	55.56%	55.56%	**90.22%**
CharacterTrajectories	88.65%	98.19%	**98.26%**
Cricket	81.94%	76.39%	**84.72%**
EMOPain	78.31%	79.15%	**84.23%**
ERing	68.52%	84.07%	**89.63%**
EthanolConcentration	28.14%	25.10%	**34.98%**
EyesOpenShut	50.00%	50.00%	50.00%
FaceDetection	65.81%	**67.57%**	65.30%
DuckDuckGeese	36.00%	20.00%	**58.00%**
JapaneseVowels	95.41%	94.86%	**98.38%**
MotionSenseHAR	92.45%	**94.34%**	93.58%
RacketSports	81.58%	76.32%	**83.55%**
SelfRegulationSCP1	**88.05%**	87.37%	84.30%
Handwriting	13.88%	12.12%	**19.18%**
Epilepsy	**74.64%**	68.12%	67.39%
AtrialFibrillation	**33.33%**	13.33%	26.67%
FingerMovements	45.00%	**62.00%**	47.00%
HandMovementDirection	**50.00%**	43.24%	39.19%

Table 12. *Cont.*

Dataset	ProtoPGTN	ProtoPGTN (Euc.)	GTN
Heartbeat	72.20%	**74.63%**	73.66%
Libras	53.89%	12.78%	**83.33%**
LSST	54.62%	**56.29%**	52.60%
MindReading	23.12%	23.12%	23.12%
NATOPS	76.11%	86.67%	**91.67%**
PEMS-SF	52.60%	12.72%	**85.55%**
PenDigits	**97.97%**	97.74%	96.91%
PhonemeSpectra	2.56%	6.29%	**9.22%**
SelfRegulationSCP2	50.56%	**52.78%**	50.56%
SpokenArabicDigits	**98.04%**	97.36%	97.95%
StandWalkJump	33.33%	33.33%	33.33%
UWaveGestureLibrary	83.44%	78.13%	**85.00%**
No. Wins	9	6	**20**

In addition, the gated transformer achieves higher accuracy when directly used for classification. The accuracy is higher than that of ProtoPGTN in general. This means that GTN can generate meaningful representations using two transformer encoders. However, ProtoPGTN intentionally introduces a prototype-based structure and cosine similarity metric to enhance interpretability. As a result, the performance of ProtoPGTN is slightly lower than the original GTN. This reduction is expected as a result of the accuracy–interpretability tradeoff. ProtoPGTN trades a small amount of accuracy to deliver transparent and human-understandable reasoning through prototype associations. In practice, interpretability may be more valuable than marginal gains in raw accuracy. Moreover, there are still conditions under which ProtoPGTN can achieve even better results than GTN itself.

6. Conclusions and Future Work

In this work, we propose ProtoPGTN, a prototype-based gated transformer network designed for interpretable time series classification. Our model utilizes the power of a gated transformer network to extract feature representations containing temporal and spatial information. The extracted feature representations help the model learn prototypes in prototype layers adapted from ProtoPNet. This hybrid architecture provides transparent explanations of how the decisions are made through learned prototypes with good performance.

Through comprehensive testing on UCR and UEA datasets, ProtoPGTN shows comparable performance to the existing ProtoPLSTM prototype-based interpretable model. It also demonstrates competitive accuracy against state-of-the-art non-interpretable models such as TimesNet and Zerveas. Importantly, ProtoPGTN remains highly efficient, obtaining faster training times and lower memory usage compared to other prototype-based interpretable models, which makes it more suitable for large-scale applications. Moreover, our interpretability analysis confirms that ProtoPGTN learns representative and class-discriminative prototypes. This enables users to trace the classification decisions back to representative patterns in the data, helping to ensure model safety and user trust in critical domains such as healthcare.

ProtoPGTN's interpretability and efficiency make it particularly suitable for real-world time series classification tasks such as healthcare monitoring, industrial fault detection, and financial anomaly detection. Through prototype-based explanations, domain experts can validate model predictions and gain insights into the temporal patterns that influence classification outcomes. In addition, the high efficiency of ProtoPGTN allows it to be

deployed in real-time environments where speed and resource constraints are critical. This advantage further broadens its applicability to real-world settings.

Future research is needed to address the limitations of ProtoPGTN. Although ProtoPGTN can obtain comparable accuracy with regard to non-interpretable models, there are still existing accuracy gaps. Future work can focus on narrowing the accuracy gap and further improving efficiency.

- Alternatives to cosine similarity: ProtoPGTN employs cosine similarity, which is sown to obtain better results than Euclidean distance. Future research could explore Dynamic Time Warping (DTW) as an alternative to cosine similarity. DTW measures the similarity between two temporal sequences of possibly different lengths.
- Alternatives to full attention: ProtoPGTN uses full self-attention, which requires computing pairwise interactions across all time steps and channels. Future research could explore more efficient variants of attention mechanisms, such as `linear attention` and `sparse attention`. These variants can approximate the full attention matrix while further reducing the computational and memory costs.
- Variable-length and hierarchical prototypes: Future research could investigate variable-length or hierarchical prototypes to capture multi-scale temporal dependencies. This could further enhance the model's expressiveness and interpretability by representing temporal and spatial dynamics more flexibly.

Author Contributions: Conceptualization, C.G.; funding acquisition, W.L.; methodology, J.H.; project administration, W.L.; software, J.H.; supervision, C.G.; validation, C.G.; visualization, J.H.; writing—original draft, J.H. and C.G.; writing—review and editing, J.H., C.G., and W.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the United Kingdom EPSRC (grant number UKRI256, EP/V028251/1, EP/N031768/1, EP/S030069/1, and EP/X036006/1).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available in the repositories mentioned in the Evaluation section.

Acknowledgments: The support of Altera, Intel, AMD and Google Cloud is gratefully acknowledged. We acknowledge the contributors of the datasets used in this study and express our gratitude to the maintainers of the UCR Time Series Archive and the UEA Multivariate Time Series Dataset for making these benchmarks publicly available.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Memory Estimation for ProtoPLSTM

We decompose the overall GPU memory usage into four components: (i) model parameters, (ii) activations, (iii) gradients, and (iv) optimizer states, all measured under float32 precision (4 bytes per element). The final GPU memory is the sum of these four components. Let d denote the number of channels (input variables), T the original sequence length, and L_ℓ the temporal length after the ℓ-th convolutional layer (kernel size $k = 5$, stride $s = 2$):

$$L_\ell = \left\lfloor \frac{L_{\ell-1} - k}{s} \right\rfloor + 1, \quad L_0 = T, \quad \ell = 1, \ldots, 4.$$

We denote L_4 as the final temporal length after the convolutional stack.

Parameters

The total number of parameters is computed per module and converted to megabytes by

$$\text{MB} = \frac{\#\text{params} \times 4}{1024^2}.$$

- Convolutional stack: A fixed constant determined by the implementation (215,680).
- GRU: $196{,}608\, d^2 + 1536\, d$ parameters.
- Context Block (GCB): Define $g = d\, L_4$. The parameter count is

$$\#\text{params}_{\text{GCB}} = 2g^2 + 7g + 1 = 2\,(dL_4)^2 + 7\,(dL_4) + 1,$$

  where the quadratic term dominates with respect to d when L_4 is fixed by the convolutional strides.
- Prototype layer: With C classes and P prototypes per class,

$$\#\text{params}_{\text{Proto}} = 128\, C\, P.$$

- Fully connected classifier:

$$\#\text{params}_{\text{FC}} = C\, P\, C + C.$$

Thus, the total number of trainable parameters is

$$\#\Theta = 215{,}680 + (196{,}608\, d^2 + 1{,}536\, d) + \left[2\,(dL_4)^2 + 7\,(dL_4) + 1\right] + 128\, C\, P + (C\, P\, C + C).$$

Activations

For a batch size B, the convolutional feature maps have shapes

$$(B,\, 32,\, d,\, L_1),\ (B,\, 64,\, d,\, L_2),\ (B,\, 128,\, d,\, L_3),\ (B,\, 256,\, d,\, L_4),$$

resulting in total activation memory

$$\sum_{\ell} \frac{4B \cdot \text{ch}_\ell \cdot d \cdot L_\ell}{1024^2}\ \text{MB}.$$

The GRU activations have shape $(L_4,\, B,\, 128d)$, the GCB activations $(B,\, 128,\, d)$, and the prototype maps $(B,\, CP,\, d,\, L_4)$. The final logits occupy $(B,\, C)$. The activation memory for the GRU, GCB, and prototype modules is estimated in the same manner as for the convolutional layers.

Gradients and Optimizer States

Gradients require the same amount of memory as parameters (float32), while Adam-type optimizers maintain two auxiliary states per parameter. Hence,

$$\text{Gradients} = \#\Theta, \quad \text{Optimizer states} = 2\,\#\Theta.$$

References

1. Ribeiro, M.T.; Singh, S.; Guestrin, C. "Why Should I Trust You?" Explaining the Predictions of Any Classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 1135–1144. https://doi.org/10.1145/2939672.2939778.
2. Miller, T. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.* **2019**, *267*, 1–38. https://doi.org/10.1016/j.artint.2018.07.007.

3. Chen, C.; Li, O.; Tao, D.; Barnett, A.; Rudin, C.; Su, J.K. This Looks Like That: Deep Learning for Interpretable Image Recognition. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; Volume 32. https://doi.org/10.48550/arXiv.1806.10574.
4. Donnelly, J.; Barnett, A.J.; Chen, C. Deformable protopnet: An interpretable image classifier using deformable prototypes. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 19–24 June 2022; pp. 10265–10275. https://doi.org/10.1109/CVPR52688.2022.01002.
5. Singh, G.; Yow, K.C. These do not look like those: An interpretable deep learning model for image recognition. *IEEE Access* **2021**, *9*, 41482–41493. https://doi.org/10.1109/ACCESS.2021.3064838.
6. Barnes, E.A.; Barnes, R.J.; Martin, Z.K.; Rader, J.K. This looks like that there: Interpretable neural networks for image tasks when location matters. *Artif. Intell. Earth Syst.* **2022**, *1*, e220001. https://doi.org/10.1175/AIES-D-22-0001.1.
7. Nauta, M.; Van Bree, R.; Seifert, C. Neural prototype trees for interpretable fine-grained image recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2021; pp. 14933–14943. https://doi.org/10.1109/CVPR46437.2021.01469.
8. Seo, S.; Kim, S.; Park, C. Interpretable Prototype-based Graph Information Bottleneck. In Proceedings of the Advances in Neural Information Processing Systems, New Orleans, LA, USA, 10–16 December 2023; Volume 36, pp. 76737–76748. https://doi.org/10.48550/arXiv.2310.19906.
9. Gao, C.; Zhang, T.; Jiang, X.; Huang, W.; Chen, Y.; Li, J. ProtoPLSTM: An Interpretable Deep Learning Approach for Wearable Fine-Grained Fall Detection. In Proceedings of the 2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Meta), Haikou, China, 15–18 December 2022; pp. 516–524. https://doi.org/10.1109/SmartWorld-UIC-ATC-ScalCom-DigitalTwin-PriComp-Metaverse56740.2022.00091.
10. Liu, M.; Ren, S.; Ma, S.; Jiao, J.; Chen, Y.; Wang, Z.; Song, W. Gated transformer networks for multivariate time series classification. *arXiv* **2021**, arXiv:2103.14438. https://doi.org/10.48550/arXiv.2103.14438.
11. Wu, H.; Hu, T.; Liu, Y.; Zhou, H.; Wang, J.; Long, M. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. In Proceedings of the Eleventh International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023. https://doi.org/10.48550/arXiv.2210.02186.
12. Zerveas, G.; Jayaraman, S.; Patel, D.; Bhamidipaty, A.; Eickhoff, C. A transformer-based framework for multivariate time series representation learning. In Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining, Virtual Event, Singapore, 14–17 August 2021; pp. 2114–2124. https://doi.org/10.1145/3447548.3467401.
13. Wang, Z.; Yan, W.; Oates, T. Time series classification from scratch with deep neural networks: A strong baseline. In Proceedings of the 2017 International joint conference on neural networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; IEEE: New York, NY, USA, 2017; pp. 1578–1585. https://doi.org/10.1109/IJCNN.2017.7966039.
14. Ding, H.; Trajcevski, G.; Scheuermann, P.; Wang, X.; Keogh, E. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.* **2008**, *1*, 1542–1552. https://doi.org/10.14778/1454159.1454226.
15. Abanda, A.; Mori, U.; Lozano, J.A. A review on distance based time series classification. *Data Min. Knowl. Discov.* **2019**, *33*, 378–412. https://doi.org/10.1007/s10618-018-0596-4.
16. Xing, Z.; Pei, J.; Philip, S.Y. Early Prediction on Time Series: A Nearest Neighbor Approach. In Proceedings of the Early Prediction on Time Series: A Nearest Neighbor Approach, Pasadena, CA, USA, 11–17 July 2009; pp. 1297–1302.
17. Górecki, T.; Łuczak, M.; Piasecki, P. An exhaustive comparison of distance measures in the classification of time series with 1NN method. *J. Comput. Sci.* **2024**, *76*, 102235. https://doi.org/10.1016/j.jocs.2024.102235.
18. Ye, L.; Keogh, E. Time series shapelets: A new primitive for data mining. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June–1 July 2009; pp. 947–956. https://doi.org/10.1145/1557019.1557122.
19. Karlsson, I.; Papapetrou, P.; Boström, H. Generalized random shapelet forests. *Data Min. Knowl. Discov.* **2016**, *30*, 1053–1085. https://doi.org/10.1007/s10618-016-0473-y.
20. Yang, Y.; Deng, Q.; Shen, F.; Zhao, J.; Luo, C. A shapelet learning method for time series classification. In Proceedings of the 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI), San Jose, CA, USA, 6–8 November 2016; pp. 423–430. https://doi.org/10.1109/ICTAI.2016.0071.
21. Hills, J.; Lines, J.; Baranauskas, E.; Mapp, J.; Bagnall, A. Classification of time series by shapelet transformation. *Data Min. Knowl. Discov.* **2014**, *28*, 851–881. https://doi.org/10.1007/s10618-013-0322-1.
22. Zhao, B.; Lu, H.; Chen, S.; Liu, J.; Wu, D. Convolutional neural networks for time series classification. *J. Syst. Eng. Electron.* **2017**, *28*, 162–169. https://doi.org/10.21629/JSEE.2017.01.18.
23. He, J.; Wang, W.; Jin, X.; Li, H.; Liu, J.; Chen, B. Multiscale Cross-Attention CNN-Transformer Two-Branch Fusion Network for Detecting Railway U-Shaped Bolts & Nuts Defects. *IEEE/ASME Trans. Mechatronics* **2025**, 1–12. *Early Access*. https://doi.org/10.1109/TMECH.2025.3587940.

24. He, J.; Lv, F.; Liu, J.; Wu, M.; Chen, B.; Wang, S. C2T-HR3D: Cross-Fusion of CNN and Transformer for High-Speed Railway Dropper Defect Detection. *IEEE Trans. Instrum. Meas.* **2025**, *74*, 1–16. https://doi.org/10.1109/TIM.2025.3540132.
25. He, J.; Duan, R.; Dong, M.; Kao, Y.; Guo, G.; Liu, J. CNN-Transformer Bridge Mode for Detecting Arcing Horn Defects in Railway Sectional Insulator. *IEEE Trans. Instrum. Meas.* **2024**, *73*, 1–16. https://doi.org/10.1109/TIM.2024.3373084.
26. Hüsken, M.; Stagge, P. Recurrent neural networks for time series classification. *Neurocomputing* **2003**, *50*, 223–235. https://doi.org/10.1016/S0925-2312(01)00706-8.
27. Tang, Y.; Xu, J.; Matsumoto, K.; Ono, C. Sequence-to-sequence model with attention for time series classification. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), Barcelona, Spain, 12–15 December 2016; pp. 503–510. https://doi.org/10.1109/ICDMW.2016.0078.
28. Narayanan, A.; Bergen, K. Prototype-Based Methods in Explainable AI and Emerging Opportunities in the Geosciences. In Proceedings of the ICML 2024 AI for Science Workshop, Vienna, Austria, 21–27 July 2024. https://doi.org/10.48550/arXiv.2410.19856.
29. Li, O.; Liu, H.; Chen, C.; Rudin, C. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018. https://doi.org/10.1609/AAAI.V32I1.11771.
30. Senin, P.; Malinchik, S. Sax-vsm: Interpretable time series classification using sax and vector space model. In Proceedings of the 2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, 12–15 December 2016; pp. 1175–1180. https://doi.org/10.1109/ICDM.2013.52.
31. Xing, Z.; Pei, J.; Yu, P.S.; Wang, K. Extracting Interpretable Features for Early Classification on Time Series. In Proceedings of the 2011 SIAM International Conference on Data Mining (SDM), Mesa, AZ, USA, 29–30 April 2011; pp. 247–258. https://doi.org/10.1137/1.9781611972818.22.
32. Liang, Z.; Wang, H. Efficient class-specific shapelets learning for interpretable time series classification. *Inf. Sci.* **2021**, *570*, 428–450. https://doi.org/10.1016/j.ins.2021.03.063.
33. Lee, Z.; Lindgren, T.; Papapetrou, P. Z-time: Efficient and effective interpretable multivariate time series classification. *Data Min. Knowl. Discov.* **2024**, *38*, 206–236. https://doi.org/10.1007/s10618-023-00969-x.
34. Middlehurst, M.; Ismail-Fawaz, A.; Guillaume, A.; Holder, C.; Guijo-Rubio, D.; Bulatova, G.; Tsaprounis, L.; Mentel, L.; Walter, M.; Schäfer, P.; et al. aeon: A Python Toolkit for Learning from Time Series. *J. Mach. Learn. Res.* **2024**, *25*, 1–10. https://doi.org/10.48550/arXiv.2406.14231.
35. Dau, H.A.; Keogh, E.; Kamgar, K.; Yeh, C.C.M.; Zhu, Y.; Gharghabi, S.; Ratanamahatana, C.A.; Yanping.; Hu, B.; Begum, N.; et al. The UCR Time Series Classification Archive. 2018. Available online: https://www.cs.ucr.edu/~eamonn/time_series_data_2018/ (accessed on 2 November 2025).
36. Bagnall, A.; Dau, H.A.; Lines, J.; Flynn, M.; Large, J.; Bostrom, A.; Southam, P.; Keogh, E. The UEA multivariate time series classification archive, 2018. *arXiv* **2018**, arXiv:1811.00075. https://doi.org/10.48550/arXiv.1811.00075.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.