amount of points in the TSs without losing information (see Piecewise Aggregate Approximation [92]). Another option is to use the Matrix Profile of minimums of distances (see Def. 3.5.4). This vector computation has been widely optimized with algorithms such as STAMP [292] or SCRIMP++ [91]. Finally, different algorithms can be applied for partially computing the MPlot or zooming-in and zooming-out. Some of these techniques (for example, SPLAT) are detailed in Shahcheraghi et al. [11].

Definition 3.5.4: AB-join Matrix Profile

An **AB-join Matrix Profile** $MP^m_{A,B}$ is a vector of Euclidean distances between each subsequence of T_A and its nearest subsequence of T_B; that is, $MP^m_{A,B} = (min(DP^{k,m}_{A,B}))^{n_B-m+1}_{k=0}$. If $A = B$, the matrix is called **self-join Matrix Profile**.

3.5.2 Using MPlot to analyze the `PulsusParadoxus` dataset

Shahcheraghi et al. [11] show different examples of how an MPlot can give initial insight on TSs data before a deeper analysis. It aids in finding different types of behaviors such as patterns, discords or novelets (emerging patterns not observed before), shapelets (for classification tasks).

Among those examples, they include the `Pulsus Paradoxus` dataset. By checking the complete matrix plot (Fig. 3.27), it becomes evithent that when the TSs T_A and T_B have the same length, the MPlot is a squared matrix. Furthermore, the MPlot is symmetric, which aligns with the inherent property of the distance functions being symmetric ($d(u,v) = d(v,u)$). This property can be used for better memory and time consumption.

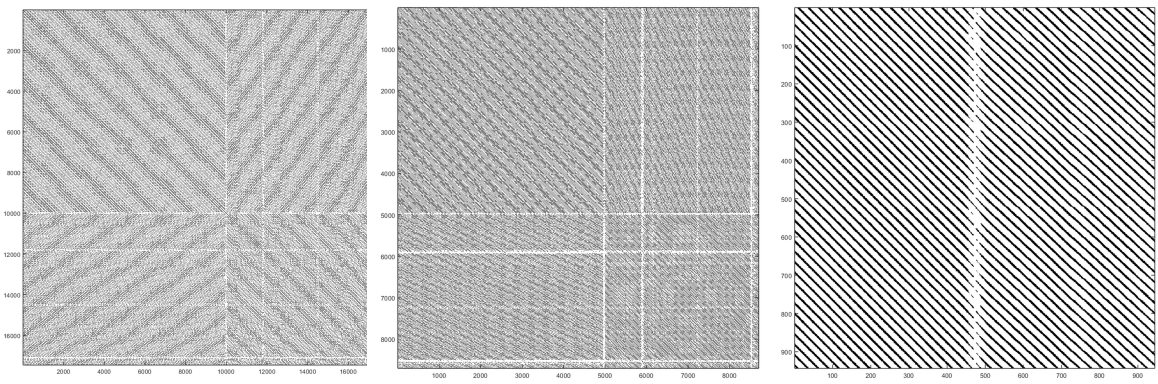

Figure 3.27: MPlot for the `Pulsus Paradoxus` TSs, using sequence length 54 (1 pulsus length). On the left, the complete TSs is used. On the middle, the TSs is subsampled getting the odd elements ($ts[::2]$). On the right, a zoom in the 10k timestamp.

As mentioned in Section 3.2, each heartbeat takes approximately 58 time steps, which means that 2,000 time steps are approximately 34 heartbeats, which is approximately the number of diagonals in the MPlot (see the last MPlot in Fig. 3.27, where subsampling to odd elements produces black bands that correspond to the mentioned "diagonals"). The reason is that the SPO_2 metric repeats the same pattern in each pulsus; when the distances are taken, the sequences starting in $58 \cdot n$ with $n \geq 0$ are very close between them, and so on. Thus, each diagonal corresponds to one beat pattern. There are also white spaces. The white breaks show that in that place the subsequences are different; that is, the pattern breaks. According to the MPlot pattern catalog [293] this behavior means that "The same pattern is reoccurred with an unseen

pattern in between".

If the Matrix Profile is used to look for a pattern, a motif is found (Fig. 3.30) that is really similar to the one in the middle of the red sequence (Fig. 3.31). This may be the "unseen pattern". But, as they mention in [11], it seems that something happens approximately every 8 beats (see Fig. 3.29). Let us see what happens in the blue subsequence (Fig. 3.32). Interestingly, the positions marked with arrows in Fig. 3.29 show that for every eight pulses, there is a heartbeat where the value does not quite reach the "minimum" but remains slightly higher. Meanwhile, the rest of the TSs closely resembles the red sequence. Something similar should happen if the rest of the TSs is analyzed. According to Shahcheraghi et al. [11], the 8 pulsus ratio is due to the relation 1 respiration cycle per each 8 heartbeats. The anomaly detected in $10K$ is due to heart damage (known as tamponade) caused during a surgery that reduces the efficiency of the person in producing oxygenated blood (see Fig. 3.29).

3.6 Foundation models for Time Seriess

The past two years have been especially important for the development of TSs models. At the beginning, Transformers [294] were thought to lose temporal correlation, raising doubts about their use for TSs analysis [295]. Afterwards, in Wu et al. [216], a new autocorrelation mechanism was added that fixed the temporal correlation problem, making transformers adaptable to TSs. Figure 3.33 shows how, only in 2023, different transformer models were adapted to TSs and their history still continues in the present year (see Timexer [296], which adds exogenous variates, or Medformer [297], developed for specific medical classification tasks). Their scalability and flexibility made them easy to adapt for pre-trained models and foundation models, showing great results in other fields like image [298, 299] or text [300, 301], leading to the option of using foundation models for TSs.

At first, transfer learning was thought not to be appropriate for TSs as Woo et al. [302] describe: "Unlike image and text data which naturally share semantic information across datasets and domains, TSs data may not enjoy such properties of transferability, as the semantics of TSs data may be unique to their dataset or domain. As such, it is still unclear how TSs models can benefit from pre-training and transfer learning." However, in the same year, transfer learning for specific tasks appeared, motivating practitioners to no longer train models from scratch when classifying TSs, but instead fine-tune pre-trained models [303, 304, 305]. Eventually, following the positive path, the TSs foundation models appeared, increasingly gaining importance in the field [306, 201, 1, 5].

The following sections introduce transformers, Transfer Learning and, specifically, the application of both to TSs.

3.6.1 Transformers and foundation models

Introduced by Vaswani et al. [294], a Transformer is a Neural Network (NN) architecture that completely changed the context of DL [307]. Originally designed for translating sentences (encoder-decoder design), it has become the foundation of many modern AI models, especially language models (e.g., [300, 301]) but also many areas including education [308] or chemistry [309]. They are based on attention mechanisms that avoid the RNNs' loops and CNNs' convolutions.

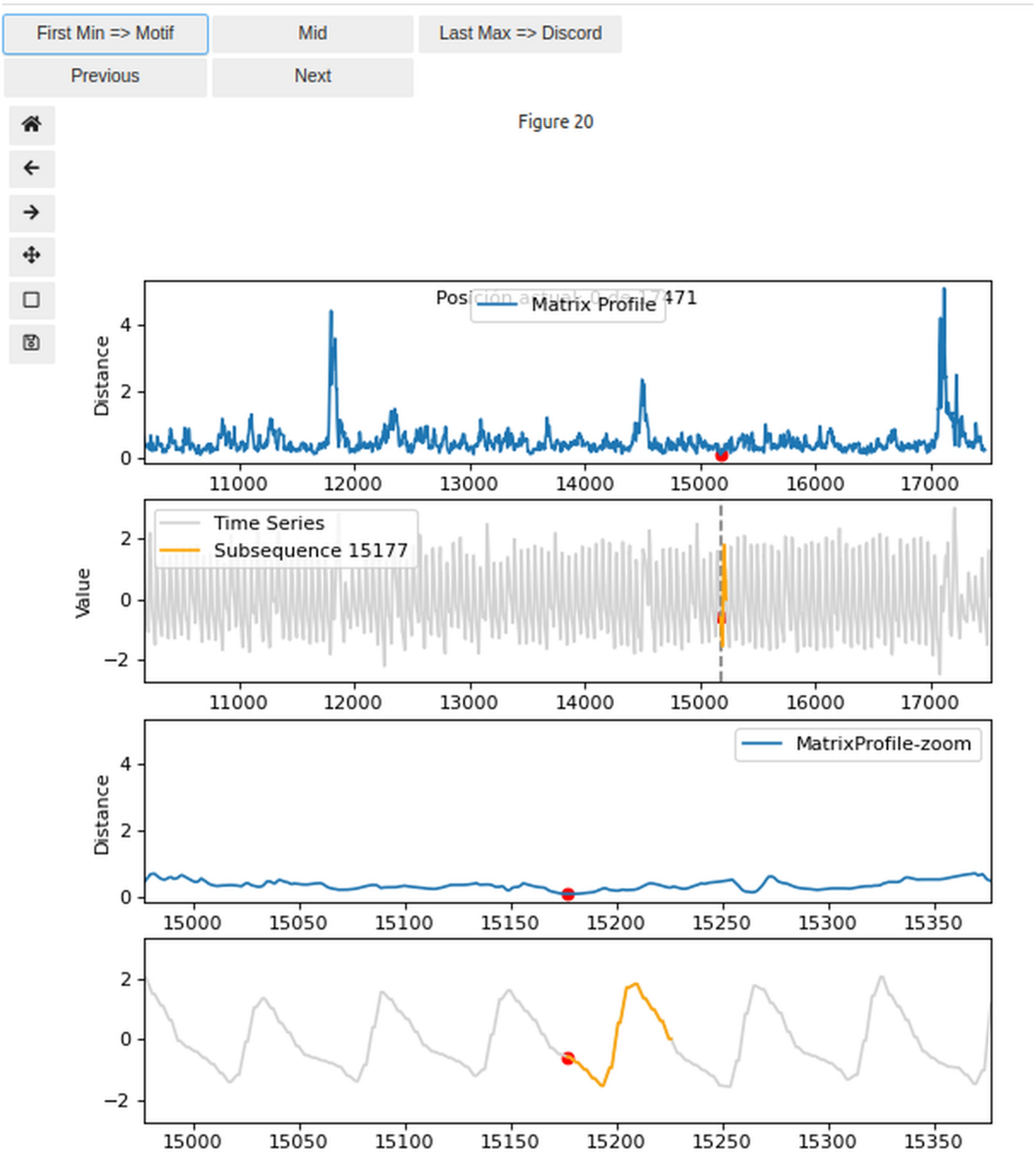

Figure 3.28: Pulsus Paradoxus example Matrix Profile interactive plot. The plot shows the minimum value in the Matrix profile (i.e., the motif).

Transformers process a whole sequence of words (tokens, atomic units) in parallel, allowing them to capture relationships between words independently of their positions.

The architecture is modular, meaning it's built by stacking similar layers on top of each other. Each layer refines the representation of each token. This makes it easy to modify or replace parts (such as the final output layer or "head") for different tasks, which is ideal for reusing models in different contexts or tasks. The original architecture follows the encoder-decoder architecture

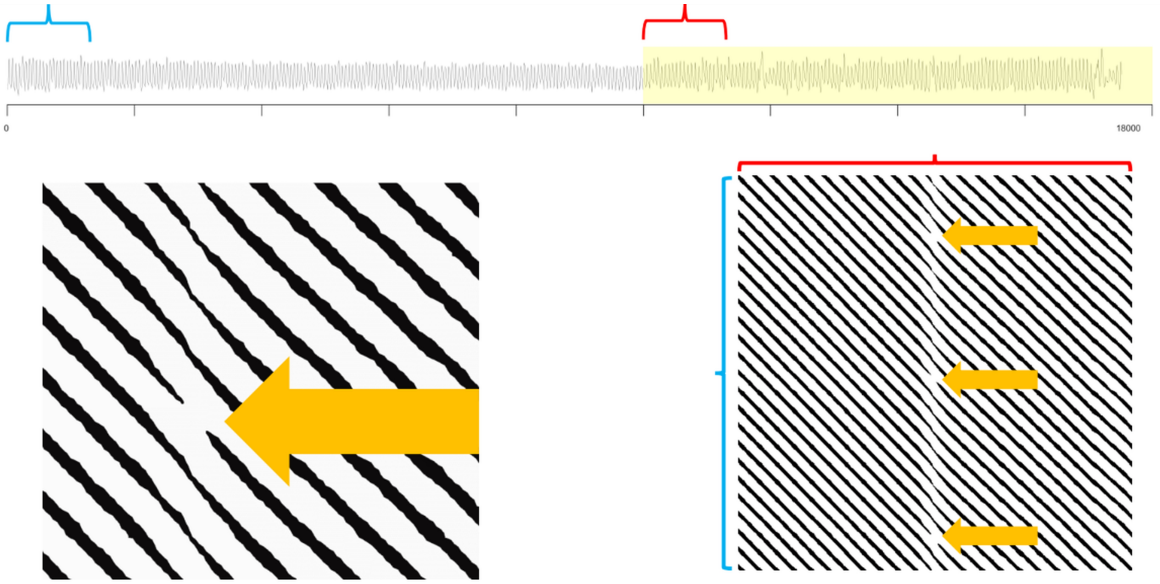

Figure 3.29: Pulsus Paradoxus SPO2 MPlot using SinMat Shahcheraghi et al. [11]. The image is obtained from the notes on https://drive.google.com/Eamonn_repository. It shows a new pattern ocurred each eight pulsus that breaks the most repeated one.

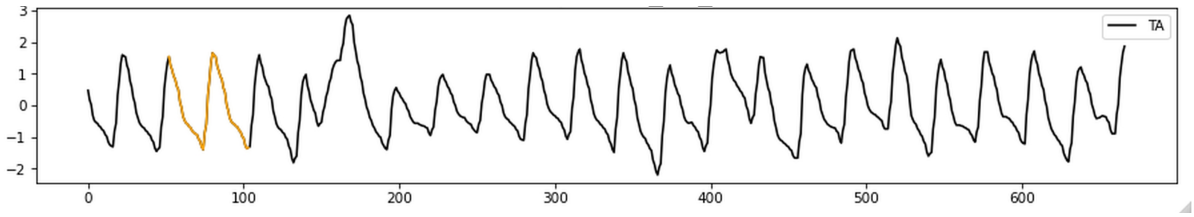

Figure 3.30: Pulsus Paradoxus motif found while looking for the motif using the Matrix Profile. The plot starts at $t_0 = 11530$, with the motif subsequence starting at $t = t_0 + 52 * 2$ with a length of 52.

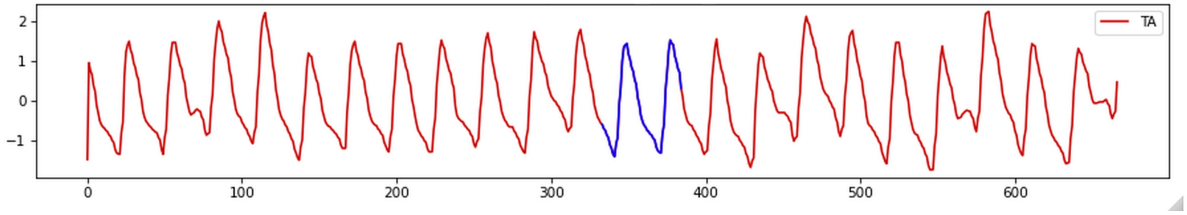

Figure 3.31: Pattern found in [11]: red subsequence (See Fig. 3.29) from $t_0 = 10666$ with length 52. The plot starts in index $t = 10000$ and uses an stride of 2 between timestamps.

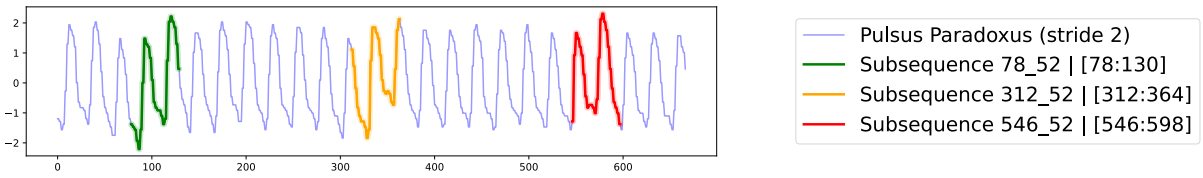

Figure 3.32: Pattern found in [11]: blue subsequences (See Fig. 3.29), starting from $t \in \{208, 624, 1092\}$ with length 52. The plot starts in index $t = 0$ and uses an stride of 2 between timestamps.

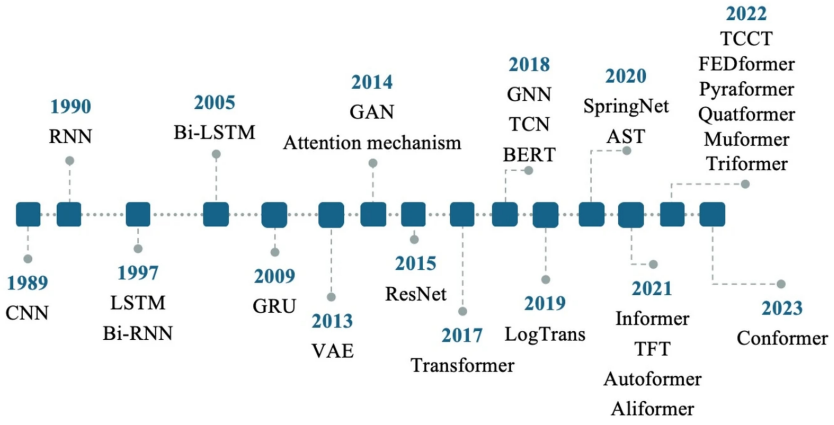

Figure 3.33: Figure 1 in [12]. Showing the historical development of time series forecasting DL.

(see Fig. 3.20). This fact makes them conceptually similar to MTSAE, but for text (discrete data). Next subsections show how Transformers have evolved to allow TSs as input data as well as their use for transfer learning and pre-training, which has become their main application within the TSs analysis.

Tokens: smallest units as input

Transformers work with data in the form of tokens. Before any preprocessing, the input text is broken into tokens (for example, syllables, words, phrases or paragraphs). These tokens are, afterwards, translated into a sequence with token id's for their next processing to numerical embedding vectors. This embedding can be thought of as a list of numbers that represents the token's meaning. These vectors are learned so that similar words (same semantic or grammar proposal) share similar embeddings.

Transformers don't read tokens one-by-one in order but all together, they add a positional encoding to each token's vector to give it a sense of position in the sequence. After this step, a sequence of input token vectors that encode both the word identity and its position has been obtained.

Tansfer Learning and pre-trained models

Transformer-based models are very flexible architectures that support multitask adaptation by simply replacing the final layer, commonly referred to as the **head**. As illustrated in Figure 3.34, their usage can be grouped into two main modalities: **direct execution** and **pre-trained model usage**.

In direct execution, the model is trained from scratch and evaluated on the same dataset and task, without any form of reuse or transfer of previous learning. In contrast, pre-trained model usage starts from a model previously trained on a dataset $\mathcal{D}$ for a task $\mathcal{T}$, and includes multiple configurations:

- **Direct inference**: inference is performed on new data $\mathcal{D}_{\text{new}} \sim \mathcal{D}$[4] for the same task $\mathcal{T}$, without additional training.

[4]Symbol $\sim$ is used to denote when two datasets follow the same distribution.

- **Zero-shot inference**: inference is performed on new data $\mathcal{D}_{\text{new}}$, possibly drawn from a different distribution, for either the same task $\mathcal{T}$ or a new task $\mathcal{T}_{\text{new}}$, still without retraining.
- **Fine-tuning**: the model is partially or fully retrained on $\mathcal{D}_{\text{new}}$ for a potentially different task $\mathcal{T}_{\text{new}}$. This may involve updating all model weights, only a subset of them (e.g., by freezing earlier layers), just the new head, or any combination thereof. When training is done only for a small percentage of the new dataset, it is called **few-shot learning**.

Any setting where knowledge from a model trained on task $\mathcal{T}$ and data $\mathcal{D}$ is reused to improve performance on a new dataset $\mathcal{D}_{\text{new}}$—whether for the same task (in-task) or a different one (out-task)—is considered **transfer learning**. This reuse can reduce the need for large labeled datasets and computational resources during training.

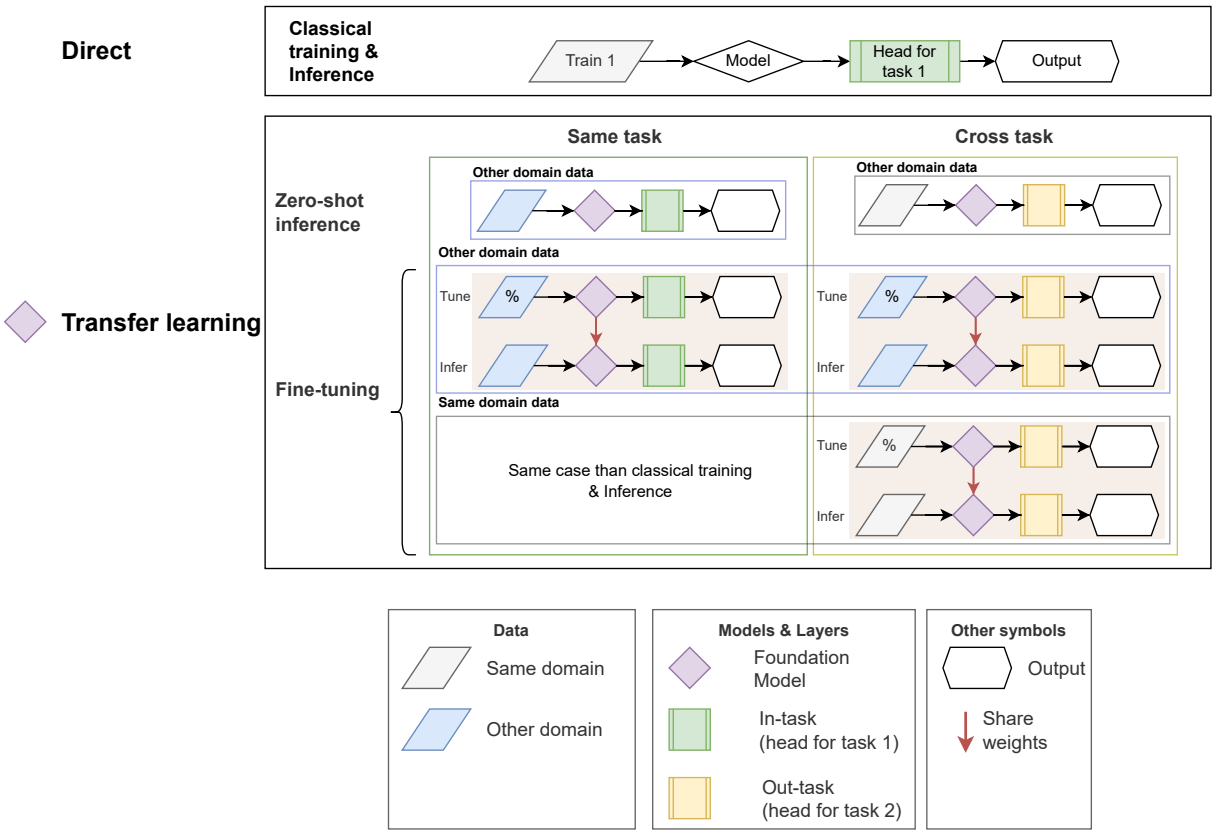

Figure 3.34: Training and inference flow for diferent training and inference techniques. It have been divided into direct training and transfer learning. The lower section distinguishes between in-task transfer (same task, green head) and out-task transfer (different task, yellow head). The fine-tuning involves training the model with a percentage of a new dataset and then infering, it may involve updating the entire model, only selected layers, or just the task-specific head. Data distribution is also considered: blue represents the original domain of the dataset used in direct training (in probabilistic terms), while gray indicates a different (out-of-domain) dataset.

3.6.2 Transformers and foundation models for TSs

Their flexibility, proved in other fields like image [298, 299] or text [300, 301] conduces to the second question. At first, transfer learning was thought to be unsuitable for TSs: "Unlike image

and text data which naturally share semantic information across datasets and domains TSs data may not enjoy such properties of transferability as the semantics of TSs data may be unique to their dataset or domain. As such, it is still unclear how TSs models can benefit from pre-training and transfer learning" [302]. However, in the same year, transfer learning for specific tasks appeared, motivating practicioners to no longer train model from scratch when classifying TSs but instead fine-tuning pre-trained models [303, 304, 305]. This discussion ended up with the aparition of the last key contribution of time series is directly related to TSFM, which are gaining importance in the field, emerging studies such as [306] that gives the data foundations for Large Scale Multimodal Clinical Foundation Models.

As mentioned above, Transformers operate on data structured as tokens, splitting the information into unitary elements such as syllables or words. In the case of TSs, the tokenization is addresed through the use of windows (see Definition 3.5.1. By fixing a window size, the TS can be splitted into equally sized parts. Each of these parts is subsequently treated as a token by the Transformer model.

To use multivariate TSs as input for transformer models, the data must be formatted so that the model can rprocess. To achieve this, the TS is flattened: the values of each feature are concatenated sequentially to form a single input vector.

To the best of our knowledge, the most influential TSFMs, based on the transformer architecture, are the following five:

Chronos An univariate TSs forecasting model based on language architectures [4].

MOIRAI A Masked encoder-based universal TSs forecasting transformer, trained on a Large-scale Open TSs Archive (LOTSA) featuring over 27B observations across nine domains (energy, climate, cloud operations, transport, web, sales, finance, nature, and healthcare) [5].

Lag-LLaMa A decoder-only transformer pre-trained with lag features, leveraging LLaMA to univariate probabilistic TSs forecasting [310].

TimeGPT-1 A proprietary TSs foundation model designed for generative forecasting across diverse domains [201].

MOMENT A multivariate TSs foundation model, designed for multitask forecasting across different datasets [1].

All foundation models have an embedding space that could be used for any of the four tasks where DeepVATS has already been tested: anomaly detection, segmentation, pattern detection, and trend detection. In particular, MOMENT has already been trained for multiple tasks, including imputation, anomaly detection, and classification; therefore, it should achieve fair results within those tasks. In fact, Goswami et al. [1] defends that MOMENT's embedding space fits adequately to the trend of the timeseries, showing interpretable embedding spaces (see Fig. 3.35).

Thus, the integration of MOMENT into DeepVATS could aid in the achievement of an interactive application with shorter training times for large TSs. The next section introduces MOMENT for its further integration into DeepVATS.

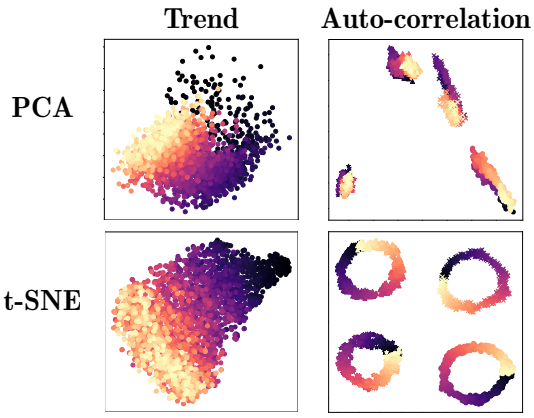

Figure 3.35: MOMENT's embedding space for synthetically generated datasets projection using PCA (top) and t-SNE (bottom) projections. Subtle trend and auto-correlation patterns are captured. This suggests that MOMENT embeddings may encode rich structural features that could be exploited for interactive exploration. Figure modified from [1].

Model	Layers	Hidden Dimension	Parameters (≈)
MOMENT-1 Small	8	512	≈ 38 M
MOMENT-1 Base	12	768	≈ 113 M
MOMENT-1 Large	24	1024	≈ 346 M
DeepVATS-MTSAE	**6**	**128**	**≈ 457K**

Table 3.6: Model sizes and architectures of MOMENT pre-trained variants and the MTSAE architecture.

3.6.3 The MOMENT Time Series Foundation models family

This section gives an overview of the MOMENT TSFM family. This family is formed by different transformer models with three different sized versions pre-trained: small, base, and large (see Tb. 3.6). Also, each version can be loaded in four different modes (tasks): reconstruction, embedding, forecasting, and classification. Each task modifies the structure of the transformer to use a specific output head based on the task to be solved, without modifying the base model (see Fig, 3.36).

Thus, MOMENT learns general representations of the TSs and adapts to different tasks by adding a specific head for each inference mode:

Reconstruction. Used for imputation and anomaly detection. This version is trained to reconstruct missing values within masked patches of the TSs.

Embedding. Used for representation learning, clustering, and classification. In this case, MOMENT generates a fixed embedding vector for each TSs, capturing its essential features.

Forecasting. Used to predict future values in the time series. Moment learns representations and maps them to a forecast horizon.

Classification. Used for labeling the TSs based on learned patterns. MOMENT extracts representations and maps them to output classes.

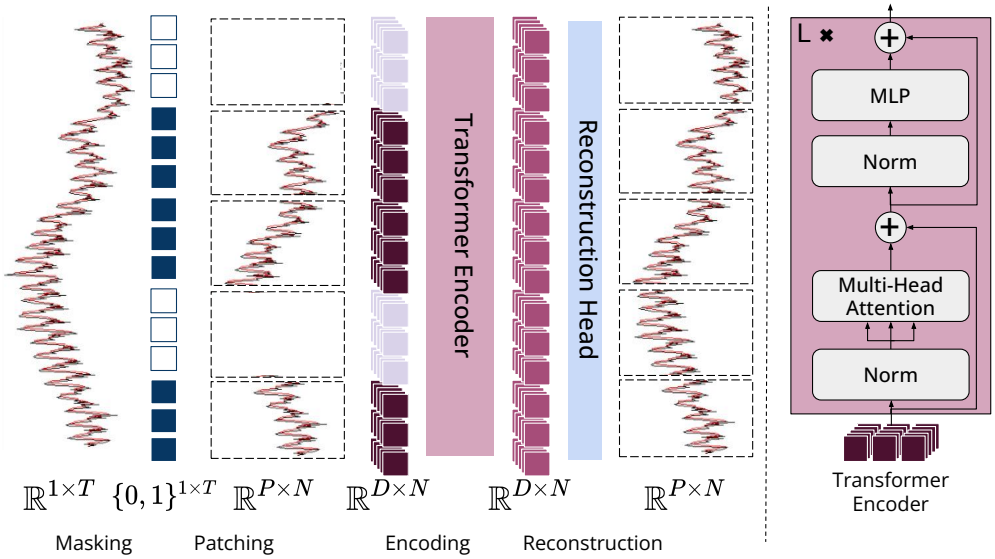

Figure 3.36: Overview of MOMENT. A TSs is broken into P disjoint fixed-length (N) patches. Each of them is mapped into a D-dimensional patch embedding. During pre-training, patches are uniformly masked at random by replacing their embeddings. The goal of pre-training is to learn patch embeddings which can be used to reconstruct the input TSs using a light-weight reconstruction head. It is important to advise that, eventhough the image shows the univariate process, the model is defined for multivariate TSs. Figure obtained from Fig. 3 of [1].

Since the embedding task generates general latent representations of the TSs, it is the most adequate for its integration into DeepVATS as it is not limited to a specific task and models the intrinsecal characteristics of the TSs. From now on the notation "MOMENT-small", "MOMENT-base" and "MOMENT-large" is used to refer to the different sizes of the embedding task version of the model.

Chapter 4

Results

Desde la sed de la extensión
al futuro voy, tras la luz,
por los vientos y el hielo
y la lumbre.

— Antonio S.R.

This chapter presents and analyzes the results derived from two key approaches: the use of MPlot (DM) and the use of Time Series Foundation models (DL). These techniques are expected to enhance visual analytics by improving pattern detection and increasing efficiency in data exploration due to the reduction of training time.

The chapter is divided into two sections. The first section 4.1 describes the integration of MPlot into DeepVATS and its benefits. The second section 4.2.1 shows the integration of TSFM into DeepVATS.

4.1 MPlot integration into DeepVATS

MPlot is a graphical view of the distances between time series subsequences that is the basis of some of the methods in the DM family (STOMP [311], STAMP [292], MERLIN [312], MADRID [313], SCRIMP [91]). These algorithms perform well in the analysis of univariate time series (see Tb. 3 in [8]). These algorithms use, in particular, the matrix profile, which is a feature built from MPlot that has been referred to as a Swiss army knife due to its capabilities in the detection of time series behavior [314].

DeepVATS is originally based on MTSAE. This DL algorithm learns compact representations (embeddings) of time series by masking and reconstructing parts of the data, which allows the identification of patterns and anomalies. The present section focuses on the integration of MPlot as a useful technique from Data Mining into DeepVATS as an interactive visual application for Deep Learning analysis.

This combination enhances the behavioral analysis of univariate large-scale time series behavioral analysis, allowing a quick preview of the results and the possibility of detecting trends within the

visual app. The detection of trends is the result of the representation of the distance between the subsequences of a specific size across the time series. So, if the time series exhibits a trend, the distance increases as the initial timesteps of the subsequences are more separated. Thus, this integration overcomes the initial lack of DeepVATS in distance analysis (see Figs. 2.8, 4.1). The contribution is analyzed using four of the datasets presented in section 3.2:

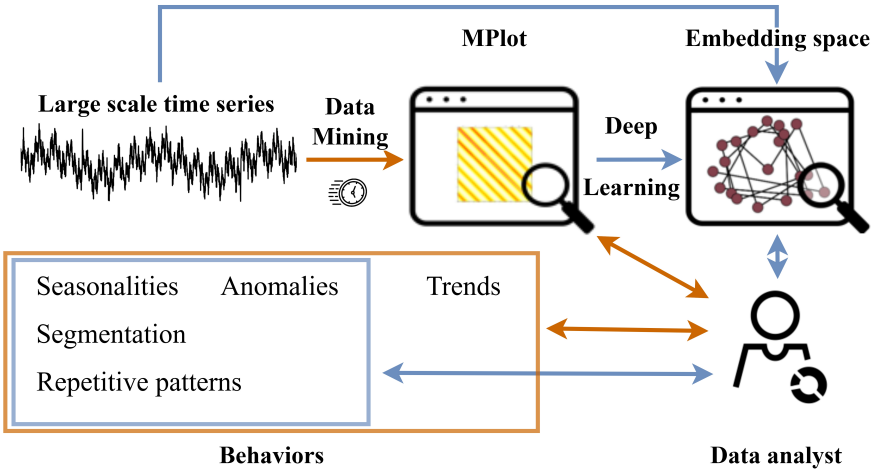

Figure 4.1: Integration of similarity matrix plot into DeepVATS for a previous fast analysis of the long time series. Both images have been adapted from [7]. The schema shows how the analyst can use the MPlot preview analysis, in orange, and the full interactive embedding space analysis, in blue, for checking the behaviors of the time series.

- The `M-Toy` synthetic dataset allows us to evaluate the detection of anomalies within Data Mining technique.
- The `S3` dataset to analyze the capability of MPlots in trend detection.
- The `Pulsus Paradoxus` dataset, which allows us to compare the fast use of the different techniques on detecting patterns.
- The `Solar 4 seconds` dataset, which will be used for the memory and time consumption analysis, as it has more than 7M elements.

This section focuses on the integration of MPlot - as a useful technique from DM- into DeepVATS as an interactive visual application for deep visual analysis. The main goal is to create an interactive visualization for high-quality analysis in order to answer the following questions (see Fig. 1.1):

RQ1 Are MPlot easy to interpret? Specifically, do they surpass DeepVATS' capabilities in behavior representation?

RQ2 Are MPlot efficient enough in their $10K$ version? That is, do they improve DeepVATS' MTSAE effiency?

4.1.1 Efficiency evaluation of MPlot-DeepVATS

To answer **RQ1.1**, a previous efficiency evaluation is performed. The computation of MPlot requires substantial computational resources and execution time, especially as the length of the

time series increases. As shown in Table 4.1, memory errors ocur when analyzing time series from 493K elements. However, the MPlot of the time series downsampled to 10k points, combined with specific zoom techniques, still provides valuable insight into the global time series Shahcheraghi et al. [11]. Therefore, the MPlot is computed using a maximum number of points (by default, 10k), resulting in significantly reduced processing times of 5 to 7 seconds for an initial view of the time series behavior. In addition, Matrix Profile-based algorithms have demonstrated good performance in different domains [315, 316, 317]. For these reasons, the Matrix Profile and the MPlot have been selected to address the shortcomings in DeepVATS. The interaction between the MPlot, the matrix profile plot, and the time series plot has been included to check the relation to the results obtained within the MTSAE algoritm.

Table 4.1 shows the computation time necessary to analyze the time series behavior before and after adding MPlot to DeepVATS.

Table 4.1: Computation time for DeepVATS first analysis, both before and after incorporating MPlot.

Time Series		**Execution Time (seconds)**		
Frequency (s)	**Elements**	**DeepVATS**	**DeepVATS + MPlot (Full)**	**DeepVATS + MPlot (10K)**
4	7,397,222	144,850.00	OOM Memory error	6.93
20	1,479,445	29,075.00	OOM Memory error	7.08
60	493,149	9,600.00	OOM Memory error	4.98
300 (5m)	98,630	1,885.71	546.06	7.42
600 (10m)	49,315	900.00	46.34	7.30

This table is based on the execution of DeepVATS for the `Solar 4 Seconds` dataset (see 3.2). This data set has been downsampled following the guidelines proposed in [89]. This downsample results in five time series with frequencies from 4 seconds to 10 minutes (first column) and $50K$ to $7.4M$ elements (second column). The third column of the table shows the execution time necessary for training the MTSAE model with a stride of 1, 100 epochs, and window lengths from 460 to 100. The fourth column shows the execution time needed for computing both MTSAE and the MPlot for the full time series. The fifth column shows the same computation but downsampling the time series to approximately $10K$ elements using Principal Components Analysis. These executions show that the tool cannot directly use the full MPlot due to the intensive memory usage required for calculations and storage, although the execution time for small time series may be faster than MTSAE training. However, the execution of the downsampled time series MPlot is very fast, making it a great tool for previsualizing the time series for further analysis.

Thus, the integration of MPlot into DeepVATS expands its usability as it allows another perspective on patterns and anomaly detection. In addition, this integration provides an interactive way to analyze MPlot within the MP and the time series. The only thing similar to this integration found is the MPlot Explorer[1], by Zach Zymmerman, which has been the baseline for our code alongside the instructions in [11] and the basis code for computing and visualizing MP[2], which has the problem that it is implemented in `MATLAB`. There is a version in the library `SCAMP` based on its same name MP algorithm [90], however, by the time this research is being undertaken, the library warns for experimental code in some parts. So, the iintegration is done through this

[1] Available in `https://github.com/zpzim/mplot-explorer`.

[2] Available in `https://www.cs.ucr.edu/~eamonn/MatrixProfile.html` and `https://sites.google.com/ucr.edu/mplots/codes?authuser=0`.

library and the widely extended STUMPY [272] as well as naive brute-force algorithms to provide a flexible library while new implementations in Python emerge.

It is especially important to note that the plots are done following the MATLAB convention to ensure the integrity with the previous work. Therefore, caution is needed when analyzing when analyzing the ordinate (y) axis. The origin will always be at the top of the image rather than at the bottom (see Figure 3.27).

Taking those details into account, the Distance Profile, Distance Matrix, MP, and MPlot computation and plotting have been implemented to get a basic code for computing and visualizing them.

Summarizing, the answer to **RQ1.1** is that, using the 10K version a good overview of the TSs can be obtained in very short time frames, which does not allow for a full analysis (as information will be lost in the reduction of the number of points) but allows a fast preview while the MTSAE model is training.

4.1.2 Using MPlot-DeepVATS in specific Use Cases

This subsection focuses on how to use the new MPlot-DeepVATS tool, and how it can be used to generate, analyze, and visualize new patterns in some specific use cases (M-Toy, S3, and Pulsus Paradoxus datasets).

The Visual Module includes a new tab "MPlot" (see Fig. 4.2), that allows to easily view the MPlot, the Matrix Profile, and the compared time series all within a single tab. Also, as it is a tool for univariate TSs, the interfaces allows for the selection of any of the variates of the currently loaded dataset, proposing as window length the first results from the Fourier Transform within the implementation in aeon library [318]. The interface also includes a selector to specify the maximum number of points to compute for the MPlot (for balancing execution time, resource consumption and precision), and adjust the subsequence length (window size). Instead of directly selecting columns and rows, the analyst interact with the time series; zooming in with the selector automatically updates the MPlot. The MP is also interactive, enabling to explore motifs and discords and view their positions in the MPlot, as well as the corresponding subsequences in the time series below. In addition, there are other selectable parameters, including computation methods, making the workflow seamless and fully accessible within the visual app. In the following, some selected use cases are shown to demonstrate how the new tool can be used to identify and discover patterns in the time series analyzed. Figure 4.3 shows a full view of the "MPLOT" tab.

Use Case 1 - M-Toy

The M-Toy dataset (see Section 3.2) has three variables with two known anomalies that will allow us to test MPlot for the anomalies detection task. The main problem with M-Toy is that it is multivariate, but MPlot can only work with 1-column data. However, $T3$ is known to be the less representative variable when trying to detect the two main sequence anomalies of the time series (see Fig. 3.16). Thus, the MPlot for $T1$ and $T2$ have been generated (Figs. 4.4, A.1).

However, it fails to provide a clear visualization of the anomalies as there is no clear diagonal break. By checking the catalog [3], it is likely that the TS contains a lot of repetitive patterns

[3] UCR catalog for understanding MPlot https://sites.google.com/ucr.edu/mplots/catalog

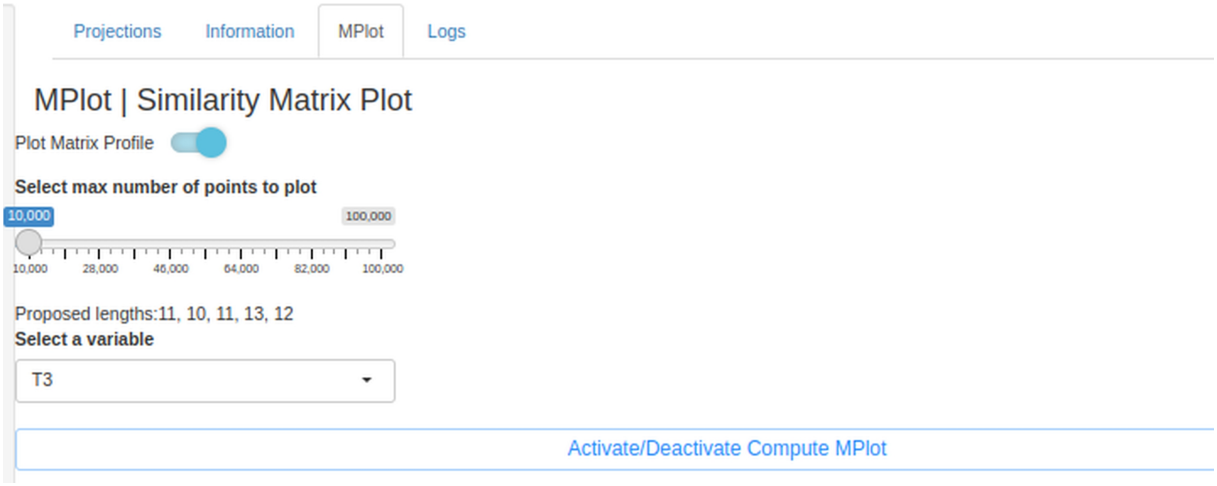

Figure 4.2: `M-Toy` MPlot in the Visual App.

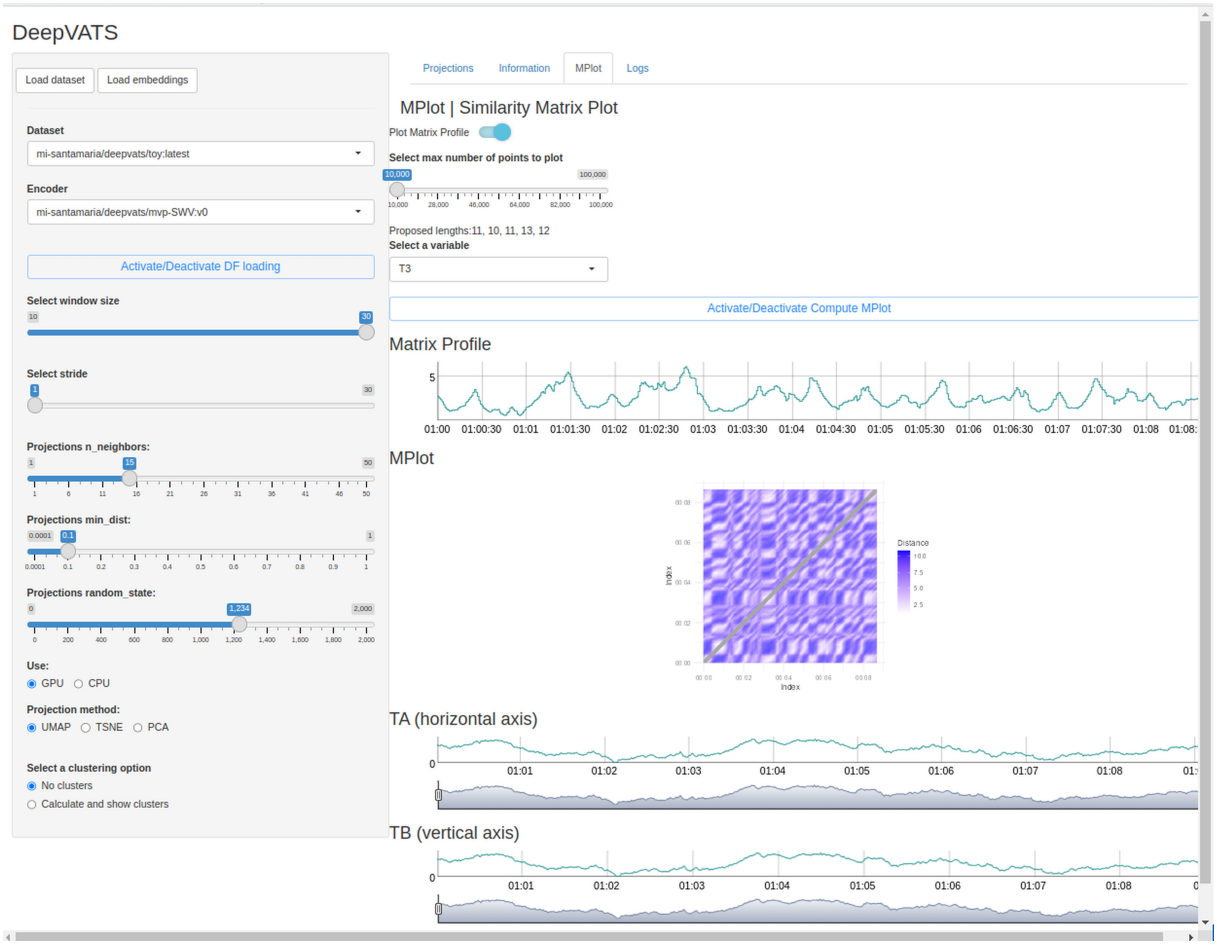

Figure 4.3: M-Toy MPlot in the Visual App.

with decreasing frequency. The only thing that is equal in 4.4 is that there is a large dark zone that reduces to a diagonal line just before reaching the anomaly indexes. This makes the idea that some parts of the time series have been really similar in that zone and suddenly start to be

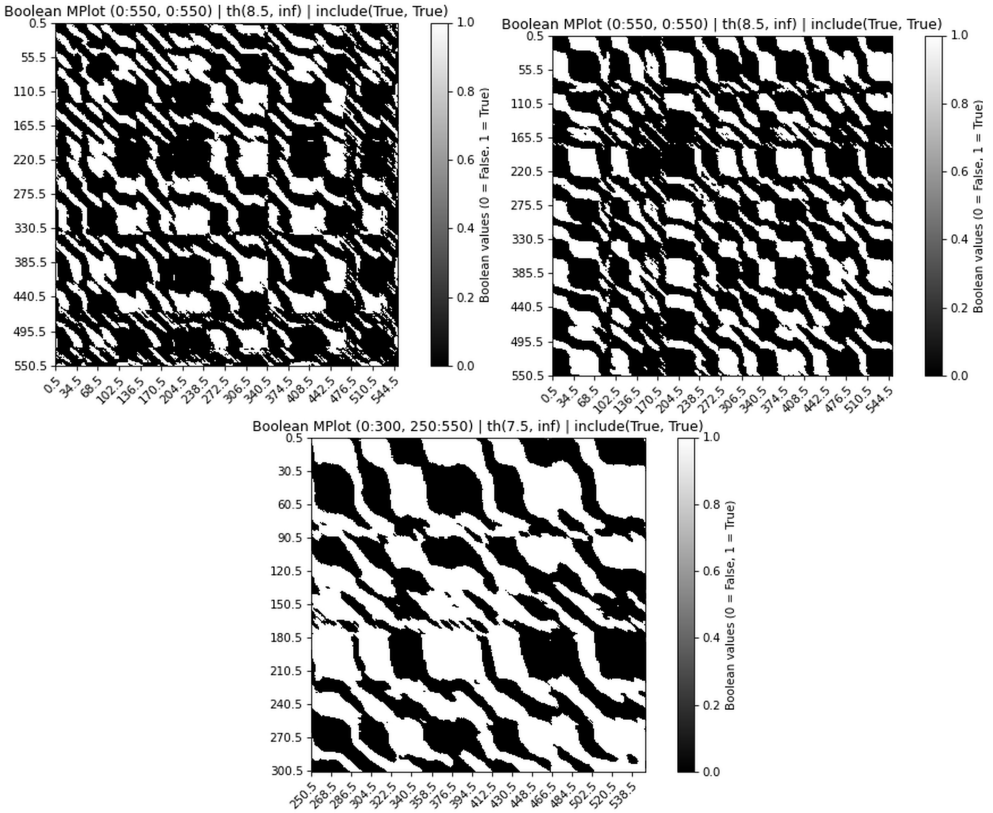

Figure 4.4: MPlot for the `M-Toy` $T1$ variable. The first row shows the full MPlot. The second row displays different zoom levels.

different.

In the case of $T1$ and $T2$ (Fig. 4.4), the $T1$'s plot has a diagonal break in the `150` timestep (horizontal time series), which may suggest the presence of a rare motif or anomaly. However, it does not seem to be a good idea to try to analyze a single MPlot to gain insight into multivariate time series.

Then, related to **RQ1.2**, using MPlot to check sequence anomalies in multivariate TSs seems not to be a good idea and the use of MTSAE already gives easy and valuable interpretability (see Fig. A.2).

Use Case 2 - Looking for trends, S3

In [7] it was shown that DeepVATS was not especially good at detecting trends. To check how the use of MPlots affects the detection of trends, `S3` has been analyzed.

The `euclidean` distance is useful for identifying trends in a time series because it preserves the magnitude of the data. In contrast, the `z-normalized euclidean` distance is not suitable for this purpose, as it standardizes the data, thus deleting the inherent value information (but the repeated pattern that changes its height is easier to see in the z-normalized version).

SCAMP can only be computed using the `z-normalized euclidean` distance as it does not allow the specification of a custom distance function. Fig. 4.5 shows this effect by running the MPlot

time series with the Euclidean distance for STUMP and the z-normalized Euclidean distance for SCAMP.

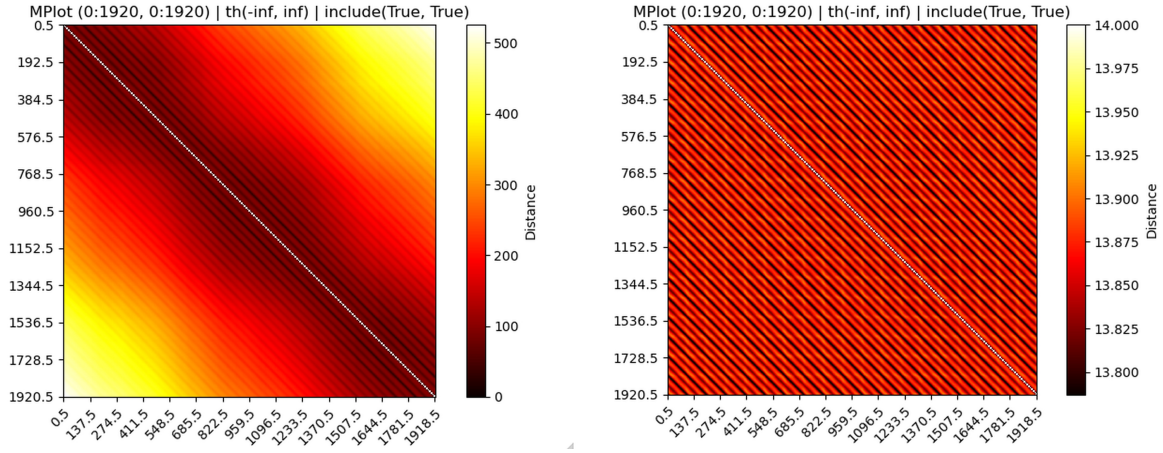

Figure 4.5: MPlot for S3. On the left, STUMP with Euclidean distance is used. On the right, using SCAMP with the `z-normalized euclidean` distance.

However, it is not possible to know with a distance function if this trend is upward or downward, as they are symmetric: the distance will be the same whether going up or down. But in DeepVATS the Distance Matrix can be computed using brute-force; this version does not ask for properties to the "distance" function. Thus, an specific function to detect trends can be defined so a "pseudo-MPlot" is obtained. The most basic: $d(\vec{u}, \vec{v}) = \mu_u - \mu_v$. As Fig. 4.6 shows, the MPlot generated with this function clearly shows the upward and downward trends. Also, memory can still be optimized by saving only one of the two main triangular submatrices of the pseudo-MPlot in a triangular matrix, since $d(u, v) = -d(v, u)$.

Then, related to **RQ1.2**, using MPlot to get an idea of global and local trends within the TS.

Use Case 3 - Pulsus Paradoxus

The third use case is the `Pulsus Paradoxus` dataset, analyzed in Shahcheraghi et al. [57]. It is important to note that the DeepVATS execution pipeline must be checked to train the model before trying to analyze the `Pulsus Paradoxus` embedding space in the visual app. The DeepVATS pipeline is followed to analyze the dataset; that is: initialize and configure Artifact, load the dataset to W&B, train the dataset executing the related notebooks, and open the app and check both the MPlot and the embedding space.

After some interactions with the `Pulsus Paradoxus`' embedding space projection (see Fig. A.3), the conclusions are the following. The MPlot is a great way to analyze the time series data to find repeat patterns (heartbeat-SPO_2, see Fig. 3.29) and novelets (emergent patterns characterized by sudden and repeatted drops in SPO_2 levels over time). It also proves effective in detecting breakpoints in the time series: indexes $10K$, $12K$, and $17K$ in Fig. 4.7 where the plot looks "similar" until that moment and suddenly changes after that point.

However, it is more difficult to check in the MPlot the relation between the time series and the patterns and segment the different patterns that have been found. The interactiveness between the MPlot and the time series facilitates this process, but it is still easier to visually segment the

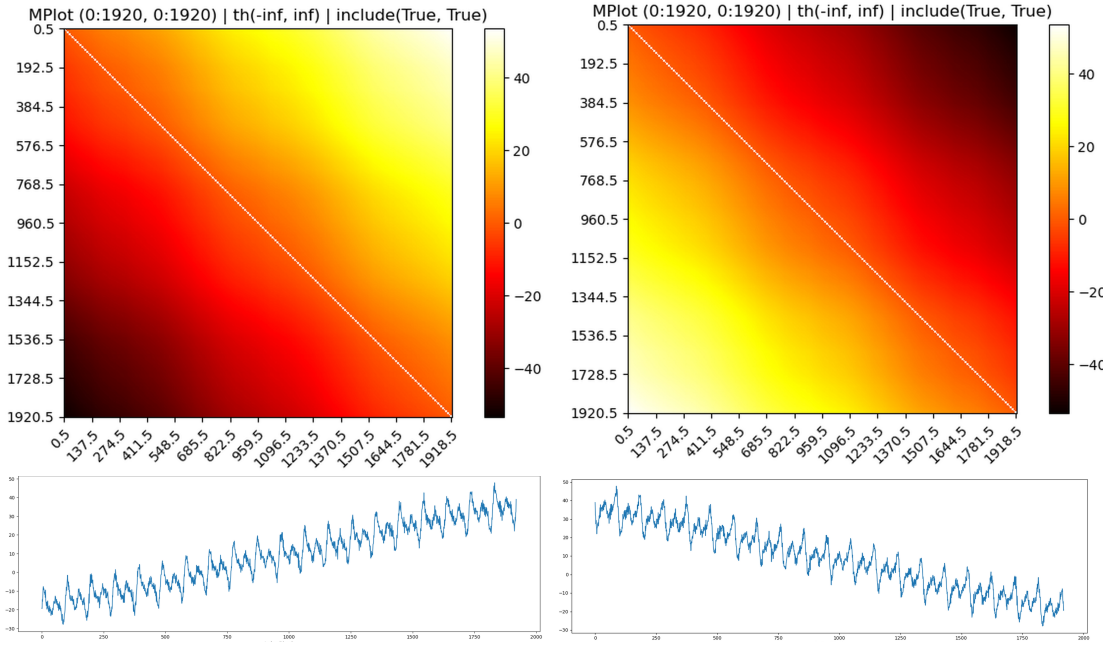

Figure 4.6: S3 pseudo-MPlot using the proposed no-distance function to check a possible trend. By the left, the original time series is used, with an upward trend. By the right, the symmetric time series, with a downward trend.

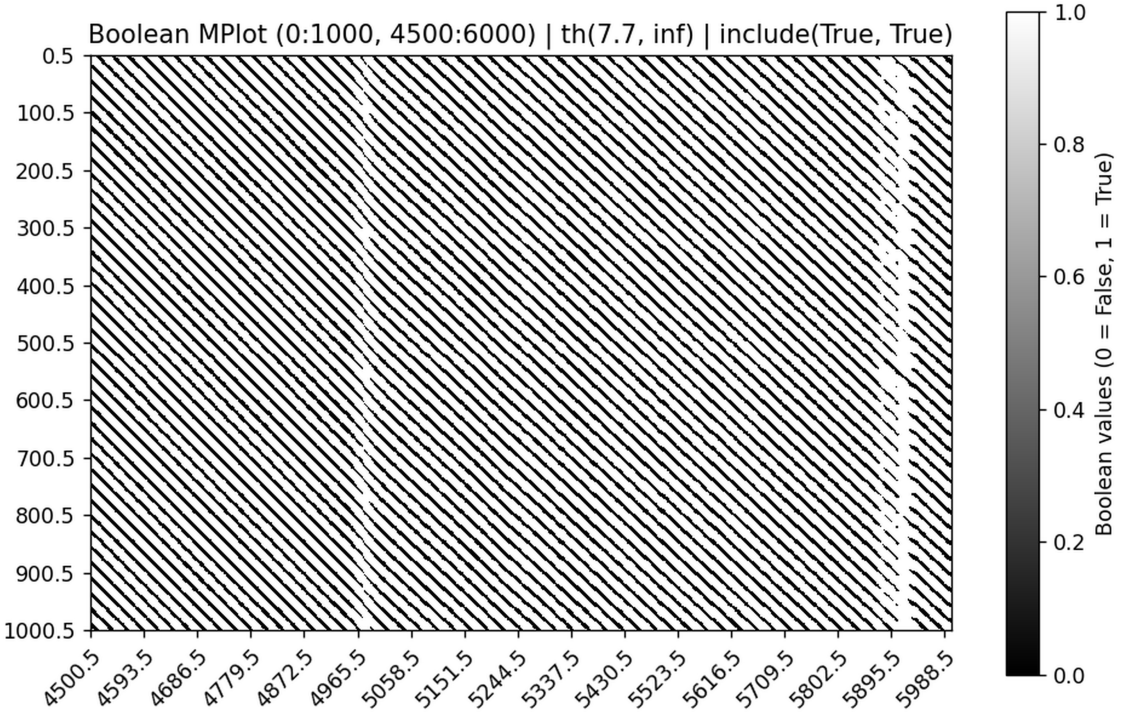

Figure 4.7: MPlot computed using *scamp* and z-normalized euclidean distance, with $threshold = 7.7$. In the position 5k corresponding to 10k, a pattern similar to the one shown in Fig. 3.28 is observed. At position $5.8k$, another repetitive pattern is found.

time series patterns by checking the clusters than by going through the MPlot (see Fig. A.3). Seasonalities and repetitive patterns are easy to check in both plots. Anomalies can be seen on both plots, but depending on the parameters, it can be easier to see in the embedding space as an apart-point or in the MPlot as a white space. Together, they allow an outstanding analysis of univariate time series, giving a positive answer to **RQ1.2**, being a good tool to preview anomalies and patterns in a fast preview.

4.1.3 MPlot contributions to DeepVATS

DeepVATS has demonstrated excellent capabilities for the analysis of univariate and multivariate time series based on the visual interaction with the embedding space of Machine Learning models, more specifically, DL models. By introducing MPlot in a more efficient and flexible version (including the specific function to measure trends), DeepVATS can be leveraged in univariate time series analysis.

Also, the integration with MPlot results in an easy interactive app that allows interaction with the MPlot to visualize the relationship with both the time series and the MP. It also gives new powerful functionality for a fast preview of the behaviors of the time series, resulting in a plot obtained in less than 10 seconds (see Table 4.1), enhancing the interactiveness of DeepVATS. Although MPlot is not directly applicable to multivariate time series, the MatrixProfile has an analog definition for this case. The addition of multidimensional MP is future work. Adding MP-based features to the time series as another variable for model training could be interesting for concise analysis.

Finally, the recommendation within MPlots is to use it for univariate analysis, allowing concurrent backbone model training and time series analysis for a global understanding of its properties.

4.1.4 Future lines to enhance MPlot-DeepVATS

Although MPlot cannot be used for multivariate TSs, the MatrixProfile has an analog definition for that case. The addition of multidimensional MatrixProfile is future work. Adding MP-based features to the TSs as another variable for model training could be interesting for concise analysis. Furthermore, some new foundation models [1, 4, 5] are emerging that can be useful to detect trends and could enhance the analysis within DeepVATS. Therefore, to improve the development and functionalities of the DeepVATS tool, there are two lines of future work:

- **FL1** Checking the use of MP-derived features (such as the multidimensional MP) for better analysis.
- **FL2** The inclusion of new TSFM to the DL module.

Both options propose a next step in the application of the third solution proposed in Section 3.4 (adding new tools from the state-of-the-art). As the next section shows, the final choice was to take a step back to DL, evaluating and integrating foundational models into the DL module of DeepVATS. Figure 4.8 summarizes the future lines and their relation with the RQs as well as how the first hypothesis guides to the second one in the seek for a complete tool.

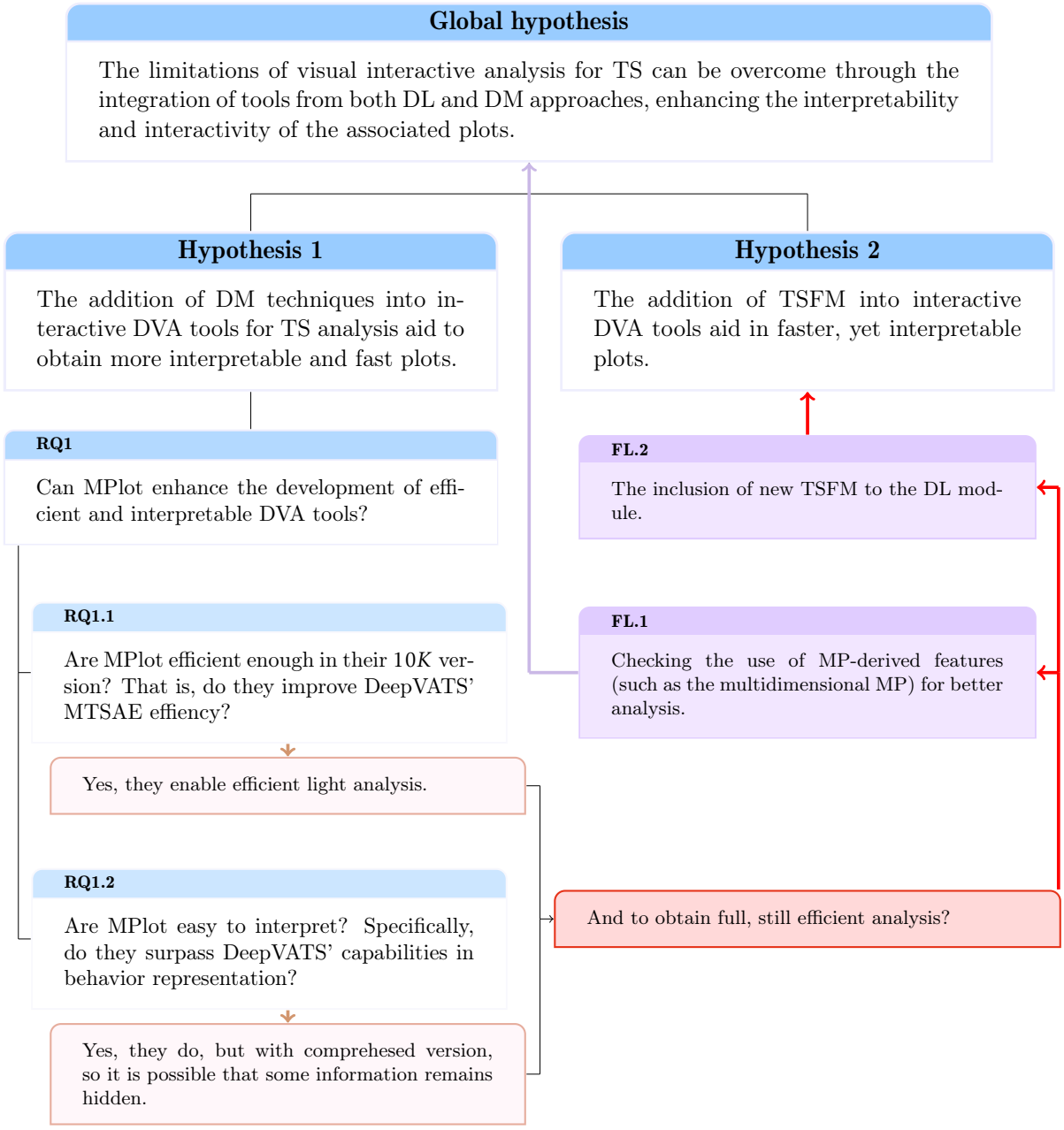

Figure 4.8: Future lines I: schema showing the relation between the hypothesis, the research questions of the DM path, their answers and the associated future lines.

4.2 Time Series Foundation models for interactive visual analytics

Understanding the embedding space generated by different algorithms has been a major focus in Deep Learning (DL). Different interactive visualization tools support dataset analysis by exploring these embedding spaces. Initially developed as a DL-based tool, DeepVATS aimed to analyze the embedding space of long TSs by leveraging DL models to extract vectorized representations, offering a more intuitive comprehension of their structure. However, the high computational cost of this approach limits interactivity as dataset size increases. To enhance the interactivity and with an eye in the Data Mining field, MPlot (similarity matrix plot) [11]

was integrated into DeepVATS. Although computationally expensive, these methods have been successfully scaled for large-scale TSs exploration. The integration of MPlot into DeepVATS has significantly enhanced trend detection capabilities for univariate TSs while also providing a fast preview of the series' properties [319] This combination facilitates an initial exploration of univariate Time Series while the DL model is still being trained, improving the overall workflow efficiency. However, this improvement is still far from the goal of DeepVATS: an interactive tool for visual analysis of both univariate and multivariate time series.

4.2.1 MOMENT integration into DeepVATS

In other domains like computer vision and natural language processing, Transfer Learning (TL) has proven effective at reusing the model knowledge across datasets, enabling faster analysis and reducing training time in new scenarios. Bringing this efficiency into the visual analytics of TSs could significantly enhance the interactivity. However, unlike image or text data, TS does not always exhibit shared semantic information between datasets and domains - the same change on the shape of a TS could be an anomaly in machine analysis but not in stock analysis. This raises important questions about how well TL can be adapted to TS [302]. However, TL is used for different TS tasks, such as forecasting workload on cloud platforms [304] or forecasting influenza hospitalization with small datasets [305]. Also, Fawaz et al. [303] motivates practitioners to no longer train models from scratch for TS classification but instead to fine-tune pre-trained models, based on the idea that models can be adapted. But this still raises various questions. First, "is the size of the dataset the only important thing or is context explicitly necessary too?" Second, "even in the same domain, is it possible to swap semantics between datasets?", "is it enough with a fine-tune or does a TS model need extra semantic information for building the TS?".

Following this path, recent studies have shown great advances on TSFM, with good performance in the different tasks (classification, anomaly detection, etc.), leading to their widespread adoption in the field. New TSFM [1, 4, 5] are appearing that could enhance visual analytics by giving more interpretable, fast and interactive analysis. Specifically, the MOMENT family has been analyzed to show interpretable embedding projections for different synthetic datasets (see Fig. 3.35) and to have good performance against the state-of-the-art [1].

In this context, three main questions arise for the field of visual analytics (see Fig. 1.1):

RQ3 How well do TSFMs' embedding space capture the behaviors of TS? Are they easier or, at least, similar to interpret than DeepVATS' MTSAE?

RQ4 Do quantitative improvements in the classical tasks result in embeddings space interpretability enhancement?

RQ5 Does fine-tune enhance the latent space interpretability? How much tuning is required to obtain high quality descriptive clusters for a specific time series?

Recent studies have shown great advances on Time Series Foundation models for time series, with good performance in the different tasks (classification, anomaly detection, etc.), leading to their widespread adoption in the field. Specifically, some new TSFM [1, 4, 5] are appearing that can be useful for detecting trends and could enhance the analysis within DeepVATS (without losing the current capabilities for anomaly and pattern detection and segmentation). Specifically, the Moment family [1] has been analyzed to show interpretable embedding projections for different synthetic datasets (See Fig. 3.35) and good performance against the state-of-the-art (See Fig. 4.9).

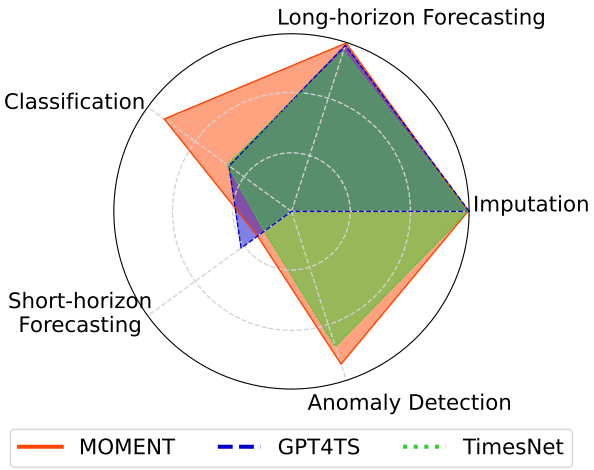

Figure 4.9: Figure 4 in [1]. Shows the performance of Moment against other state-of-the-art time series models: GPT4TS [13] and TimesNET [14].

As exposed in Section. 3.6.1, between the available models, MOMENT is the most adequate for general-purpose interactive visual analysis.

The next sections analyze the TSFM, including a statistical analysis of how much training is necessary to tune it to specific datasets and a visual analysis of the embedding space using the same projection techniques as MTSAE (PCA followed by UMAP) to ensure a valid comparison.

In order to analyze the integration of MOMENT into DeepVATS in an equitable manner, most of the TS that have already been analyzed [7, 89] have been selected, focusing on `S1`, `S2`, `S3`, M-toy and Kohl's with the original window sizes and strides. Dataset `S1` is used to analyze segmentation, `S2` for anomaly detection, `S3` for trends, M-toy for anomaly detection, and Kohl's for both segmentation and trend detection.

The goal is to check that MOMENT achieves - at least - the capabilities of Masked Timeseries AutoEncoder model, with an embeddings projection space easy to check for anomalies, patterns and segments; and also if it adds the trend detection capability. This fact would make DeepVATS more interactive and would solve the question **RQ2.1**.

4.2.2 Analysis of `S1` using the zero-shot version of MOMENT-small

The execution of Moment for `S1` in the same configuration as in the original research paper results in five very interlaced clusters, like a tangle. Four of them cover the whole series and the other one seems to be getting the whole series but not the df_2 segment (see Fig.4.10). Making a zoom in on all the clusters (figures A.4, A.5, A.6, A.7, A.8), MOMENT is detected to be trying to get patterns, but they do not seem to be very clear. Also, the only segment that may be detected correctly is df_2.

This fact leads to two questions. The first one is: "Is moment trying to learn more specific information and the size of the window is hiding it?". And second: "Can I force moment to learn specific information of the time series with few shots of the data?" (**RQ2.2**). Based on the fact that foundation models work better when fine-tuning is applied, the research focuses on the second one, analyzing the amount of dataset MOMENT needs to know to get better insights of

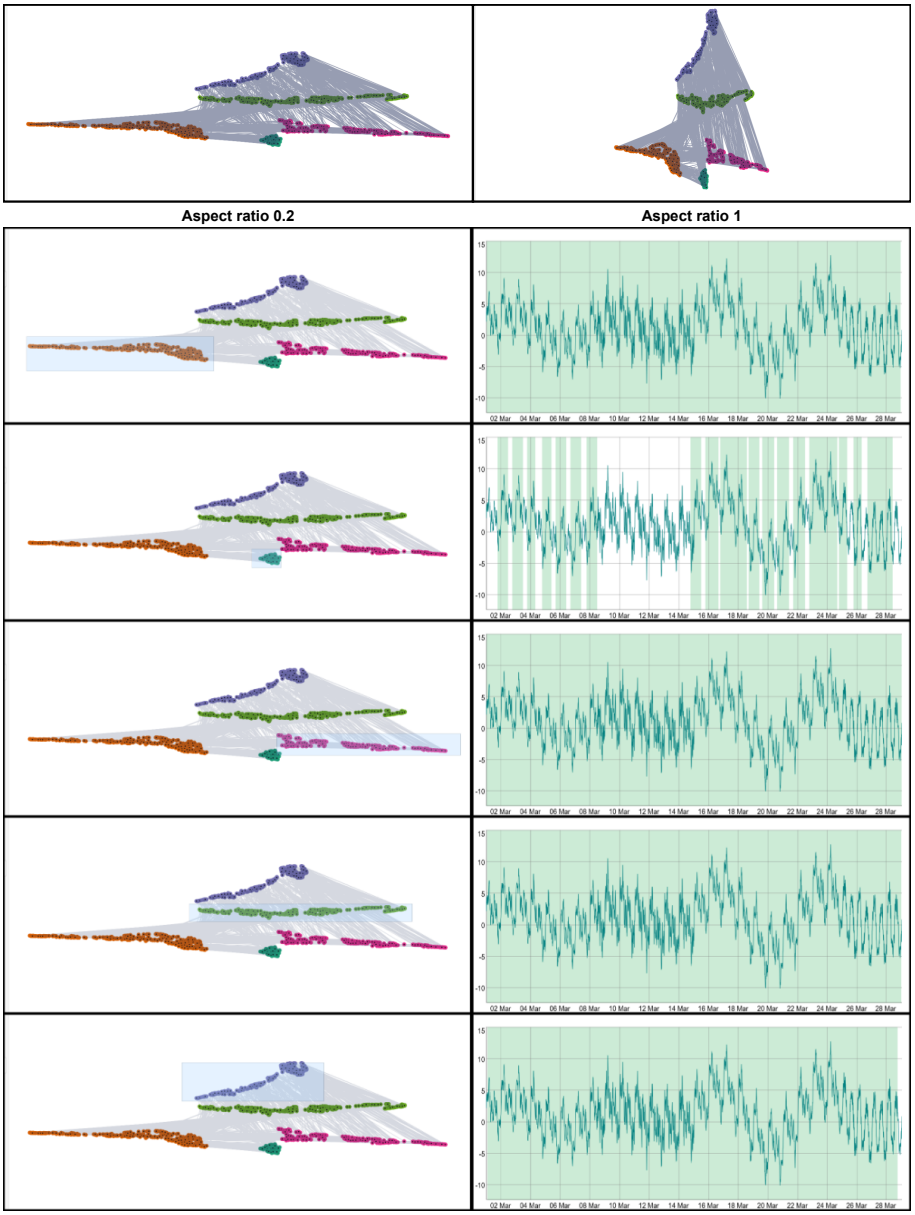

Figure 4.10: Execution of MOMENT-small for `S1` taking the mean in batches of 20 minutes. The execution is done for window length 54 (up) and stride 2. The parameters for dimensionality reduction (using PCA followed by UMAP with GPU) are `n_neighbors`= 15, `min_dist`= 0.1, `random_state`= 1234. The parameters for cluster computation are `metric=euclidean`, `min_size`= 40, `min_samples`= 15,`epsilon`= 0.08.

it in the inference step.

4.2.3 Fine-tune in DeepVATS

For fine tuning, a wrapper has been built so that any torch-based model can easily be included in DeepVATS's fine-tuning process. The fine-tuning is done like an imputation task training, in a similar way to MTSAE training, thought for the imputation task: based on the percentage of the training dataset and masking part, use and validate the model for the reconstruction task

using both a validation and train dataset.

The goal is to check if this reinforcement makes MOMENT fit better the TSs and to check if the PP shows this enhancement. The data will be selected from different parts of the TSs to avoid focusing on a specific/special part of time series.

The fine-tuning algorithm is based on four main parameters:

- `window_lengths`. Hosts the window sizes that will be used for the batches in the fine-tuning.
- `training_percent`. This value is the percentage of the total percentage of the dataset that will be used to get the training batches.
- `valid_percent`. This value is the percentage of the total percentage of the dataset that will be used to get the validation batches.
- `mask_percent`. This value is the percentage that will be masked in each batch.

The selection of the window lengths and the position of the selected windows within the time series resulted in two different versions of the fine-tuning algorithms, modelled in the code based on a *mix_windows* flag:

Mix Windows: `False`. This version loops through `window_lengths`, splitting the total dataset for the curreng `wlen` size and randomly selecting (within the `training_percent` and `valid_percent` percentages) windows across the time series.

Mix Windows: `True`. . This version splits the time series using the validation and training percentages. The validation and training datasets are built from the beginning of the time series. The first part will be the training dataset and the second part, the validation dataset. The batches are dinamically built getting a random window length from `window_length` in each batch. To ensure the integrity of the training and validation batches, the values of the lengths are cached so each validation is done within the same batches.

The first version did not show much enhancement and seemed to be skewed as each window size re-trained the entire model just after the previous one. This sequential process may have caused the model to alternately learn and forget patterns at each step.

The second version uses varying window sizes in each batch to create a more stochastic process. This approach gives the model a global view of the TSs, allowing it to identify patterns across different scales while preserving information from previous iterations of the training loop. This second version results in improvements of more than 20% within the loss, but makes it difficult to select random parts of the time series to make sure the model learns the global shape of it. The problem is that it is not possible to directly divide the time series correctly in different window sizes at the same time without intersections. This can be fixed by taking the largest window for the division and later reducing the batch window size, getting the first or latest values, so that both fine-tune modes can be fused.

4.2.4 Statistical loss improvement analysis

To select the values for the main parameters of the fine-tune of the Moment models, a previous analysis using the Kohl's time series is done. For each version of the MOMENT embedding

model (small, base, and large), the validation percentage is fixed to 0.3 and four parameters are analyzed.

The first parameter is the number of epochs for the training. The model is trained until 20 epochs, saving the best-performing model as the final model. Once the training is complete, the best epoch is fixed as the one where the optimal model was saved. This approach allows us to reduce the execution time in the visualization by selecting the most suitable number of epochs.

The second parameter is the dataset percentage, which is tested for $\{0.15, 0.2, 0.25, 0.3\}$.

The third parameter is the number of window lengths. The first window length is fixed to 17 as in the previous MTSAE experiment at Rodriguez-Fernandez et al. [7]. The remaining window lengths are selected using the Fourier Transform result, which identifies the most relevant frequencies, as window sizes until the total number of sizes desired is reached (through the `find_dominant_window_sizes` function in `aeon.segmentation`) [318]. The experiment considers the following number of window sizes: $\{1, 2, 4, 6, 8, 10\}$.

The combination of these options (a total of 72 cases) leads to a sufficiently complete scenario for a preliminary selection of the fine-tune parameters for the visual analysis based on the loss improvement. However, as executing base and large models is more costly in terms of memory and time, the scenario is reduced, setting the number of epochs to 20, dataset percentages to $0.15, 0.2, 0.3$, masked percentages to $0.25, 0.5, 0.75$, and window lengths to $1, 5$, resulting in 18 different use cases.

This setup allows us to compare MOMENT not only against itself but also against MTSAE. However, this comparison is challenging and not entirely fair, as the research evaluates against a pre-trained MTSAE model (replicating the one used in [7], trained with window sizes from 12 to 17).

The results of the experiments show really similar losses between the different versions of MOMENT, with much more adaptability for MOMENT-small, followed by MOMENT-base, and next, MOMENT-large 4.11. The expectation is that MOMENT gets good interpretable visual embeddings for the same cases as MTSAE, being better as the number of parameters increases or the loss is reduced. The rest of the statistical analysis allows us to select the best versions and see if any of them are preferable to the others.

To ensure the best comparison metric, the original loss function is used, taking into account the masked part within the computation. For each execution, the model was validated before and after training, obtaining $loss_{first}$ from the previous evaluation and $loss_{final}$ from the post-evaluation (using the best version of the model). Thus, the loss improvement is defined as $\frac{(loss_{first} - loss_{final}) \cdot 100}{loss_{first}}$. The greater the improvement, the larger the reduction in loss. Therefore, this improvement will be used as the optimization objective in analyzing the best combination of parameters.

The experiment achieves improvements up to 22% in the small model, up to 10% in the base model, and up to 4% in the large model (see Fig. 4.12). This reduction is expected since each larger model contains significantly more parameters than the previous one, allowing it to fit the original time series more effectively. Consequently, achieving further improvement becomes increasingly difficult.

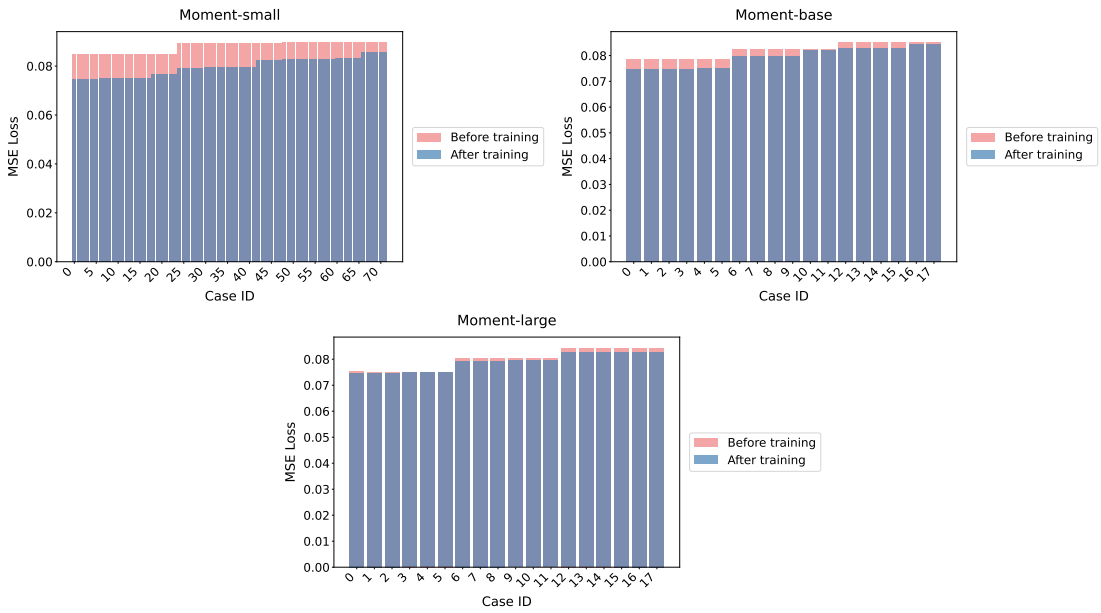

Figure 4.11: MSE losses for all the cases in the experimentation. The red line is used to show the MSELoss for the original model used for *Kohls*. Each bar represent the MSELoss before (red) and after (blue) the fine-tuning.

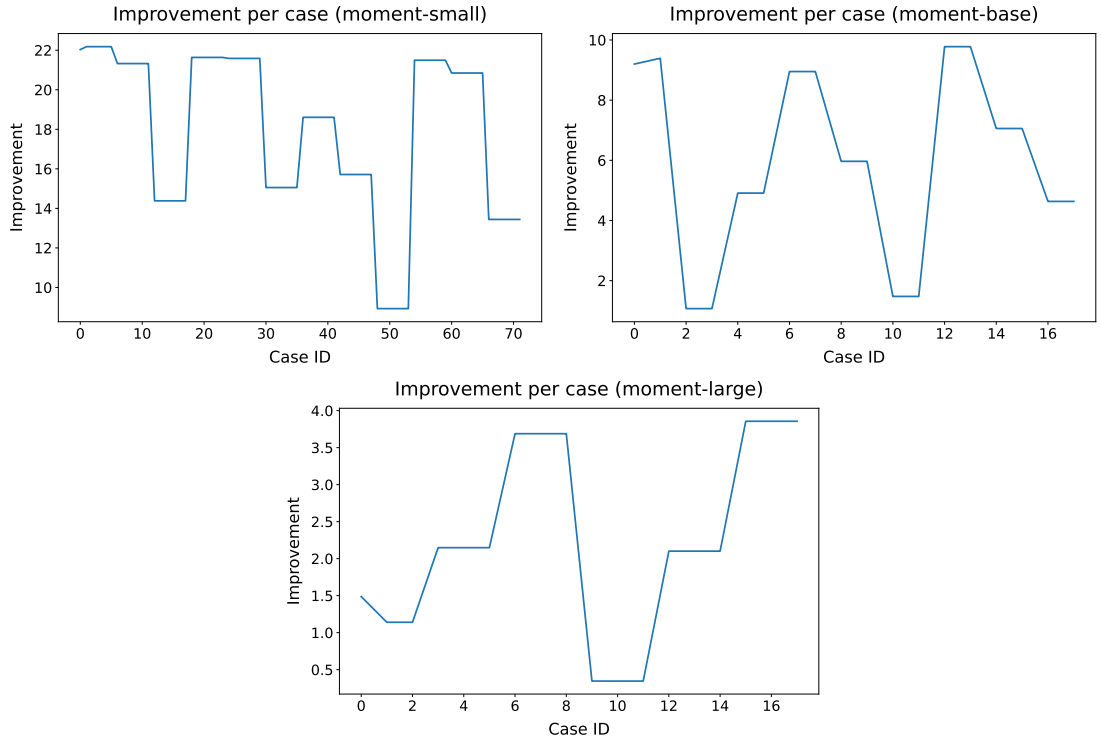

Figure 4.12: Loss improvement for the three version of MOMENT across the experimentation.

To check the parameters that are more related to the loss improvement, the correlations are checked. First, a linear correlation matrix is displayed to check the relationship between the analyzed values: $time$, $best\_epoch$, $dataset\_percent$, $masked\_percent$, $n\_windows$ and $improvemnt$.

The expectation was to have a high positive correlation between the percentage of the dataset and the improvement, as well as the masked dataset and the improvement. However, the only constant correlation found is the time with the dataset percent (see Fig. 4.13). Thus, the researc continues to apply other techniques to check non-linear influences.

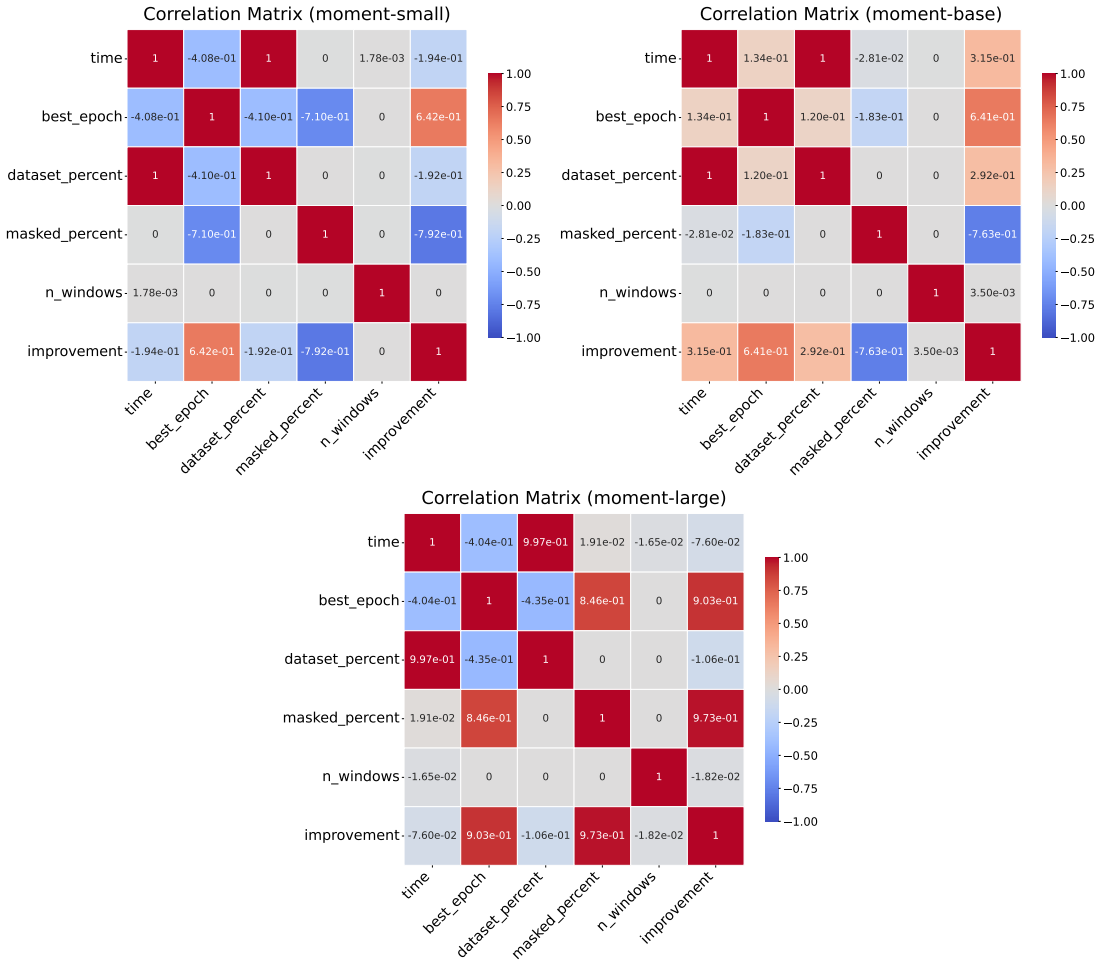

Figure 4.13: Experimentation parametres correlation matrix for the three versions of MOMENT.

As there is no linear correlation, DL-based algorithms are applied to detect the most relevant features: `sklearn.feature_selection`'s `SelectKBest` and `f_regression` functions and `sklearn.ensemble`'s `RandomForestRegressor`. Also, the SHAP impact percent is used as another correlation metric that also gives an idea of the "direction" of the influence - the largest the best or the smaller the best -). The result is constant among the versions of MOMENT: $masked\_percent > best\_epoch >> dataset\_percent >> n\_windows$. The masked_percent has a direct correlation with the loss improvement in the MOMENT-large model while maintaining an inverse correlation in the small and base versions. The best epoch index maintains a direct relation in the three cases (see Tbs. 4.2, 4.3, 4.4).

Based on these results, the best combination of parameters is computed using the two more relevant parameters best improvement rows, obtaining the best combination as a start (see Tb. 4.5).

The next step is to compare MOMENT versus MTSAE. The main challenges arise when con-

Table 4.2: Feature importance scores from SelectKBest, Random Forest and linear correlation for the MOMENT-small model.

Feature	RandomForest %	SelectKBest %	SHAP Impact %	Correlation %
masked_percent	71.475	69.532	21.088	-79.213
best_epoch	21.336	28.893	10.514	64.156904
dataset_percent	7.186776	1.575342	5.575208	-19.172301
n_windows	0.002807	0.000017	0.019719	0.064595

Table 4.3: Feature importance scores from SelectKBest, Random Forest and linear correlation for the MOMENT-base model.

Feature	RandomForest %	SelectKBest %	SHAP Impact %	Correlation %
masked_percent	51.215851	63.808609	18.643660	-76.351055
best_epoch	48.214195	31.917883	16.382083	64.123328
dataset_percent	0.535006	4.272948	0.674598	29.239174
n_windows	0.034947	0.000560	0.095786	0.035000

Table 4.4: Feature importance scores from SelectKBest, Random Forest and linear correlation for the MOMENT-large model.

Feature	RandomForest %	SelectKBest %	SHAP Impact %	Correlation %
masked_percent	56.768665	80.185667	18.106021	97.329411
best_epoch	41.515426	19.762584	12.507370	90.324372
dataset_percent	1.385625	0.050263	1.214560	-10.555755
n_windows	0.330283	0.001485	0.199861	-1.824281

Parameter	Small	Base	Large
masked_percent	25	25	75
best_epoch	17	13	17
n_windows	1	5	1
dataset_percent	15	15	20

Table 4.5: Best parameter values for Small, Base, and Large models based on feature importance analysis.

structing the validation dataset: MTSAE does not support batches with varying window sizes, and the masked percentage was predefined during training. To address the first issue, a window size of 17 is used in validation, since it is the only common size (fixed by hand) in all execution cases. For the second issue, the MSE is computed using the full original input and predictions instead of focusing only on the masked part and execute Moment evaluation for the best case in the same case, using the same functions and percentages as in MTSAE for the generation of masks, ensuring an equivalent experiment.

The comparison of the best versions of MOMENT and MTSAE shows how enlarging the num-

ber of parameters supposes less improvement in terms of MSELoss than the fine-tuning (see Fig. 4.14). Also, MTSAE has a loss more than 70% larger than MOMENT models, regardless of the window length used for the inference (see Fig. 4.15). Thus,the expectation is that MOMENT gets really interpretable visual embeddings for the same cases as MTSAE but not to obtain too different plots between three versions of MOMENT (once fine-tuned). The rest of the statistical analysis allows us to select the best versions and see if any of them is preferable to the others.

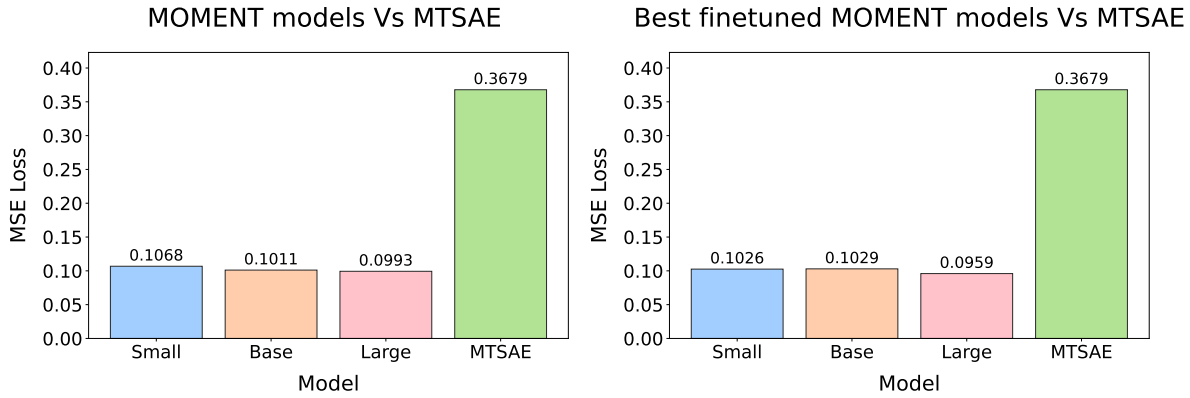

Figure 4.14: Comparison of MOMENT models and the MTSAE model using the mse loss comparing the full prediction to the original batch. By the left, the original version of MOMENT models. By the right, a re-training of the best cases for each MOMENT version.

As the number of epochs may suppose a difference in obtaining the best model in any execution case, its frequencies have been computed to get the largest of the two most repeated values (or the first one if the difference was too big). This analysis resulted in the selection of 17 epochs for MOMENT-small, 13 for MOMENT-base and 20 for MOMENT-large (see Fig.4.16).

To confirm the strange situation that the number of values seems irrelevant, a normalized Parallel Coordinates Plot is computed for the best improvement values. The plots in figure 4.17 show how, indeed, the value remains irrelevant as the value is across all the options instead of focusing on a concrete value. Thus, the best option in Tb. 4.5 is selected for each case.

Based on the analysis conducted, the following parameters were used within the visual experimentation:

Small: 17 epochs, 15% dataset percentage, 25% masked percentage, and 1 window length.

Base: 13 epochs, 25% masked percentage, 15% masked percentage, and 5 window length.

Large: 20 epochs, 0.2 training dataset 75% masked percentage, and 1 window length.

In all cases, the percentage of the validation data set was set at 30%. The study focuses on analyzing the best improvement case, but does not analyze the relationship between the improvement and the execution time, which would also be really interesting in an interactive application.

4.2.5 Visual Analysis of the embedding space of MOMENT

This section completes the visual analysis of the embedding space generated by the six versions (zero-shot/fine-tuned and small/large/base) for the datasets `S1`, `S2`, `S3`, `Kohl's`, and `M-Toy`.

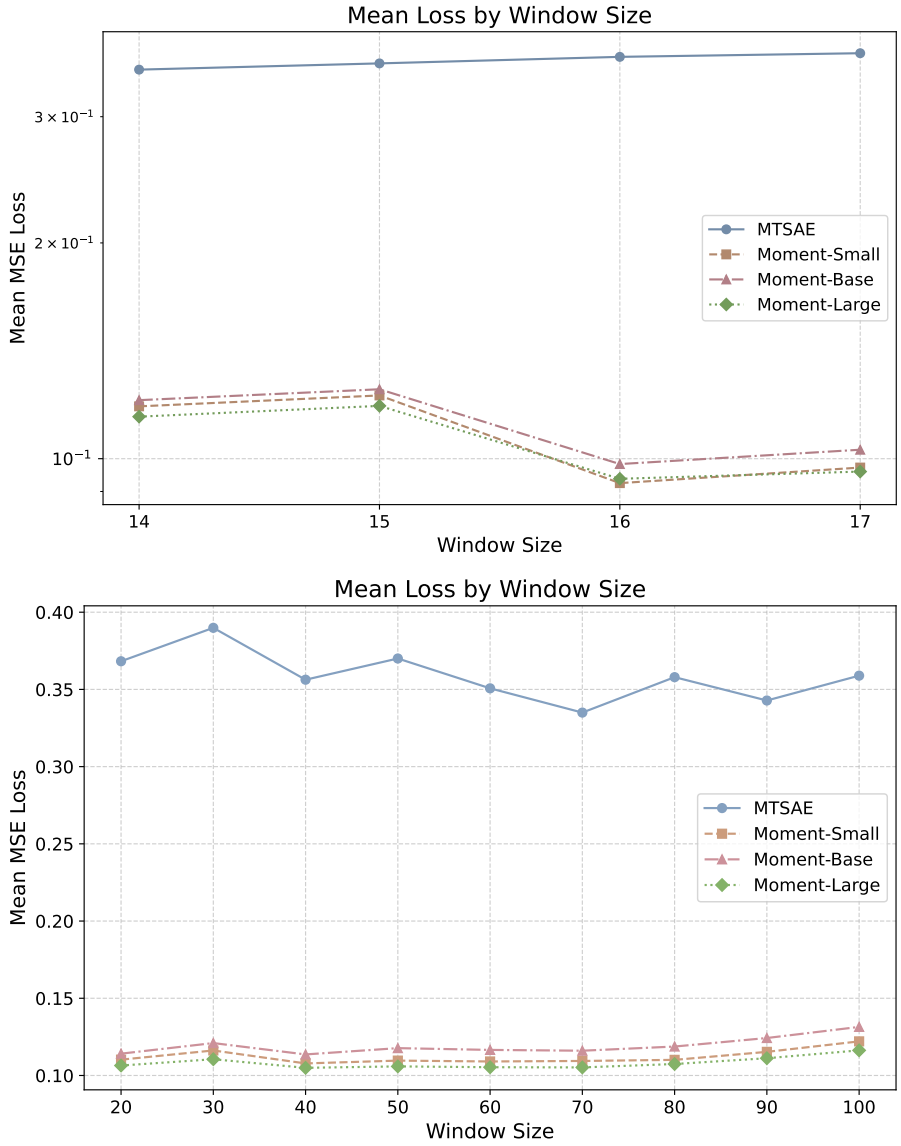

Figure 4.15: Comparison MOMENT models in their best version and the MTSAE model using the mse loss comparing the full prediction to the original batch. At the top, using sizes in the range of those used in MTSAE training. At the bottom, usin sizes in the range between 20 and 100 in steps of 10.

Analysis of S1 using the fine-tuned version of MOMENT-small

Figure 4.18 shows how Moment-small fine-tuning does not represent a visual difference with the zero-shot plot, resulting in the same four clusters shaped as a tree and (somehow) with the only segmentation of the segment $df2$. This is strange as Time Series Foundation models are supposed to get a really good adaptation to the new time series when fine-tuned. The loss improvement evidences this fact. However, the visual plot of the embedding space does not seem to capture this enhancement. Thus, related to **RQ2.1**, this data set does not appear to get a more visually interpretable embedding space when the loss decreases.

To better answer this question, the analysis is performed for the different tasks in DeepVATS.

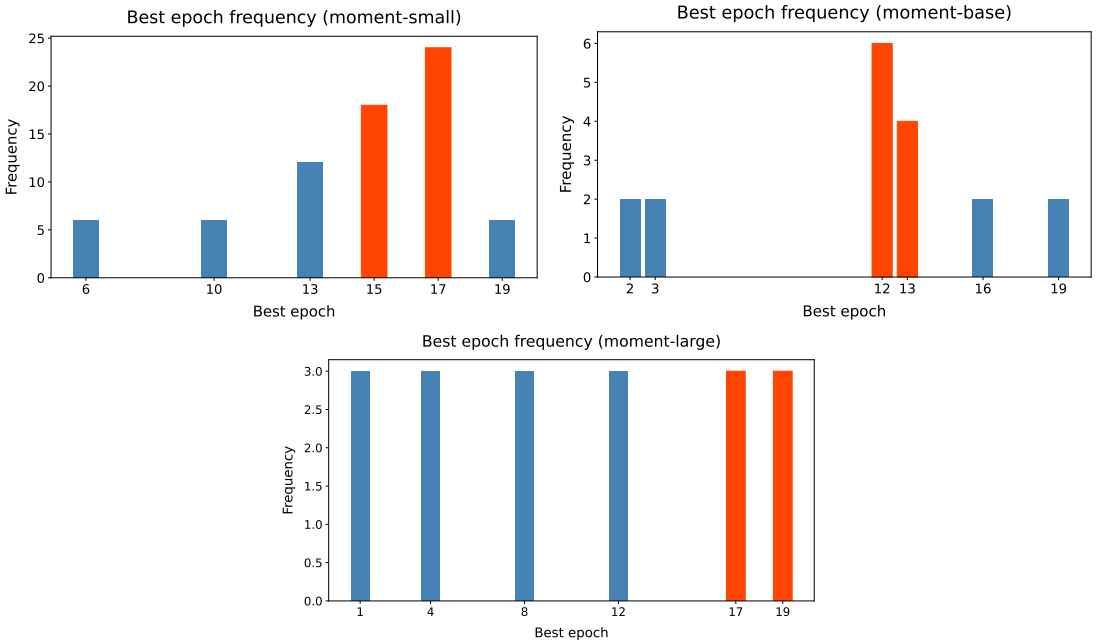

Figure 4.16: From top to botton, the linear correlation matrices for the small, base and large versions of Moment-embedding model.

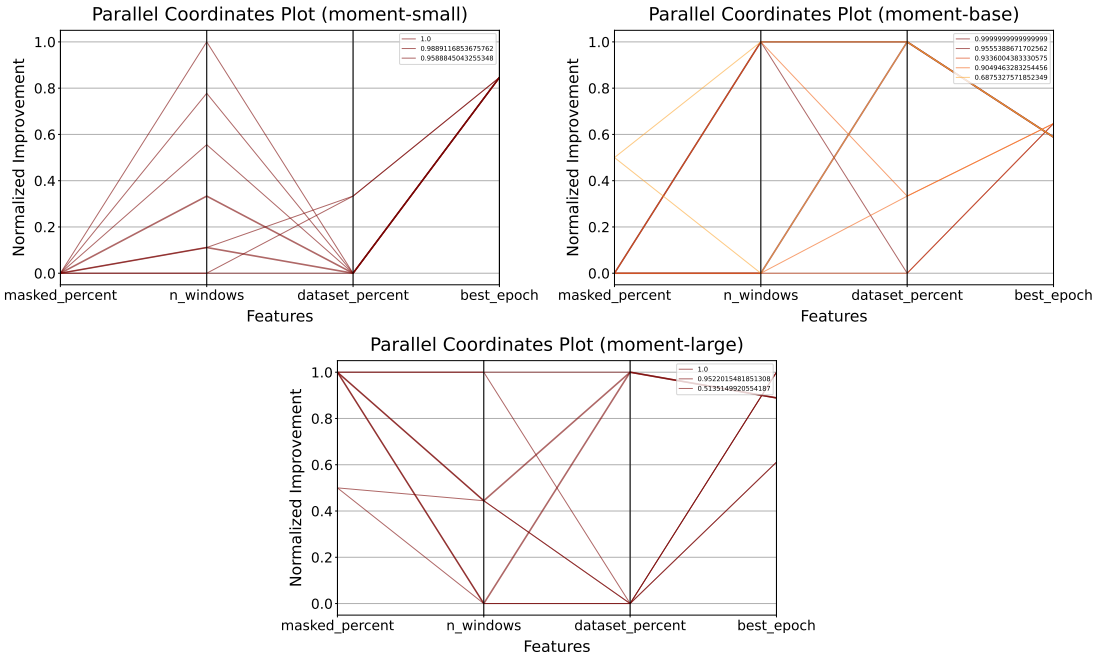

Figure 4.17: From top to botton, the best epoch frequencies matrices for the small, base and large versions of Moment-embedding model.

For segmentation, `S1` (with the rest of the model versions). For anomaly detection, `S2` and `M-Toy`. For trend detection, `S3`.

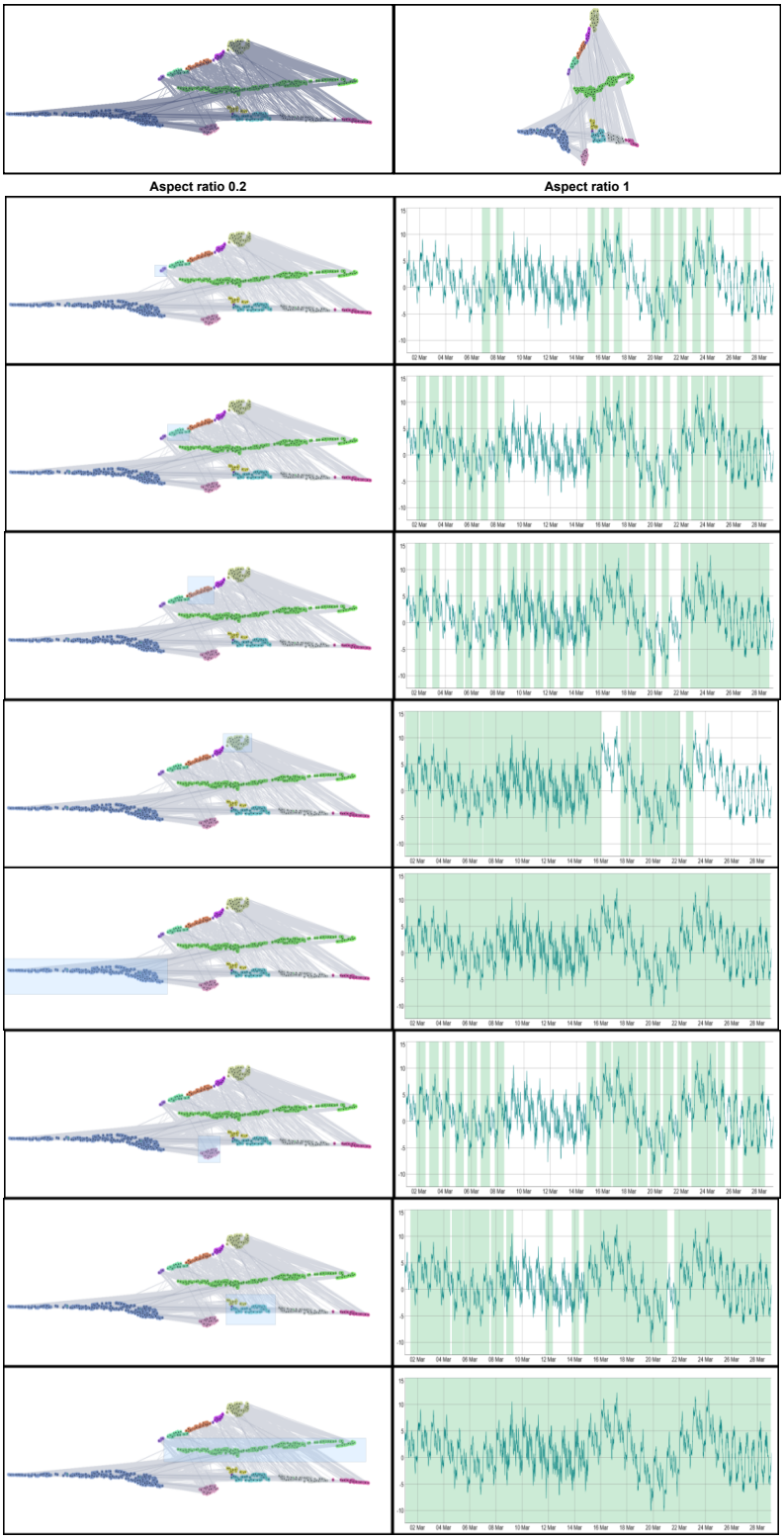

Figure 4.18: Analysis of the embeddings projection plot of the fine-tuned version of MOMENT-small for S1.

Analysis of S1 using both versions of MOMENT-base

Figure 4.19 show that the plot of the base and the fine-tuned model are practically the same, showing no visual enhancement within the projection plot. The embedding space is divided into three clusters.

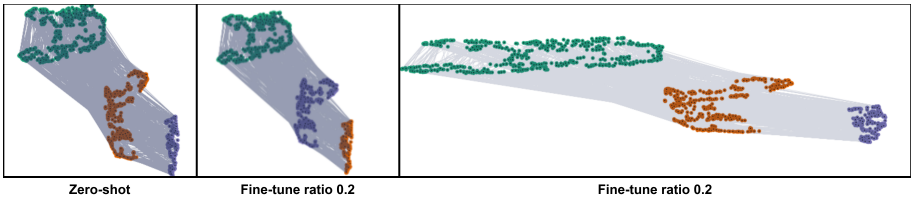

Figure 4.19: Embeddings proyections plot of the zero-shot and fine-tuned versions of the MOMENT-base model for S1.

The first one seems to detect small patterns, not showing a clear classification of the time series' segments (see Fig.A.11). The second cluster makes a really clear segmentation between $df2$ and the rest of the time series and starts to detect $df4$. However, it is not still as clear as it should be, and the split is not done within the 4 segments, so maybe the model and maybe the embedding projection are not capturing the full information needed (see Fig.A.12). However, the third cluster starts to detect four segments really well, but mixed with other patterns (see Fig.A.13).

At this point, it seems that the base version of the model is capturing much better the segmentation information of S1.

Analysis of S1 using both versions of MOMENT-large

Figure 4.20) show the plots of the embeddings for S1 using the large model in both the fine-tuned and zero-shot configurations exhibit a high degree of similarity.

Therefore, the analysis will focus primarily on the fine-tuned model, as it is expected to provide a more complete and interpretable representation.

The embedding space initially consists of three main clusters, resembling those observed in the base model embeddings (see Figs. 4.20, 4.19). However, these clusters appear less distinct and are positioned closer to each other.

Table 4.6 summarizes the analysis for S1. The use of larger versions of Moment seems to be a good idea if the computation resources are good enough as the time series are better matched. However, the fine-tuning has not shown great differences with the zero-shot version of the models. In a future analysis for segmentation, some preprocessing should be done within the training dataset, maybe reducing the noise through Simple Moving Average (SMA). Also, other dimensionality reduction methods should be tried to check if they end up with more time-ordered clusters. Related to the limited enhancement of the fine-tune for the visual analysis, it may be neccesary to check if there is a better distance for the segmentation when optimizing the loss. Also, fine-tune the model with much larger window lengths and then use it to model the original length could yield to long-term relationship. Training with larger SMA orders should also help in analyzing the global structure of the TSs.

Table 4.6: Advantages and disadvantages of using Moment-Small, Moment-Base, and Moment-Large in zero-shot and fine-tuned configurations for S1 dataset (Segmentation).

Model	Training Type	Advantages	Disadvantages
MOMENT-Small	Zero-shot	✓ Captures general patterns in the data. ✓ Requires no additional training, reducing computational cost.	✗ Produces poorly defined embeddings ("blurry" clusters). ✗ Struggles to differentiate time series segments effectively. However, it starts to detect $df1, df2$ and $df3$.
	Fine-tuned	✓ Moderate loss reduction (~22%), improving embedding quality. ✓ Some improvement in pattern detection.	✗ Visual embedding projection shows minimal improvement over zero-shot. ✗ Cluster structure remains unclear despite fine-tuning.
MOMENT-Base	Zero-shot	✓ Clearly detects $df2$ and it is more near to detect $df1$ and $df2$ better than Small. ✓ Captures some patterns of the time series.	✗ Still lacks precise segmentation of time series into clusters. Losses the precision with $df1$ and $df2$.
	Fine-tuned	✓ Detects patterns.	✗ Fine-tuning provides only a marginal improvement. ✗ Cluster structure still lacks well-defined segmentation.
MOMENT-Large	Zero-shot	✓ Detects patterns.	✗ May be influenced by the time series' noise. ✗ Cluster are not very sepparated. ✗ Segments are not clear.
	Fine-tuned	No special advantages.	✗ fine-tuned embeddings are still highly similar to zero-shot, not getting any different result within the analysis.