An Empirical Evaluation of Foundation Models for Multivariate Time Series Classification

Pinar Sungu Isiacik, Thach Le Nguyen, Timilehin Aderinola, and Georgiana Ifrim

University College Dublin, Dublin, Ireland
pinar.sunguisiacik@ucdconnect.ie
{thach.lenguyen,timilehin.aderinola,georgiana.ifrim}@ucd.ie

Abstract. Foundation models have recently emerged as a promising approach for time series analysis, adapting transformer architectures originally designed for natural language processing to handle continuous temporal data. While these models demonstrate strong performance across various time series tasks, their handling of multivariate time series, particularly inter-channel dependencies, remains underexplored. In this paper, we present a comprehensive analysis of current foundation models for time series, including tokenization-based, patch-based, and shape-based approaches, focusing on their mechanisms and data representations for capturing relationships between channels. Our analysis shows that even though these models have advanced architectures, they mostly process channels independently, which may prevent them from fully capturing cross-channel patterns. We examine this limitation across different model families and discuss its implications for multivariate time series analysis. Our empirical evaluation shows that foundation models perform well on simpler tasks but exhibit diminished effectiveness as channel dependencies increase, with specialized time series methods consistently outperforming them on complex datasets. These findings highlight the critical need for channel-aware architectures and more effective strategies for modeling inter-channel relationships in foundation models.

Keywords: Foundation Models · Multivariate Time Series Classification · Inter-Channel Dependencies · Evaluation

1 Introduction

The success of foundation models in natural language processing has inspired their adaptation to other domains, including time series analysis. These models, pre-trained on large-scale datasets and fine-tuned for specific tasks, have shown promising results in time series forecasting, classification, and anomaly detection [20]. Recent approaches such as Chronos [1], MOMENT [17], One Fits All [29], aLLM4TS [2], Mantis [12] and VQShape [27] demonstrate various strategies for adapting transformer architectures to handle continuous temporal data [20], ranging from tokenization schemes to patch-based processing and shape-based representations. However, as these models are increasingly applied to multivariate time series (MTS) problems, a critical question emerges: how effectively do they capture dependencies between different channels or variables? This question is particularly relevant in domains such as human activity analysis and healthcare, where relationships between multiple measurements are crucial, e.g., tracking multiple body parts during an exercise, or in financial markets, where correlations between different assets drive system behavior. Traditional time series analysis methods, including both statistical approaches and neural networks, have explicitly addressed channel dependencies through various mechanisms. Some Convolutional Neural Networks (CNNs) architectures use 2D convolutions to capture cross-channel patterns, while methods like ROCKET [5] combine channel information during convolution through weighted summation across channels and further leverage these dependencies in the classification stage. In contrast, our analysis shows that current foundation models for time series predominantly process channels independently, treating them as separate sequences in their batch dimension. This design choice, while computationally efficient and effective for capturing temporal patterns within individual channels, may limit these models' ability to learn and leverage inter-channel dependencies. This paper makes the following contributions:

- We provide a systematic analysis of how current foundation models handle multivariate time series, categorizing their approaches into tokenization-based, patch-based, and shape-based methods.
- We examine the specific mechanisms each model employs for processing multiple channels and capturing channel dependencies.
- We identify a common limitation across these models in their treatment of inter-channel relationships, suggesting an important direction for future research in time series foundation models. We showcase this limitation through extensive experiments on synthetic and real multivariate time series classification (MTSC) datasets.
- We discuss potential solutions for addressing this limitation. We make all our data and code publicly available[1].

2 Related Work

Time series classification has seen significant methodological advances, from traditional feature-based approaches to advanced architectures, including foundation models. This section reviews key methods evaluated in our study, emphasizing their architectural design and capabilities in managing channel dependencies in MTS data.

2.1 Traditional Machine Learning Methods

Traditional machine learning (ML) approaches for time series classification (TSC) can be categorized into several families. Tree-based methods include Random

[1] https://github.com/mlgig/FM4MTSC

Forest [3] and Gradient Boosting [15], which build ensembles of decision trees to create robust classifiers. Linear classifiers, including Logistic Regression [4] and Ridge Classifier [18] separate classes using hyperplanes in the feature space. K-Nearest Neighbors (KNN) [13] makes predictions based on proximity in the feature space without constructing an explicit model. For MTSC, these methods typically treat the data as tabular by concatenating all channels into a single feature vector, flattening the temporal and channel dimensions. While this approach may not explicitly capture temporal dependencies or channel interactions, it allows these traditional methods to be directly applied to time series data, providing baseline performance for comparison with specialized temporal models [9].

2.2 Time Series Methods

ROCKET [5] is a convolution-based algorithm which transforms time series using a large number of random convolutional kernels, where each kernel has random length, weights, bias, dilation, and padding. Its implementation for MTS combines channel information during convolution through weighted summation across channels, thus capturing channel dependencies. MiniRocket [6] follows the same principle, but is more efficient, using 84 deterministic kernels.

HYDRA [7] is a hybrid method combining ideas of dictionary and convolutional approaches. Like ROCKET, it first uses convolutional kernels for feature extraction, and then implements concepts of dictionary methods by organizing kernels into groups and counting the best-matching patterns at each timepoint.

QUANT [8] is an interval-based algorithm which serves as a strong baseline in recent TSC benchmarks [23]. It computes quantiles over fixed dyadic intervals on the input time series and its transformations (first and second difference, Fourier transform). When applied to MTS, it extracts features independently from each channel and concatenates them into a single feature vector which is fed to an Extra Trees Classifier [16].

Among feature-based methods, Catch22 [21] is a popular approach that provides a compact set of 22 statistical features designed to capture diverse time series characteristics. These features are extracted independently from each channel, lacking explicit mechanisms to capture interactions between channels.

2.3 Deep Learning Approaches

CNNs have been widely utilized in time series classification due to their ability to automatically learn hierarchical features. In multivariate settings, CNNs process multiple channels using 2D convolutions [19], where early layers operate on each channel independently, extracting local features, while deeper layers combine information across channels. This architecture allows CNNs to capture some inter-channel relationships, but the initial independent processing may limit their effectiveness in scenarios with strong channel dependencies, where interactions between channels are crucial for accurate classification.

InceptionTime [31] introduces an ensemble of five inception-based classifiers that apply convolutions with different kernel sizes in parallel to capture multi-scale temporal patterns. While achieving state-of-the-art performance on many benchmarks, it processes multivariate channels through parallel pathways without explicitly modeling inter-channel dependencies during feature extraction.

LITEMVTime [30] extends lightweight ensemble approaches to MTS, balancing accuracy and computational efficiency for resource-constrained applications. It has channel-aware design elements that better utilize inter-channel relationships while maintaining lightweight computational characteristics.

Although TimesNet [28] is described as a foundation model in its original paper, we categorize it as a deep learning approach due to its architectural design and training paradigm. Unlike foundation models, which typically rely on large-scale pre-training and transformer-based architectures to learn generalizable representations across diverse tasks, TimesNet employs a CNN-based architecture that transforms 1D time series into 2D representations through Fast Fourier Transform (FFT) based periodicity analysis. Its TimesBlocks reshape the time series based on identified periodicities and process these 2D tensors with multi-scale kernels to capture complex temporal patterns. Similar to other deep learning approaches, TimesNet processes channels separately in its initial stages, which may limit its ability to fully leverage strong inter-channel dependencies present in MTS data.

2.4 Transformer-based Methods

ConvTran [14] combines convolutional layers and transformer blocks to capture both local and global dependencies in time series data. Initially, convolutional layers extract local features, efficiently modeling short-term dependencies within each channel. Subsequently, transformer blocks leverage self-attention mechanisms to capture long-range dependencies and complex inter-channel interactions. The integration of these components allows ConvTran to overcome the limitations of using either approach alone. This can make ConvTran effective in MTS classification, particularly in datasets with strong channel dependencies. However, for datasets with weak channel dependencies, the model's complexity might not yield substantial benefits compared to simpler architectures.

TSLANet [11] introduces a novel approach to time series representation learning by integrating the Adaptive Spectral Block (ASB) and the Interactive Convolution Block (ICB). The ASB leverages the Discrete Fourier Transform to transform time series data into the frequency domain. This process involves computing the power spectrum and applying a trainable threshold to filter out noise, followed by the Inverse FFT to reconstruct time-domain features. The ICB further refines these features using a dual-layer convolutional structure with varying kernel sizes to capture both local and long-range dependencies. This block encourages interactions between features extracted at different scales, enhancing the model's ability to capture complex temporal relationships. TSLANet may be effective in scenarios with strong channel dependencies due to its ability to model complex inter-channel interactions through its spectral and convolutional components. However, in datasets with weak channel dependencies, its architecture may not provide significant advantages over simpler models.

2.5 Foundation Models

Recent advances in foundation models have introduced various approaches to time series analysis, which we categorize here by their data representation strategies.

Tokenization-based methods convert continuous time series values into discrete tokens, similar to vocabulary-based approaches in natural language processing. Chronos [1] uses this approach by transforming continuous-valued data through scaling and quantization schemes that map values to discrete tokens while preserving probabilistic characteristics, enabling language models to process time series for downstream tasks. Despite its potential versatility, Chronos was empirically validated only on univariate time series forecasting tasks, leaving its effectiveness for MTS scenarios and other applications as an open question.

Patch-based methods segment time series into fixed-length subsequences (i.e., patches) that serve as input tokens to transformer architectures, drawing inspiration from vision transformers where image patches are treated as sequence elements. OneFitsAll [29] implements a transfer learning framework that processes time series in patches while maintaining frozen pre-trained transformer blocks, while Adapting LLMs (Large Language Models) for Time Series (aLLM4TS) [2] extends this approach through its framework for handling arbitrary temporal windows. MOMENT [17] advances patch-based processing by introducing "The Time Series Pile" for large-scale pre-training, supporting multiple tasks through various deployment modes. Mantis [12] adapts the Vision Transformer [10] architecture to time series data by generating tokens from patched time series and their differentials, employing contrastive learning during pre-training and introducing channel-level adapters to handle multivariate inputs efficiently.

Shape-based methods represent time series through abstract shape features and attributes. VQShape [27] uses vector quantization to create interpretable, reusable shape-level representations through learned codebooks that generalize across domains.

A critical limitation across all these approaches is their handling of MTS: most current foundation models process channels independently, potentially missing inter-channel dependencies crucial for multivariate analysis.

3 Systematic Analysis of Foundation Models for MTSC

This section examines how current foundation models handle MTS, focusing on their data representations and mechanisms for processing multiple channels and capturing inter-channel dependencies. We denote a MTS as $X \in \mathbb{R}^{L \times C}$, where L denotes the sequence length and C represents the number of channels.

Chronos [1] approaches time series analysis by adapting language modeling techniques through a tokenization strategy. The model processes channels independently at multiple stages. First, in the scaling operation, each channel c is normalized separately using its own scale factor s_c: $\tilde{x}_c = \frac{x_c - m}{s_c}$, $s_c = \frac{1}{L}\sum_{i=1}^{L} |x_c[i]|$, where $\tilde{x}_c$ represents the scaled values for channel c, x_c is the original time series for channel c, m is the scaling factor, s_c is the channel-specific scale factor computed as the mean absolute value, L denotes the sequence length, and $x_c[i]$ is the value at time step i in channel c.

This channel-wise processing continues in the tokenization phase, where the quantization function $q(\tilde{x})$ maps the scaled values of each channel to tokens independently:

$$q(\tilde{x}) = \begin{cases} 1 & \text{if } -\infty \leq \tilde{x} < b_1 \\ 2 & \text{if } b_1 \leq \tilde{x} < b_2 \\ \vdots & \\ B & \text{if } b_{B-1} \leq \tilde{x} < \infty \end{cases} \tag{1}$$

where $q(\cdot)$ is the quantization function, B is the total number of tokens in the vocabulary, and $b_1, b_2, \ldots, b_{B-1}$ are the quantization boundaries that partition the scaled value space into discrete intervals. The dequantization function $d : \{1, 2, 3, ...B\} \to \mathbb{R}$ is defined as $d(j) = c_j$, which maps each token back to a representative real value. While the transformer's self-attention mechanism processes these tokenized sequences:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

each channel must either be processed independently or flattened into a single sequence. Unlike methods such as ROCKET [5], where cross channel processing is followed by a linear classifier that can learn channel interactions through its weights, Chronos lacks an explicit mechanism for capturing inter-channel dependencies. The model processes each channel in isolation throughout its pipeline, from initial scaling through tokenization to the final prediction phase where the categorical distribution $p(z_{t+1}|z_{1:t})$ is computed separately for each channel, with $z_{1:t}$ representing the tokenized time series.

An in-depth analysis of the Chronos implementation code confirms this critical limitation. The tokenizer class processes inputs strictly as 2D tensors of shape (batch size, time length), with no provision for a channel dimension. The normalization process calculates scaling factors along the time dimension only, confirming that normalization happens independently for each channel. Moreover, the model's input validation explicitly requires inputs to be at most 2-dimensional, forcing MTS to be processed as separate univariate sequences. For MTS applications, this architecture imposes significant constraints.

In our implementation of zero-shot feature extraction for classification tasks, this limitation requires a preprocessing approach where MTS data is converted to univariate sequences by concatenating all channels before feeding it to the model. While this enables Chronos to process the data, it fundamentally changes the time relationships between channels.

OneFitsAll [29] adapts pre-trained language models for time series analysis through a transfer learning framework. Given a MTS $X \in \mathbb{R}^{L \times C}$, the model first applies reverse instance normalization to each channel independently: $\tilde{X}_c = \text{InstanceNorm}(X_c) + \mu_c + \sigma_c$ where $\tilde{X}_c$ is the normalized time series for channel c, X_c is the original time series for channel c, μ_c and ω_c are the channel-wise mean and standard deviation used for reverse instance normalization, and $\text{InstanceNorm}(\cdot)$ applies instance-level normalization to the input, followed by patch creation. These patches are processed through a transformer architecture where self-attention blocks and feedforward layers are frozen, while only the positional embeddings and layer normalization are fine-tuned. The reverse instance normalization is completed by adding the channel-wise mean μ_c and standard deviation σ_c to the outputs of the transformer blocks: $\hat{X}_c = \text{TransformerOutput}_c + \mu_c + \sigma_c$, rather than directly after the initial normalization step. Similar to other foundation models, OneFitsAll processes each channel independently in the batch dimension.

aLLM4TS [2] introduces a two-stage framework for time series analysis. Given a MTS $X \in \mathbb{R}^{L \times C}$, the model first flattens it into M univariate sequences. For each channel i, the sequence is divided into patches: $p^{(i)}_{tp:tp+L_p-1} = \{p^{(i)}_{tp}, ..., p^{(i)}_{tp+L_p-1}\} \in \mathbb{R}^{L_p \times P}$, where $p^{(i)}_{tp:tp+L_p-1}$ represents the patch sequence for channel i, $t_p = \lfloor (t-P)/S \rfloor + 1$ is the starting patch index, $L_p = \lfloor (L-P)/S \rfloor + 1$ is the number of patches, P is the patch length, S is the sliding stride, and L is the total sequence length. In the causal next-patch pre-training stage, the model predicts the next patch for each channel independently: $\hat{p}^{(i)}_{tp+1:tp+L_p} = \{\hat{p}^{(i)}_{tp+1}, ..., \hat{p}^{(i)}_{tp+L_p}\} \in \mathbb{R}^{L_p \times D}$.

The loss function is computed independently for each channel and averaged:

$$\mathcal{L}_p = \mathbb{E}_p[\frac{1}{M} \sum_{i=1}^{M} \|\hat{p}^{(i)}_{tp+1:tp+L_p} - p^{(i)}_{tp+1:tp+L_p}\|_2^2] \tag{2}$$

where $\mathcal{L}_p$ is the prediction loss, M is the number of channels, $\hat{p}^{(i)}_{tp+1:tp+L_p}$ is the predicted patch sequence for channel i, $p^{(i)}_{tp+1:tp+L_p}$ is the ground truth patch sequence, and $\| \cdot \|_2^2$ denotes the squared L2 norm.

The model explicitly adopts a "channel-independence setting" where each sequence is processed independently through the causal LLM backbone. This design choice is confirmed in the implementation, where multivariate inputs undergo a series of transformations: first transposing the time and channel dimensions, then applying padding before creating patches through unfolding operations, and finally rearranging the data to concatenate channels and patch values into a single dimension. While this concatenation operation might suggest some cross-channel modeling, it actually follows the channel-independence principle. The model simply treats the concatenated patches as longer univariate sequences, without any mechanism to learn relationships between different channels.

MOMENT [17] follows a patch-based strategy that processes channels independently. For a MTS $X \in \mathbb{R}^{L \times C}$, the model first applies reversible instance normalization to each channel separately, then segments each normalized channel into non-overlapping patches: $X_c = [p_1^c, ..., p_N^c], \quad p_i^c \in \mathbb{R}^P$ where P is the patch length and N is the number of patches. Each patch is then embedded through a linear projection or replaced with a special mask token during pre-training:

$$e_i^c = \begin{cases} \text{Linear}(p_i^c) & \text{if } M_i = 1 \\ [\text{MASK}] & \text{if } M_i = 0 \end{cases} \tag{3}$$

where e_i^c is the embedding for patch i of channel c, p_i^c is the patch, M_i is a binary mask indicator (1 for non-masked, 0 for masked), Linear$(\cdot)$ is a linear projection layer, and [MASK] is a special mask token.

A key aspect of MOMENT's design is how it handles multiple variables (channels) in time series data. Although the model uses a transformer, which could potentially capture relationships between different variables, MOMENT takes a different approach. It processes each variable separately by using a simple reshaping: Input to Transformer $=$ reshape$(E, (B \times C, N, D))$ where E is the embedding matrix, B is the batch size, C is the number of channels, N is the number of time segments (patches), and D is the embedding dimension.

This reshaping treats each variable as if it were a separate time series. As a result, the transformer processes all variables in parallel but keeps them completely separate - like processing multiple independent univariate time series rather than a single MTS.

The model applies both relative and absolute positional encodings to the patch embeddings before processing them through the transformer encoder: $Z^c = \text{Transformer}([e_1^c, e_2^c, ..., e_N^c])$ where Z is the final representation, Z^c is the representation for channel c, and $\|$ denotes concatenation. Only at the task-specific output layer does MOMENT attempt to handle cross-channel information, primarily through two reduction strategies: mean reduction, which averages representations across channels ($Z = \frac{1}{C}\sum_{c=1}^{C} Z^c$), or concatenation, which preserves channel-specific information ($Z = [Z^1||Z^2||...||Z^C]$) but increases parameter count in the final layer.

This architectural decision has significant implications for MTS modeling. Processing channels independently reduces the quadratic complexity of self-attention from $O((L \cdot C)^2)$ to $O(C \cdot L^2)$, enabling the model to scale to high-dimensional MTS. However, it also means the model cannot directly capture dependencies between channels during feature extraction, which may limit its ability to model complex inter-channel relationships.

Mantis [12] is a time series classification foundation model that functions as an encoder $F : \mathbb{R}^L \to \mathbb{R}^q$, projecting any univariate time series $x \in \mathbb{R}^L$ with fixed sequence length L to a discriminative hidden space $\mathbb{R}^q$. During pre-training, an unlabeled dataset $\mathcal{D}_0$ containing multiple time series is used to learn rich embeddings that generalize across tasks. For fine-tuning, given a dataset $\mathcal{D}$ of time series $\{X_1, X_2, ..., X_n\}$ and corresponding labels Y, either F extracts embeddings $Z = \{F(X_i) \mid X_i \in \mathcal{D}\}$ for a classifier $h : \mathbb{R}^q \to \{1, \dots, K\}$, or a classification head $h : \mathbb{R}^q \to \mathbb{R}^K$ is appended to fine-tune $h \circ F$.

For MTS $X \in \mathbb{R}^{L\times C}$, Mantis processes each channel independently with the embedding defined as $z = \text{concat}[(F(X_c))_{1\leq c\leq C}]$, where z is the final multivariate embedding, $F(\cdot)$ is the univariate encoder function, X_c represents the c-th channel, C is the total number of channels, and concat$[\cdot]$ denotes concatenation along the feature dimension. An adapter $a : \mathbb{R}^{L\times C} \to \mathbb{R}^{L\times C_{\text{new}}}$ compresses the original channels. The model's confidence in classifying X is $\text{conf}(X) = \max[\sigma(h\circ F(X))]$, where conf$(X)$ is the classification confidence for input X, $\sigma(\cdot)$ is the softmax function, h is the classification head, $F(X)$ is the encoded representation, and max$[\cdot]$ returns the maximum probability across all classes.

Mantis adapts the Vision Transformer (ViT) architecture for time series data. The pre-processing involves setting the input sequence length to 512 and applying instance-level standard scaling. The token generator unit performs instance-level normalization, splits the time series into 32 patches using convolution and mean pooling, generates patches for the time series differential to reduce trend influence, and preserves information about original measurements by encoding statistics of raw patches. These features are concatenated and passed through a linear projector with layer normalization to generate 32 tokens of dimension 256. The ViT unit appends a learnable class token, applies sinusoidal positional encoding, and processes the tokens through 6 transformer layers with 8-head attention. The class token's final representation serves as the output. During pre-training, a projection layer is added for similarity calculations, while during fine-tuning, a classification head maps embeddings to class logits. Mantis is pre-trained using contrastive learning with pairwise cosine similarities computed as $s_i(\phi,\psi) = [s_{\cos}(g \circ F \circ \phi(x_i), g \circ F \circ \psi(x_j))]_{j=1}^{b}$, where g is a projector. The model minimizes a contrastive loss defined as $\sum_{i=1}^{b} l_{\text{ce}}\left(\frac{s_i(\phi,\psi)}{T}, i\right)$ with temperature $T = 0.1$, using RandomCropResize as the primary augmentation method.

For MTS, Mantis employs channel-level adapters that transform the original d channels into d_{new} channels, including Principal Component Analysis (PCA), Truncated Singular Value Decomposition (SVD), Random Projection, Variance-Based Channel Selection, and a Differentiable Linear Combiner. Despite these adapters, Mantis follows the same pattern observed in other foundation models, processing each channel independently and lacking explicit mechanisms for modeling inter-channel dependencies.

VQShape [27] introduces a shape-based representation approach for time series analysis. For a MTS $X \in \mathbb{R}^{L\times C}$, VQShape processes each channel $x_i^m \in \mathbb{R}^T$ independently through a shape-level representation framework. Each subsequence is represented by an attribute tuple: $\tau_k = (z_k, \mu_k, \sigma_k, t_k, l_k)$ where $z_k \in \mathbb{R}^{d_{code}}$ is the shape code, μ_k is the offset, σ_k is the scale, t_k is the relative starting position, and l_k is the relative length. The model processes each channel independently through its encoding pipeline: $\{h_k \in \mathbb{R}^{d_{embed}} \mid k = 1, ..., K\} = E(x^m)$ where E is the time series encoder that transforms patches into latent embeddings. The attribute decoder processes these embeddings channel-wise: $\hat{\tau}_k = (\hat{z}_k, \mu_k, \sigma_k, t_k, l_k) = A_{dec}(h_k)$

While VQShape introduces an innovative approach to time series representation through shape abstraction, it follows the common pattern of processing each channel independently. The model architecture, from the initial encoding through shape representation to the final reconstruction, maintains separate processing streams for each channel. Despite its novel shape-based tokenization strategy, the model lacks mechanisms for capturing dependencies between channels in MTS data. Looking at the implementation code, the model flattens all channels and time steps together into a single long sequence. This removes the clear boundaries between channels. Since the channel structure is lost in this flattening, the model cannot easily identify or learn relationships between different variables in the MTS.

The model's loss function $\mathcal{L}_{\text{pretrain}} = \lambda_x \mathcal{L}_x + \lambda_s \mathcal{L}_s + \lambda_{vq} \mathcal{L}_{vq} + \lambda_{div} \mathcal{L}_{div}$ where $\mathcal{L}_x$ is the time series reconstruction loss, $\mathcal{L}_s$ is the shape reconstruction loss, $\mathcal{L}_{vq}$ is the vector quantization loss, $\mathcal{L}_{\text{div}}$ is the shape disentanglement loss, and $\rho_x, \rho_s, \rho_{vq}, \rho_{\text{div}}$ are the respective loss weights. $\mathcal{L}_{\text{pretrain}}$ optimizes for time series reconstruction, vector quantization, shape reconstruction, and shape disentanglement, but none of these components explicitly addresses cross-channel relationships. This framework yields highly interpretable representations within each channel but sacrifices the ability to model how these representations interact across the multivariate structure of the data.

Summary. Across these various types of foundation models, we observe a consistent pattern in handling MTS data: channels are processed independently regardless of the underlying representation strategy. Even approaches that introduce specialized components for multivariate data, such as Mantis with its channel-level adapters, still maintain separate processing streams for each channel. This design choice, while computationally efficient and effective for capturing temporal patterns within individual channels, means that none of these models incorporate explicit mechanisms for learning inter-channel dependencies. The computational advantage is significant, reducing self-attention complexity from $O((L \cdot C)^2)$ to $O(C \cdot L^2)$, but it comes at the cost of potentially missing critical cross-channel interactions that characterize complex MTS data. A structured comparison of these approaches is given in the Appendix (Table 9).

4 Datasets

We study four multivariate time series classification datasets with varying degrees of channel dependency. We emphasize that in this paper **channel dependency** refers to the necessity of using multiple channels (at least two) for successful classification, rather than the statistical correlation between channels. This distinction is crucial: a dataset exhibits strong channel dependency when accurate classification cannot be achieved using a single channel alone, even if the channels themselves are not correlated (e.g., this is the case for the SYNTH dataset). In contrast, we consider a dataset to have weak channel dependency when a single channel contains sufficient discriminative information for accurate classification. Detailed dataset characteristics are provided in Table 7 in the Appendix.

CounterMovementJump (CMJ) [24] contains accelerometer data from countermovement jump exercises with 3 classes and 3 channels (x, y, z acceleration). This dataset exhibits *weak channel dependency*, as domain experts confirm that class distinctions are primarily observable on the y-channel.

Military Press (MP) [26] includes motion capture data of exercise performances in two variants: MP8 (8 key body point coordinates) and MP50 (all 50 coordinates from 25 body parts) with 4 classes. Domain experts confirm *strong channel dependency*, requiring at minimum four channels (left/right elbow and wrist coordinates) for effective classification.

Synthetic (SYNTH) [25] is specifically designed to evaluate *strong channel dependency* scenarios. It contains 8 channels where discriminative features appear in only two randomly selected channels, making classification impossible using any single channel alone, demonstrating strong channel dependency.

5 Experiments

All experiments were run on an Apple M1 Pro with 16GB RAM. Traditional ML and time series methods were run on CPU using default parameters. For deep learning approaches, transformer-based models, and foundation models requiring fine-tuning, the MPS backend was utilized. We used the original implementations provided by each method's paper. The training process used 25% of the training set as a validation set, with early stopping applied using a patience of 10 epochs and a maximum of 100 epochs to determine the optimal training duration and mitigate overfitting. Our experimental evaluation uses four distinct MTS datasets, with three exhibiting strong channel dependency characteristics. As suggested in [9], we establish baseline performance using traditional ML methods including Logistic Regression and Random Forest.

For **traditional ML** approaches, we flattened the MTS data by concatenating all channels into a single feature vector, maintaining temporal order within each channel. All traditional models were implemented using *scikit-learn* with default hyperparameters to provide a fair and reproducible baseline. As shown in Table 1, traditional methods showed strong performance on the simpler CMJ dataset but struggled with channel-dependent datasets, highlighting the limitations of flattened representations in capturing inter-channel relationships. We next compare these baselines against specialized time series approaches, deep learning architectures, transformer-based models, and foundation models.

Time series methods utilize the *aeon* library [22] implementations with default parameters for ROCKET, MiniRocket, QUANT, HYDRA, and Catch22. As shown in Table 2, these methods consistently outperformed traditional approaches, with ROCKET achieving superior accuracy on CMJ and MP50, while HYDRA led on MP8 and QUANT excelled on the SYNTH dataset. Notably, MiniRocket maintained competitive accuracy across all datasets while offering significantly faster computation times, making it the most balanced choice for

Table 1: Accuracy and Runtime Results for Traditional ML Methods

Dataset	**Model**	**Acc**	**TrainTime(s)**	**PredTime(s)**
CMJ	Random Forest	0.933	0.41	0.01
	Gradient Boosting	**0.939**	30.37	0.01
	KNN	0.620	0.01	0.01
	Logistic Regression	0.687	0.16	0.01
	Ridge Classifier	0.536	0.16	0.01
MP8	Random Forest	0.607	2.10	0.08
	Gradient Boosting	0.605	166.72	0.01
	KNN	0.548	0.01	0.01
	Logistic Regression	**0.642**	2.80	0.01
	Ridge Classifier	0.607	0.33	0.01
MP50	Random Forest	0.476	4.73	0.01
	Gradient Boosting	0.555	964.20	0.12
	KNN	0.341	0.01	0.06
	Logistic Regression	**0.622**	36.83	0.02
	Ridge Classifier	0.540	0.85	0.01
SYNTH	Random Forest	0.538	17.84	0.03
	Gradient Boosting	0.518	199.24	0.01
	KNN	**0.544**	0.06	0.40
	Logistic Regression	0.537	9.28	0.02
	Ridge Classifier	0.519	0.90	0.01

practical applications. The results demonstrate the effectiveness of these specialized approaches in handling complex time series data, particularly in capturing channel dependencies that traditional methods struggled with.

Deep learning approaches showed varying effectiveness across datasets with different channel dependency characteristics, as shown in Table 3. InceptionTime demonstrated better performance on channel dependent datasets compared to basic CNN approaches, highlighting the benefits of its multi-scale ensemble architecture. However, CNN (aeon) achieved the best results on both weak dependency and synthetic tasks, while performing poorly on real-world strong dependent datasets. LITEMVTime showed mixed results with generally lower accuracy and unexpectedly high computational overhead on these datasets. The results show that no single deep learning architecture consistently performs well across all channel dependency scenarios.

Transformer-based methods (Table 4) showed promise, particularly ConvTran's strong performance on SYNTH and MP8, but performance varied considerably depending on the data characteristics. TSLANet generally performed worse than ConvTran, especially on the datasets with strong channel dependence. The performance gap expands noticeably on MP50 and SYNTH, suggesting ConvTran's architecture better handles strong channel dependencies, though at the cost of longer training times on larger datasets. Channel dependency emerges as a key factor in determining when transformer-based approaches become necessary. We note that on these datasets, MiniRocket is comparable with ConvTran regarding accuracy, but much faster to train and predict.

For the **foundation models** analyzed, we evaluated both zero-shot and fine-tuning approaches, as shown in Table 5. As a sanity check, we reproduced the

Table 2: Accuracy and Runtime Results of Time Series Methods (aeon)

Dataset	Model	Acc	TrainTime(s)	PredTime(s)
CMJ	ROCKET	**0.950**	17.67	7.37
	MiniRocket	0.944	2.26	0.49
	QUANT	0.933	2.65	0.22
	HYDRA	0.944	6.01	2.59
	Catch22	0.922	19.79	8.27
MP8	ROCKET	0.743	43.67	17.65
	MiniRocket	0.741	3.58	1.09
	QUANT	0.696	26.27	0.77
	HYDRA	**0.748**	8.79	3.17
	Catch22	0.635	29.76	11.93
MP50	ROCKET	**0.793**	50.32	20.60
	MiniRocket	0.787	4.86	1.61
	QUANT	0.740	199.30	4.68
	HYDRA	0.738	9.98	3.53
	Catch22	0.672	747.74	285.73
SYNTH	ROCKET	0.861	793.20	94.78
	MiniRocket	0.882	119.53	4.88
	QUANT	**0.964**	553.77	2.92
	HYDRA	0.912	313.10	16.96
	Catch22	0.906	58.96	4.83

experiments from the original papers using their datasets and confirmed that we achieved the same performance metrics as reported by the authors, validating our implementation before applying these models to our datasets.

5.1 Impact of Classifier Selection on Zero-Shot Performance

Our evaluation demonstrates that foundation models show distinct performance patterns that are critically influenced by both dataset complexity and classifier selection. On simpler datasets with weak channel dependencies (CMJ), zero-shot methods like Chronos perform competitively (0.927 with Random Forest), while performance declines significantly on channel-dependent datasets. Notably, classifier choice becomes increasingly important as channel dependency strengthens: while Random Forest consistently performs best on simpler datasets, Ridge Classifier demonstrates notable effectiveness when paired with aLLM4TS embeddings on complex tasks, achieving better performance on MP8 (0.689) and MP50 (0.472) compared to alternative classifier combinations. This pattern indicates that different foundation models produce embeddings with distinct characteristics: Chronos generates representations that benefit from Random Forest's ensemble approach, while aLLM4TS creates more linearly separable patterns that Ridge Classifier can effectively utilize.

Beyond classifier selection, our evaluation shows significant computational trade-offs, with fine-tuning approaches like MOMENT demonstrating better adaptability to complex data but requiring exponentially higher computational cost compared to zero-shot methods. Despite these optimization strategies, foundation models consistently underperform specialized time series methods on channel-dependent tasks, with even transformer architectures specifically de-

Table 3: Accuracy and Runtime Results of Deep Learning Methods

Dataset	**Model**	**Acc**	**Epochs**	**TrainTime(s)**	**PredTime(s)**
CMJ	CNN	0.922	12	2.07	0.05
	TimesNet	0.866	24	220.66	0.95
	CNN (aeon)	**0.950**	2000	335.61	0.25
	InceptionTime (aeon)	0.905	500	578.49	1.35
	LITEMVTime (aeon)	0.899	500	2127.92	1.08
MP8	CNN	0.810	55	27.28	0.08
	TimesNet	0.351	11	186.74	1.82
	CNN (aeon)	0.659	2000	659.40	0.31
	InceptionTime (aeon)	**0.832**	500	2472.29	1.62
	LITEMVTime (aeon)	0.812	500	2618.88	1.42
MP50	CNN	0.661	56	24.37	0.10
	TimesNet	0.245	32	530.50	1.79
	CNN (aeon)	0.252	2000	1390.24	0.45
	InceptionTime (aeon)	**0.781**	500	4167.91	2.61
	LITEMVTime (aeon)	0.579	500	4924.69	2.59
SYNTH	CNN	0.732	49	92.14	0.26
	TimesNet	0.486	17	3822.72	6.56
	CNN (aeon)	**0.868**	2000	5895.06	0.48
	InceptionTime (aeon)	0.691	500	10194.65	1.68
	LITEMVTime (aeon)	0.576	500	14895.80	2.68

Table 4: Accuracy and Runtime Results of Transformer-based Methods

Dataset	**Model**	**Acc**	**Epochs**	**TrainTime(s)**	**PredTime(s)**
CMJ	ConvTran	0.866	48	114.39	0.65
	TSLANet	**0.894**	34	138.45	1.39
MP8	ConvTran	**0.804**	100	296.04	0.81
	TSLANet	0.727	60	387.87	2.16
MP50	ConvTran	**0.570**	48	273.15	1.58
	TSLANet	0.424	43	287.26	2.08
SYNTH	ConvTran	**0.930**	50	3821.37	6.20
	TSLANet	0.871	20	1898.66	9.74

signed for time series (ConvTran) outperforming general foundation models on complex multivariate data. These findings emphasize that foundation model evaluation requires careful consideration of classifier selection as a critical hyperparameter, particularly for MTS with varying channel dependencies.

Due to the novel proposal of adapters to address channel dependency, **Mantis model variations** (Table 8, Appendix) were evaluated in detail across extraction approaches (with/without adapters) and fine-tuning strategies. Key findings include that adapters offer significant computational efficiency, reducing transform times by 33-95% across datasets, with the most dramatic gains on complex data like MP50. However, accuracy implications vary by dataset complexity. Adapters maintain performance on weak channel dependent data (CMJ) and improve accuracy on moderately complex data (MP8), but reduce accuracy significantly on datasets with strong channel dependence (MP50, SYNTH).

Full fine-tuning significantly outperforms adapter-head fine-tuning on strongly channel dependent datasets, despite incurring a 15–20% increase in training time.

Table 5: Accuracy and Runtime Results of Foundation Models

Dataset	Model	Classifier	Fine-tune	Acc	FeatTime(s)	TrainTime(s)	PredTime(s)
CMJ	Chronos	RandomForest	-	**0.927**	10.79	5.85	0.05
	Chronos	LogisticRegr	-	0.916	10.79	9.32	0.09
	Chronos	RidgeClassif	-	**0.927**	10.79	0.66	0.03
	MOMENT	-	50	0.855	-	81.17	2.17
	One-fits-all	-	28	0.866	-	16.39	0.29
	aLLM4TS	RandomForest	-	0.832	20.29	0.37	0.01
	aLLM4TS	LogisticRegr	-	0.844	20.29	0.06	0.01
	aLLM4TS	RidgeClassif	-	0.872	20.29	0.01	0.01
	VQSHAPE	RandomForest	-	0.922	54.35	0.22	0.01
	VQSHAPE	LogisticRegr	-	0.922	54.35	0.18	0.01
	VQSHAPE	RidgeClassif	-	0.883	54.35	0.02	0.01
MP8	Chronos	RandomForest	-	0.434	32.17	38.68	0.16
	Chronos	LogisticRegr	-	0.467	32.17	30.20	0.61
	Chronos	RidgeClassif	-	0.459	32.17	4.90	0.18
	MOMENT	-	18	0.489	-	103.52	9.56
	One-fits-all	-	33	0.644	-	64.46	0.63
	aLLM4TS	RandomForest	-	0.629	31.38	2.02	0.01
	aLLM4TS	LogisticRegr	-	0.517	31.38	0.09	0.01
	aLLM4TS	RidgeClassif	-	**0.689**	31.38	0.01	0.01
	VQSHAPE	RandomForest	-	0.659	184.68	1.09	0.03
	VQSHAPE	LogisticRegr	-	0.556	184.68	0.52	0.01
	VQSHAPE	RidgeClassif	-	0.422	184.68	0.08	0.01
MP50	Chronos	RandomForest	-	0.287	34.66	39.74	0.16
	Chronos	LogisticRegr	-	0.261	34.66	32.08	0.69
	Chronos	RidgeClassif	-	0.235	34.66	4.59	0.16
	MOMENT	-	58	**0.662**	-	1791.58	119.38
	One-fits-all	-	33	0.266	-	219.96	1.61
	aLLM4TS	RandomForest	-	0.361	32.22	1.80	0.01
	aLLM4TS	LogisticRegr	-	0.390	32.22	0.09	0.01
	aLLM4TS	RidgeClassif	-	0.472	32.22	0.09	0.01
	VQSHAPE	RandomForest	-	0.368	185.38	1.02	0.03
	VQSHAPE	LogisticRegr	-	0.341	185.38	1.22	0.01
	VQSHAPE	RidgeClassif	-	0.281	185.38	0.05	0.01
SYNTH	Chronos	RandomForest	-	0.491	144.34	697.98	2.54
	Chronos	LogisticRegr	-	0.513	144.34	859.18	4.55
	Chronos	RidgeClassif	-	0.491	144.34	178.85	2.70
	MOMENT	-	60	**0.768**	-	5164.22	61.30
	One-fits-all	-	40	0.672	-	414.87	1.19
	aLLM4TS	RandomForest	-	0.497	387.88	16.59	0.02
	aLLM4TS	LogisticRegr	-	0.500	387.88	0.32	0.01
	aLLM4TS	RidgeClassif	-	0.505	387.88	0.07	0.01
	VQSHAPE	RandomForest	-	0.644	855.21	8.32	0.14
	VQSHAPE	LogisticRegr	-	0.671	855.21	0.73	0.01
	VQSHAPE	RidgeClassif	-	0.669	855.21	0.17	0.01

The choice of classifier also plays a crucial role in performance: Random Forest excels on simpler datasets, while Logistic Regression demonstrates effectiveness with adapters on channel-dependent data, reinforcing our findings on foundation models. The most notable disparity is in computational efficiency. Mantis adapter extraction completes in seconds, whereas full fine-tuning requires hours, underscoring the critical trade-off between accuracy and efficiency.

Table 6 summarizes performance across all methods, demonstrating that foundation models struggle to consistently outperform established approaches, particularly on datasets with strong channel dependencies. While foundation models achieve top results, the performance differences are often marginal, with time series and deep learning methods remaining highly competitive across all evaluation scenarios.

Table 6: Accuracy Comparison Across All Methods and Datasets

Category	**Method**	**CMJ**	**MP8**	**MP50**	**SYNTH**
Traditional ML	Random Forest	0.933	0.607	0.476	0.538
	Gradient Boosting	0.939	0.605	0.555	0.518
	KNN	0.620	0.548	0.341	0.544
	Logistic Regression	0.687	0.642	0.622	0.537
	Ridge Classifier	0.536	0.607	0.540	0.519
Time Series	ROCKET	0.950	0.743	**0.793**	0.861
	MiniRocket	0.944	0.741	0.787	0.882
	QUANT	0.933	0.696	0.740	**0.964**
	HYDRA	0.944	0.748	0.738	0.912
	Catch22	0.922	0.635	0.672	0.906
Deep Learning	CNN	0.922	0.810	0.661	0.732
	CNN (aeon)	0.950	0.659	0.252	0.868
	TimesNet	0.866	0.351	0.245	0.486
	InceptionTime	0.905	**0.832**	0.781	0.691
	LITEMVTime	0.899	0.812	0.579	0.576
Transformers	ConvTran	0.866	0.804	0.570	0.930
	TSLANet	0.894	0.727	0.424	0.871
Foundation Models	Chronos (best)	0.927	0.467	0.287	0.513
	MOMENT	0.855	0.489	0.662	0.768
	OneFitsAll	0.866	0.644	0.266	0.672
	aLLM4TS (best)	0.872	0.689	0.472	0.505
	VQShape (best)	0.922	0.659	0.368	0.671
	Mantis Full FT	**0.955**	0.697	0.773	0.929
Dataset Channel Dependency		**Weak**	**Strong**	**Strong**	**Strong**
BEST MODEL		**Mantis**	**InceptionTime**	**ROCKET**	**QUANT**

Potential Solutions to Address Strong Channel Dependence. Based on these findings, we identify several promising directions for improving foundation model performance on channel-dependent MTSC data: (1) pre-training objectives that explicitly model inter-channel relationships; (2) attention mechanisms designed specifically for channel dependencies; (3) hybrid architectures combining foundation model capabilities with channel-aware components like those in ConvTran; (4) more efficient fine-tuning strategies that mitigate the prohibitive computational costs observed in our experiments; (5) data representation approaches that extend Mantis' adaptive channel compression techniques to better preserve cross-channel information, e.g., by redesigning adapters to better capture dependencies between channels. Our results emphasize that model selection should be guided by both dataset characteristics and computational constraints, with specialized approaches like MiniRocket currently still providing more reliable and efficient solutions than foundation models for MTS classification.

6 Conclusion

Our analysis of foundation models for time series demonstrates a consistent pattern in their handling of multivariate data: channels are predominantly processed independently across all three categories of tokenization-based, patch-based, and shape-based approaches. While this design choice offers computational efficiency and has proven effective for many tasks, it fundamentally limits these models' ability to capture complex relationships between channels. This limitation stands in contrast to methods like ROCKET and MiniRocket, which combine channel information during convolution through weighted summation across channels and further leverage channel interactions through the classification stage. Future development of foundation models for time series may benefit from explicitly addressing channel dependencies, either through architectural modifications or novel pre-training objectives that encourage the learning of cross-channel patterns [20]. As foundation models continue to evolve in the time series domain, addressing the challenge of modeling inter-channel dependencies while maintaining the computational advantages of current approaches remains an important area for future research. This could potentially lead to more powerful models that better capture the complex interactions present in real-world multivariate time series data.

Acknowledgement. This publication has emanated from research conducted with the financial support of Taighde Éireann – Research Ireland under Grants [ML-Labs 18/CRT/6183, Insight Centre for Data Analytics 12/RC/2289_P2]. For the purpose of Open Access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

References

1. Ansari, A.F., Stella, L., Turkmen, C., Zhang, X., Mercado, P., Shen, H., Shchur, O., Rangapuram, S.S., Arango, S.P., Kapoor, S., Zschiegner, J., Maddix, D.C., Wang, H., Mahoney, M.W., Torkkola, K., Wilson, A.G., Bohlke-Schneider, M., Wang, Y.: Chronos: Learning the language of time series. TMLR (2024).
2. Bian, Y., Ju, X., Li, J., Xu, Z., Cheng, D., Xu, Q.: Multi-patch prediction: Adapting LLMs for time series representation learning. ICML (2024).
3. Breiman, L.: Random forests. Machine Learning **45**, 5–32 (10 2001).
4. Cox, D.R.: The regression analysis of binary sequences. Journal of the Royal Statistical Society: Series B (Methodological) **20**(2), 215–232 (12 2018).
5. Dempster, A., Petitjean, F., Webb, G.I.: ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. DAMI (2019).
6. Dempster, A., Schmidt, D.F., Webb, G.I.: Minirocket: A very fast (almost) deterministic transform for time series classification. SIGKDD (2021).
7. Dempster, A., Schmidt, D.F., Webb, G.I.: Hydra: Competing convolutional kernels for fast and accurate time series classification. DAMI (2022).
8. Dempster, A., Schmidt, D.F., Webb, G.I.: Quant: A minimalist interval method for time series classification. DAMI (2023).
9. Dhariyal B., Nguyen, T.L., Ifrim, G.: Back to Basics: A Sanity Check on Modern Time Series Classification Algorithms. AALTD (2023).
10. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. ICLR (2021).

11. Eldele, E., Ragab, M., Chen, Z., Wu, M., Li, X.: Tslanet: Rethinking transformers for time series representation learning. ICML (2024).
12. Feofanov, V., Wen, S., Alonso, M., Ilbert, R., Guo, H., Tiomoko, M., Pan, L., Zhang, J., Redko, I.: Mantis: Lightweight calibrated foundation model for user-friendly time series classification. arXiv:2502.15637 (2025).
13. Fix, E., Hodges, J.L.: Discriminatory analysis - nonparametric discrimination: Consistency properties. International Statistical Review **57**, 238 (1989).
14. Foumani, N.M., Tan, C.W., Webb, G.I., Salehi, M.: Improving position encoding of transformers for multivariate time series classification. DAMI (2023).
15. Friedman, J.: Greedy function approximation: A gradient boosting machine. The Annals of Statistics **29** (2000).
16. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. MACH (2006).
17. Goswami, M., Szafer, K., Choudhry, A., Cai, Y., Li, S., Dubrawski, A.: Moment: A family of open time-series foundation models. PMLR (2024).
18. Hoerl, A., Kennard, R.: Ridge regression: Biased estimation for nonorthogonal problems. Technometrics **12**, 55–67 (2012).
19. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. DAMI (2019).
20. Liang, Y., Wen, H., Nie, Y., Jiang, Y., Jin, M., Song, D., Pan, S., Wen, Q.: Foundation models for time series analysis: A tutorial and survey. SIGKDD (2024).
21. Lubba, C.H., Sethi, S.S., Knaute, P., Schultz, S.R., Fulcher, B.D., Jones, N.S.: catch22: Canonical time-series characteristics. DAMI (2019).
22. Middlehurst, M., Ismail-Fawaz, A., Guillaume, A., Holder, C., Guijo-Rubio, D., Bulatova, G., Tsaprounis, L., Mentel, L., Walter, M., Schäfer, P., Bagnall, A.: aeon: a python toolkit for learning from time series. JMLR (2024).
23. Middlehurst, M., Schäfer, P., Bagnall, A.: Bake off redux: a review and experimental evaluation of recent time series classification algorithms. DAMI (2024).
24. Nguyen, T.L., Gsponer, S., Ilie, I., O'Reilly, M., Ifrim, G.: Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations, DAMI (2020).
25. Serramazza, D.I., Nguyen, T.L., Ifrim, G.: Improving the evaluation and actionability of explanation methods for multivariate time series classification, ECMLPKDD (2024).
26. Singh, A., Bevilacqua, A., Nguyen, T.L., Hu, F., McGuinness, K., OReilly, M., Whelan, D., Caulfield, B., Ifrim, G.: Fast and robust video-based exercise classification via body pose tracking and scalable multivariate time series classifiers. DAMI (2022).
27. Wen, Y., Ma, T., Weng, T.W., Nguyen, L.M., Julius, A.A.: Abstracted shapes as tokens – a generalizable and interpretable model for time-series classification. NeurIPS (2025).
28. Wu, H., Hu, T., Liu, Y., Zhou, H., Wang, J., Long, M.: Timesnet: Temporal 2d-variation modeling for general time series analysis. TMLR (2023).
29. Zhou, T., Niu, P., Wang, X., Sun, L., Jin, R.: One fits all:power general time series analysis by pretrained lm. NeurIPS (2023).
30. Ismail-Fawaz, A., Devanne, M., Berretti, S., Weber, J., Forestier, G.: Look into the lite in deep learning for time series classification. *International Journal of Data Science and Analytics*, Springer, (2025).
31. Fawaz, H.I., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D.F., Weber, J., Webb, G.I., Idoumghar, L., Muller, P.-A., Petitjean, F.: InceptionTime: Finding AlexNet for time series classification. DAMI (2020).