technische universität
dortmund

Bachelor Thesis

Autoencoder Embeddings for Adaptive Model Selection and Drift Detection

Ilias Attary

August 29, 2025

Reviewer:
Dr. rer. nat. Amal Saadallah
M. Sc. Matthias Jakobs

TU Dortmund University
Department of Computer Science
Lamarr Institute for Machine Learning and
Artificial Intelligence
https://lamarr-institute.org/

Contents

1. Introduction

Time series forecasting plays a pivotal role across various industries, enabling informed decision-making, risk mitigation, and resource optimization [35]. To cope with the complex nature of real-world forecasting, researchers have developed many sophisticated models ranging from statistical approaches to neural networks and deep learning architectures that aim to improve both accuracy and adaptability [28, 33].

1.1. Motivation

While significant progress has been made, no single model has been proven universally effective across all applications or across all segments of a time series [28]. This is because different models are inherently biased toward different types of structures. Some are better suited for capturing seasonal cycles, while others excel at modeling long-term dependencies or nonlinear dynamics within the data. Accordingly, each forecasting model tends to perform better on specific patterns of a time series than others. This makes time series forecasting a very complex task, especially when real-world data exhibit multiple, overlapping structures. This challenge is further amplified by concept drift, in which the statistical properties of the time series evolve over time [20]. External factors such as shifts in consumer habits or technological advancements disrupt established patterns. Eventually, the gap between the model's assumptions and the properties of the actual data grows. As a result, models that were previously optimal may become less effective or even obsolete.

To address these challenges, dynamic model selection frameworks have been introduced [11]. These approaches employ a pool of candidate models and dynamically select the most appropriate model for the current data characteristics. By identifying where and when each model performs best, the system can make informed, context-aware selections. This idea is the core of region of competence (RoC) based selection, where model choice is driven by contextual similarity rather than fixed assumptions [39]. Defining similarity between time series segments, however, is a non-trivial task. Neural networks, particularly autoencoders, offer valuable opportunities for improving dynamic model selection systems. Autoencoders learn compact latent representations of data, while still preserving the most essential information [25]. Rather than comparing time series segments directly in the time domain, comparisons in latent space may result in more reliable similarity measures and, consequently, more effective model selection. Moreover, reconstruction errors produced by the autoencoder can serve as an indicator of concept drift, enabling drift-aware adaptation [29].

In this thesis, we explore a framework that combines convolutional autoencoder representations with region of competence-based adaptive model selection for time series forecasting. Our approach dynamically selects the most suitable model at each forecasting step based on latent similarity to previously observed contexts and integrates reconstruction-based drift detection for online adaptability.

1.2. Thesis Structure

The rest of this thesis is organized as follows. In chapter 2, we provide the necessary background relevant to our work. This includes an overview of time series analysis and model selection, followed by an introduction to autoencoders and concept drift. chapter 3 presents the forecasting models employed in this thesis. chapter 4 introduces our dynamic model selection framework for time series forecasting: Autoencoder Embeddings for Adaptive Model Selection and Drift Detection (AE-AMS). This chapter covers data preprocessing, the model selection strategy, and the drift detection mechanism. In chapter 5, we conduct an experimental evaluation on the proposed framework, including the evaluation setup specification, the comparative baselines, and the results of these experiments. chapter 6 concludes the thesis and provides future research directions.

2. Background & Related Work

Time series analysis refers to the set of methods and techniques used to extract meaningful patterns from time series data in order to support tasks such as forecasting, anomaly detection, and statistical modeling [28]. Time series data can be described from either a parametric or non-parametric perspective [10]. Parametric methods assume that the data follow a structure, called a stochastic process, which can be described using a small number of parameters [6]. Non-parametric approaches estimate the variability and dependencies directly from the data without assuming any particular structure [43]. In the following sections, we primarily introduce time series from a parametric perspective, as this provides a strong foundation for essential concepts of time series modeling.

2.1. Time Series Definitions and Overview

A time series is defined as a sequence of observations recorded at successive points in time, representing the evolution of a variable or set of variables over the temporal domain [6, 23]. Mathematically, it is represented as a collection of random variables $\{X_t\}$, indexed by time $t \in T$, where X_t denotes the random variable representing the observed value at time t, and T is the time index set. From a probabilistic perspective, we say the observed series is generated by an underlying stochastic process, specifically one particular realization out of all the possible paths that the process could generate. This process defines the probabilities that describe how the variable changes over time [43].

Definition: Stochastic Process [13]

A stochastic process is a collection of random variables $\{X_t : t \in T\}$ defined on a common probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where T is the index set (often interpreted as time). For each $t \in T$, the random variable X_t assigns to every outcome $\omega \in \Omega$ a value $X_t(\omega)$, typically $\mathbb{R}$ or $\mathbb{R}^n$. These values represent the state or measurement of the process at time t when the outcome ω occurs. Each X_t has an associated probability distribution, which can be described using its moments when they exist:

- Mean: $\mathbb{E}[X_t]$,
- Variance: $\mathrm{Var}(X_t) = \mathbb{E}[(X_t - \mathbb{E}[X_t])^2]$,
- Covariance: $\mathrm{Cov}(X_s, X_t) = \mathbb{E}[(X_s - \mathbb{E}[X_s])(X_t - \mathbb{E}[X_t])]$.

The dimension of X_t determines the type of the time series [23]. A p-vector-valued X_t produces a multivariate time series, meaning that at each time index t, p variables are observed. These variables can be cross-dependent, meaning that they are related to each other, either simultaneously or at different time steps. Each variable may also be dependent on its own past values. This type of dependence is called autocorrelation. Some variables might be autocorrelated but cross-independent, or vice versa. A

time series with no temporal dependencies is referred to as a white noise process [10]. A univariate time series is a special case of a multivariate time series with $p = 1$. In other words, a univariate time series consists of only scalar-valued X_t.

The time index set T specifies the points in time at which the process is defined, which may be either discrete (e.g., $T = \{1, 2, 3, \ldots\}$) or continuous (e.g., $T = [0, \infty)$). This set defines the temporal domain of the process, independent of how or when we actually observe it. Sampling a time series involves selecting a subset $T_{\text{obs}} \subseteq T$, where X_t is recorded. If T_{obs} has constant intervals, the sampling is termed regular sampling, whereas variable spacing between points denotes irregular sampling. The sampling frequency determines the granularity of the observed values and defines the limit of the patterns that can be analyzed. If the frequency is too low, we lose the ability to detect patterns that occur on a timescale smaller than the sampling interval. Regular sampling is often required for many statistical methods, so irregularly sampled data often must be interpolated (i.e., estimating missing values) or resampled to create evenly spaced series before analysis [28].

2.1.1. Components of Time Series

A time series is often decomposed into interpretable components to better understand its structure. The trend T_t describes the long-term direction of the series. Seasonality S_t specifies the regular, repeating pattern on fixed periods (e.g., daily, weekly, yearly). Cyclical patterns C_t are long-term aperiodic fluctuations characterized by varying duration and amplitude. Lastly, the random component R_t represents the irregular noise that cannot be explained by other components [28].

Decomposition models explain how the components combine to form the observed time series. Common decomposition models include the additive model and the multiplicative model. The additive model is most effective when variation remains approximately constant over time:

$$X_t = T_t + S_t + C_t + R_t$$

In contrast, the multiplicative model is more useful when the trend or seasonal variation scales with the level of the series:

$$X_t = T_t \cdot S_t \cdot C_t \cdot R_t$$

2.1.2. Stationarity

A stochastic process $\{X_t\}$ is strictly stationary if the joint cumulative distribution function of any set of k random variables generated by the process is the same over time [10]. This implies that its statistical properties, in particular the probability distribution, are constant over time. A stochastic process $\{X_t\}$ is weakly stationary if it has the following statistical properties:

- Constant mean $\mathbb{E}[X_t] = \mu$,
- Constant variance $\mathrm{Var}(X_t) = \sigma^2$,
- Autocovariance $\mathrm{Cov}(X_t, X_{t+h}) = \gamma(h)$ depends only on the lag h.

Many statistical models assume at least weak stationarity [10]. This presents challenges in practice, as real-world time series are often non-stationary and require transformations (e.g., detrending, differencing, or deseasonalizing) before analysis.

2.1.3. Time Series Forecasting

Time series forecasting refers to the task of predicting future values of a series, typically by fitting a statistical or machine learning model, based on its historical observations [28, 23]. Let $\{y_t\}_{t=1}^{N}$ be a real-valued time series, where y_t is the observed value at time t. The goal is to estimate a function f that maps past observations to future values:

$$\hat{y}_{t+h} = f(y_1, y_2, \ldots, y_t)$$

where $\hat{y}_{t+h}$ is the forecast value at time $t + h$, and h is the forecasting horizon. The function f can be derived from statistical, machine learning, or deep learning models designed to capture the temporal dependencies, trends, seasonality, and noise in the observed data.

These forecasts are evaluated using an evaluation metric, which quantifies the prediction accuracy. Each metric is sensitive to different aspects of model error [35]. Some commonly used evaluation metrics include [28, 35]:

- **Mean Squared Error (MSE)**

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

  The mean squared error measures the mean of squared differences between actual and predicted values. It is more sensitive to outliers due to the squaring, making it more appropriate when penalizing large errors is critical.

- **Root Mean Squared Error (RMSE)**

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2}$$

  The root mean squared error is the square root of MSE. This transformation brings the error back to the original unit of the data, while maintaining the properties of the MSE.

- **Mean Absolute Error (MAE)**

$$\text{MAE} = \frac{1}{N}\sum_{n=1}^{N}|y_n - \hat{y}_n|$$

  The mean absolute error measures the mean absolute difference between predicted and actual values. It equally weights all errors, making it more robust to outliers than MSE and RMSE.

- **Mean Absolute Percentage Error (MAPE)**

$$\text{MAPE} = \frac{100\%}{N}\sum_{n=1}^{N}\left|\frac{y_n - \hat{y}_n}{y_n}\right|$$

  The mean absolute percentage error expresses error as a percentage of the observed values. It's useful for comparing performance across different scales. However, MAPE can be unstable when these values are zero or close to zero.

2.2. Model Selection

Model selection is the process of using statistical or empirical evaluation to choose from a set of candidate models the one that provides the best predictive performance on unseen data [46]. Model selection strategies can be broadly classified into two categories. Static model selection is the simplest and most common approach. A key characteristic of static model selection is that the model is selected once and used for all future forecasts [46]. Since the model remains fixed, it is simpler to implement. However, this also means it's more prone to losing accuracy in dynamic environments. In practice, each model is first trained on a designated training set, then all models are evaluated on a separate validation set. Cross-validation is a widely used technique for model evaluation due to its ability to provide stable and fair comparisons between multiple models or configurations [2]. The model with the best performance is then selected for the forecasting task.

A more sophisticated technique is dynamic model selection. At each forecasting step, the most appropriate forecasting model is reevaluated and used for this specific step [40, 11]. The key difference between dynamic and static model selection is that static selection focuses on estimating each model's overall performance across the full data distribution, whereas dynamic selection tries to find the best-performing model at each forecasting point, making the approach more adaptable and addressing the limitations of static selection. Identifying a suitable model from the pool relies on constructing a representational understanding of each candidate [37]. A robust mechanism to achieve that is by defining Regions of Competence (RoCs): a set of examples where individual models have shown superior forecasting performance.

2.2.1. Regions of Competence

The concept of a region of competence, as used in machine learning and predictive modeling, refers to a subset of the input space where a given model or method is known to perform reliably [40]. This concept is at the core of dynamic model selection and dynamic ensemble selection methods, as it facilitates modeling each model's relative strengths across different regions of the input space.

In region of competence-based selection, models are evaluated on a validation set using local regions. These regions can be obtained either by partitioning the set (e.g., via clustering) or by dynamically identifying neighborhoods (e.g., using the k-nearest neighbors algorithm). The competence of each candidate model is then assessed by assigning each region to the highest-performing model as part of its RoC. A single model may be associated with several regions, or with none at all. When a new input is presented, it is compared against each model's RoC using a predefined similarity metric, and the model whose region is closest to the input is selected. In the context of time series forecasting, the inputs are segments obtained by sliding a fixed-length window across the series, while the new input corresponds to the most recent window of observations.

Saadallah builds on the concept of region of competence to improve the adaptability and transparency of model selection in time series forecasting. In [39], Saadallah introduces OMS-ROC, an online framework that updates each model's RoCs as new windows are observed. By monitoring changes in these regions, the system can detect evolving patterns in the time series and trigger a recalculation of the RoCs, maintaining adaptability and ensuring suitability for real-world tasks. The framework also uses these RoCs to provide clear, model-agnostic explanations for why a specific model is chosen for each forecast. By combining adaptability with interpretability, OMS-ROC achieves state-of-the-art performance on 100

real-world datasets.

2.2.2. Similarity Metrics

The similarity metrics used in RoC-based selection quantify how close the input is to the examples in each model's RoC. Let $x, y \in \mathbb{R}^d$ denote two vectors and let $\|\cdot\|$ denote the Euclidean norm defined as $\|x\| = \sqrt{\sum_{i=1}^{d} x_i^2}$. In this thesis, we employ the following metrics:

- **Euclidean Distance**

$$d_{\text{eucl}}(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

  Euclidean distance assumes isotropic and homoscedastic geometry, meaning that all dimensions contribute equally and independently to the distance.

- **Cosine Distance**

$$d_{\cos}(x, y) = 1 - \frac{x \cdot y}{\|x\| \, \|y\|}$$

  Cosine distance measures the distance as the angle between two vectors. It takes values in $[0, 2]$, with smaller values indicating greater similarity. It is especially useful when the direction of vectors is more important than their magnitude.

- **Dynamic Time Warping (DTW)**

  Dynamic time warping is a distance metric specifically designed to compare temporal sequences that may be misaligned in time [4]. Let $u = (u_1, \ldots, u_n)$, $v = (v_1, \ldots, v_m)$ and $d(u_i, v_j)$ denote the distance between u_i and v_j. DTW is calculated as

$$d_{\text{dtw}}(i, j) = d(u_i, v_j) + \min \begin{cases} d_{\text{dtw}}(i-1, j-1), \\ d_{\text{dtw}}(i-1, j), \\ d_{\text{dtw}}(i, j-1) \end{cases}$$

  with $d_{\text{dtw}}(0, 0) = 0$, $d_{\text{dtw}}(i, 0) = \infty$ and $d_{\text{dtw}}(0, j) = \infty$. The final distance is $d_{\text{dtw}}(n, m)$, representing the minimal cumulative alignment cost between u and v.

2.3. Autoencoders & Convolutional Autoencoders

Dynamic model selection based on regions of competence requires each new input to be compared against the RoC associated with every candidate model, typically through a distance metric [39]. A basic approach involves computing the distance between the center of each region in a model's RoC and the incoming input, with smaller distances indicating stronger similarity and thus potentially higher model performance. However, direct comparison in the raw input space can be suboptimal. Two time series sequences can have identical underlying dynamics but look different in raw form due to noise, local misalignment, scaling or other distortions [1]. Neural networks enable the learning of transformations that map raw sequences into latent representations, where only the essential structural information is preserved and

irrelevant variations are minimized [22]. Once in an appropriately learned latent space, even simple distance metrics can serve as effective tools for comparing inputs and identifying the most suitable forecaster [25]. This section provides an overview of neural networks and introduces an architecture particularly well suited for the objectives of our experiment.

2.3.1. Neural Networks

Neural networks are machine learning models inspired by the structure of biological neural systems [22]. They consist of interconnected processing units, called "neurons", arranged into layers. Each neuron computes a weighted sum of its inputs and adds a bias term. A nonlinear transformation is applied to the output, using an activation function. By organizing these computations into multiple layers, neural networks are designed to approximate functions between inputs and outputs automatically through training, adjusting their parameters to minimize a predefined objective function [22]. They are especially useful for problems that are difficult to describe explicitly.

Activation Function

Activation functions are essential to any neural network architecture, since they introduce nonlinearity into the model. Real-world problems are rarely linear and these functions allow the network to approximate arbitrarily complex decision boundaries. Regardless of the number of layers, a neural network without activation functions is just a single-layer linear transformation [22]. Common choices include ReLU, Sigmoid, and Tanh, each with distinct characteristics:

- **ReLU**

  $$f(z) = \max(0, z)$$

  ReLU is a simple and efficient activation function. It outputs the input back if it's positive, otherwise zero. It introduces nonlinearity without significant computational cost. That said, neurons can become permanently inactive if inputs are negative too often.

- **Sigmoid**

  $$f(z) = \frac{1}{1 + e^{-z}}$$

  The sigmoid function maps inputs to the interval $(0, 1)$, which makes it suitable for modeling probability. However, as the input z becomes very large (or very small), changes in z produce little to no change in the output.

- **Tanh**

  $$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

  The tanh function outputs in the interval $(-1, 1)$. As a consequence, the outputs are centered around zero, improving optimization during training. Like the sigmoid function, it suffers from saturation at extreme input values.

Learning Process

A neural network learns by iteratively refining its parameters θ (weights, biases) to improve its performance on a given task. This performance is quantified through an objective function $\mathcal{J}(\theta)$, which represents the core objective of the problem [5]. Depending on the task, $\mathcal{J}$ can take different forms. In supervised learning, where the outputs are known in advance, the objective function measures the difference between the model's predicted output and the true target values. In unsupervised learning, it may instead evaluate how effectively the model captures the underlying structure of the data. To minimize the objective function, an optimization algorithm, such as gradient descent, iteratively updates θ in the direction that reduces $\mathcal{J}$ [38]. Each iteration consists of the following steps.

The forward pass is the first step of the learning process. The network processes input data to produce an output, that is then evaluated using the objective function $\mathcal{J}$. In the backward pass, we compute the gradient of the objective function with respect to the model parameters $\nabla_\theta \mathcal{J}$. This is done layer-by-layer starting from the last layer using the chain rule. Finally, we update the parameters in the opposite direction of the gradient using a learning rate η that controls the magnitude of each adjustment: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{J}$. As training progresses over multiple iterations, the parameters gradually move toward values that minimize the objective function [22].

2.3.2. Autoencoders

Autoencoders are a special type of neural networks designed to learn meaningful, low-dimensional latent representations of the input data through unsupervised learning. In its simplest form, an autoencoder consists of the encoder, which compresses the input into a lower-dimensional space, and the decoder, which expands the latent vector back to its original dimension [25].

Formally, an autoencoder learns two functions: given an input $x \in \mathbb{R}^d$, the encoder is a function $\phi_\theta : \mathbb{R}^d \to \mathbb{R}^m$, with parameters θ, that maps the input space $\mathbb{R}^d$ to a lower-dimensional latent space $\mathbb{R}^m$, where $m \ll d$. The decoder, on the other hand, is a function $\psi_\eta : \mathbb{R}^m \to \mathbb{R}^d$, with parameters η, which attempts to reconstruct the original input from its latent representation, i.e., $\hat{x} = \psi_\eta(\phi_\theta(x))$. Training consists of optimizing θ and η to minimize the objective function, formally $\min_{\theta,\eta} \mathcal{L}(x, \hat{x}) = \mathcal{L}(x, \psi_\eta(\phi_\theta(x)))$. This type of objective function is called reconstruction loss and it measures the difference between the original and the reconstructed input, typically as $\|x - \psi_\eta(\phi_\theta(x))\|^2$.

Standard autoencoders treat the input as a flat vector, ignoring its temporal structure. This makes "vanilla" autoencoders suboptimal for time series data. Convolutional autoencoders address these issues by implementing convolutional and pooling layers constructed to preserve local structures, making them more effective for data types with inherent spatial or temporal structure [36].

2.3.3. Convolutional Autoencoders

Formally, a convolutional autoencoder (CAE) redefines the encoder ϕ_θ and decoder ψ_η as a sequence of convolutional layers and pooling layers [36]. Each convolutional layer performs a set of convolution operations, followed by a nonlinear activation function, to learn filters. These filters, also called kernels, are small tensors (multidimensional arrays) that detect local patterns in the input data [32]. The result of each convolution is a feature map that indicates where and how strongly the input matches the filter. Pooling layers are responsible for downsampling the feature maps to expand the receptive field for the following convolutional layers. The decoder then reconstructs the original input by reversing these steps.

This involves using upsampling operations to restore the feature maps to their original dimensions, and then refining them by applying the learned kernels. These building blocks make CAEs well-suited for modeling local patterns of time series sequences, producing informative latent representations [36].

Convolution Operation

Time series data are typically represented as one-dimensional sequences, where the number of channels corresponds to the number of variables observed [32]. For univariate time series, this corresponds to a single channel. Convolution operations involve sliding a small kernel over the input data, extracting features that represent local structures. For an input $\mathrm{X} \in \mathbb{R}^{C_{\text{in}} \times L}$ with C_{in} channels and a kernel $\mathrm{W} \in \mathbb{R}^{C_{\text{in}} \times k}$ of size k, the 1D convolution at position i is given by:

$$(\mathrm{W} * \mathrm{X})(i) = \sum_{c=1}^{C_{\text{in}}} \sum_{m=0}^{k-1} W_{c,m} \, X_{c,\, i+m}.$$

For univariate data ($C_{\text{in}} = 1$), this simplifies to:

$$(\mathrm{w} * \mathrm{x})(i) = \sum_{m=0}^{k-1} w_m \, x_{i+m}.$$

Pooling Layer

Pooling layers are used in convolutional neural networks to reduce the size of the feature maps while retaining the most important information, allowing deeper layers to see global context [32]. This is done by sliding a window over the feature map and computing a statistic, such as the maximum or average, within each window. The result of this computation becomes one element of the downsampled output. Common pooling layers include max pooling and average pooling.

Max pooling selects the maximum value in each pooling window. As a result, it captures the strongest features in each segment, making the resulting output more robust to noise. That said, potentially informative patterns may be lost if they are not the largest values in the window. Average pooling takes the average value in each segment, which preserves information about the overall pooling region, but it can blur strong values.

2.4. Concept Drift

Concept drift refers to changes in the underlying probability distribution of the time series over time [20]. Over a time interval Δt and time step t, concept drift is formally defined as:

$$P_t(X, Y) \neq P_{t+\Delta t}(X, Y)$$

where $P_t(X, Y)$ represents the joint probability distribution of the random variables (X, Y) at time t. Since the forecasting models are trained on $P_t(X, Y)$, this change in distribution causes the performance of the models to degrade [20]. $P_t(X, Y)$ can be described using the product rule as: $P_t(X, Y) = P_t(Y \mid X)\, P_t(X)$. Here, $P_t(Y \mid X)$ denotes the posterior, which quantifies the probability of the target value Y given input X. $P_t(X)$ denotes the marginal distribution over the input variables and $P_t(Y)$ denotes the

prior distribution over output values, independent of the input. As a result, any change in $P_t(X)$, $P_t(Y)$, or $P_t(Y \mid X)$ over time will introduce concept drift. Following Žliobaitė [51], it is typically classified as real or virtual drift depending on which component changes:

- **Real Drift**

  $$P_t(Y \mid X) \neq P_{t+\Delta t}(Y \mid X)$$

  Real concept drift is characterized by a change in the relationship between the inputs and the target variables. This directly impacts model performance, since the assumptions on which the model was trained are no longer valid.

- **Virtual Drift**

  $$P_t(X) \neq P_{t+\Delta t}(X), \quad P_t(Y \mid X) = P_{t+\Delta t}(Y \mid X)$$

  Virtual concept drift is defined by a change in input (or target) distribution, but the posterior stays stable. It may be less impactful compared to real drift, as the underlying decision boundaries remain the same. A change in $P_t(X \mid Y)$ may correspond to virtual drift if $P_t(Y \mid X)$ remains unchanged, or to real drift if it changes $P_t(Y \mid X)$.

In addition to categorizing the source of distributional change, it is also important to consider how drift manifests over time. Sudden drift occurs when the change in the data distribution happens abruptly, often between two consecutive time steps. Gradual drift refers to progressive distributional shifts, where both the old and new concepts coexist for a period. Recurrent drift is characterized by the reappearance of a previously seen concept after some interval of time, typically in cyclical or seasonal patterns. Incremental drift refers to a progressive and continuous change of the data distribution, with small, ongoing changes accumulating over time [47].

2.4.1. Drift Detection

Drift detection is essential to mitigate model degradation in dynamic environments [20]. Multiple drift detection strategies have been proposed to cope with the evolving nature of these non-stationary environments. This section presents several established drift detection methodologies, along with autoencoder-based approaches.

Traditional Drift Detection Strategies

Error-based detection methods identify concept drift by monitoring model prediction errors. In a stationary data distribution, errors are expected to remain stable over time [20]. Therefore, significant changes in the statistical properties of errors may indicate drift. As a concrete example, the Drift Detection Method (DDM) tracks the model's error-rate (i.e., $p_i = \frac{\text{number of errors}}{\text{number of predictions}}$) and its standard deviation s_i. A drift alarm is issued when $p_i + s_i \geq p_{\min} + 3 \cdot s_{\min}$, where $p_{\min}$ and $s_{\min}$ represent the lowest error rate and its corresponding standard deviation observed so far [19]. The primary advantage of error-based detection methods is detecting the changes that actually impact the model's performance, triggering adaptation only when necessary. However, they require true target values for error computation and can be sensitive to noise or outliers.

Distribution-based methods monitor shifts in the statistical distribution of observed time series values [20]. New observed values are stored in a sliding window and compared to a reference window. The

observations in the reference window are assumed to come from the same underlying distribution. These algorithms signal drift when the compared windows show statistically significant differences. Instead of fixing the reference and current windows in advance, similar approaches update summary statistics with each new observation, such as running mean or cumulative deviation (i.e., accumulated change from a baseline mean $\sum_i (x_i - \hat{\mu})$). When these statistics exceed a predefined threshold, a drift is signaled. These thresholds are either dynamically computed or empirically predefined. Hoeffding's Drift Detection Method (HDDM) follows this approach by monitoring the moving average of a target metric (e.g., error rate, feature values) using Hoeffding's inequality [17].

Theorem : Hoeffding's inequality

Hoeffding's inequality states that for an independent and bounded random variable X_t with range R, after N independent observations, the probability that the sample mean $\bar{X}_N$ deviates from the true mean $\mathbb{E}[X_t]$ by more than ξ is at most δ:

$$P\left(\left|\bar{X}_N - \mathbb{E}[X_t]\right| > \xi\right) \leq \delta,$$

where

$$\xi = \sqrt{\frac{R^2 \cdot \ln\left(\frac{1}{\delta}\right)}{2N}}.$$

HDDM keeps track of the running means ($\mu_{\text{ref}},\ \mu_{\text{current}}$) and sample counts ($n_{\text{ref}},\ n_{\text{current}}$), updating them with each new observation. A drift signal is triggered when the mean difference exceeds the adaptive Hoeffding bound. In contrast to error-based methods, distribution-based approaches do not require ground truth. These methods have the ability to detect drift in its early stages, before it impacts the performance of the models [17]. The detection thresholds, on the other hand, must be carefully tuned for different datasets, since they heavily impact sensitivity.

Autoencoder-based Detection Strategies

Many studies have explored using autoencoders as drift detectors, specifically by monitoring reconstruction errors [50]. Similar to error-based detection methods, the core motivation for autoencoder-based detection strategies is that in a stable environment, the reconstruction error for new observations should remain statistically stationary over time. Significant deviations in the statistical properties of these errors might indicate a shift in the underlying data distribution, suggesting potential concept drift.

Maciej Jaworski et al. [29] proposed an autoencoder-based concept drift detection method for streaming data. Their approach begins by training an autoencoder on an initial segment of the data stream assumed to be stable. The trained autoencoder then reconstructs each new incoming data instance, while the reconstruction error and the binary cross-entropy are continuously monitored. A drift alarm is triggered when the current error significantly exceeds a dynamic threshold, computed from a sliding window of recent error values. This approach was tested on synthetic and real-world datasets, showing effectiveness at detecting both sudden and gradual drift.

2.4.2. Drift Adaptation

Adaptation strategies aim to maintain model performance under concept drift by modifying or retraining the model in response to detected changes in the data distribution [20].

Passive adaptation strategies continuously update the model over time [20]. These strategies do not require any explicit drift detection mechanism, making them simpler to implement. This continuous adaptation is also well-suited for gradual and incremental drift. The trade-off is the potential wasted computation, when no drift is present. Moreover, they may suffer from catastrophic forgetting, where previously learned information is discarded, even if it is still relevant. These strategies can be implemented using different mechanisms. A sliding window approach, for example, trains the model from scratch on the most recent n instances. Another approach is incremental learning, where the model is updated with each new instance or batch. Finally, weighted training gives higher weight to recent instances, allowing the model to prioritize more recent patterns in the data.

Active adaptation strategies, in contrast, update the model only when a drift detector signals a shift in the data distribution [20]. Consequently, unnecessary updates are avoided when the distribution is stable. Active strategies also enable, in contrast to passive adaptation, aggressive reactions to accommodate large shifts in the distribution. Since drift detectors are required for this approach, their inherent limitations must be taken into account, and careful tuning of detector parameters is critical to ensure adaptation quality.

3. Forecasting Models

This chapter presents the forecasting models employed in this thesis. These models were chosen to form a heterogeneous pool of forecasters, incorporating both statistical and machine learning approaches. This diversity is valuable in dynamic model selection, as it combines different biases and strengths in the pool [35]. The following sections provide brief descriptions of each model, all trained for one-step-ahead forecasting on fixed-length input windows, as defined in section 4.2.

3.1. Classical Models

Classical methods refer to parametric approaches that explicitly assume the data follow a stochastic process. Their parameters are typically estimated via likelihood-based techniques, often corresponding to interpretable components such as level, trend, or seasonality. In this category we include exponential smoothing (and its Holt-Winters variants) and ARIMA.

3.1.1. Exponential Smoothing

Exponential Smoothing is a classical time series forecasting technique. Predictions are made using a weighted average of past observations, where the weights decrease exponentially over time [28]. The model is defined under the assumption that the trend and seasonal components are absent, but extensions exist to handle those as well.

Let y_t denote the observed value at time t and $\hat{y}_t$ denote the forecast made for time t. For a smoothing parameter $\alpha \in (0, 1)$, the forecast for time $t + 1$ is given by:

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\, \hat{y}_t$$

Holt's linear trend method extends simple exponential smoothing by introducing a second state component to capture trend [27]. The level l_t (i.e., baseline value) and the trend b_t are updated recursively as follows:

$$l_t = \alpha y_t + (1 - \alpha)\, (l_{t-1} + b_{t-1})$$
$$b_t = \beta\, (l_t - l_{t-1}) + (1 - \beta)\, b_{t-1}$$
$$\hat{y}_{t+1} = l_t + b_t$$

A third component s_t is added in the Holt-Winters method to capture seasonality [49].

3.1.2. ARIMA

Autoregressive Integrated Moving Average (ARIMA) is a time series forecasting model that combines autoregressive (AR) and moving average (MA) components along with differencing operations to handle non-stationarity in the data [6, 23].

An autoregressive process of order p, denoted AR(p), defines the current value of the series as a linear function of its previous p observations and a random error term:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

A moving average process of order q, denoted MA(q), expresses the current observation as a linear combination of the past q error terms:

$$y_t = \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

The integrated component applies differencing of order d to remove trends. Let B represent the backshift operator defined as $B^k y_t = y_{t-k}$. The differencing of order d is expressed as:

$$y'_t = (1 - B)^d y_t$$

By combining these components, the resulting $ARIMA(p, d, q)$ model is given by:

$$y'_t = \mu + \sum_{i=1}^{p} \phi_i y'_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

where μ is the mean of the series and ε_t the error term at time t.

3.2. Regression Models

Regression models treat forecasting as a supervised learning task, modeling lagged observations to predict future values, without assuming that the data are generated by a specific stochastic process. In this category we include linear regression, support vector regression (with kernels), and tree-based methods.

3.2.1. Linear Regression

Linear Regression is a fundamental statistical and machine learning model that assumes a linear relationship between the input features and the target variable [16]. For an input vector $\mathrm{x} = (x_1, x_2, \ldots, x_n)^\top$ at time t, the model predicts the next value as:

$$\hat{y}_{t+1} = \mathrm{w}^\top \mathrm{x} + b$$

where $\mathrm{w}^\top$ denotes the weight vector, and b the bias term. The model parameters w and b are typically estimated by minimizing the mean squared error between predicted and true target values over a training set:

$$\min_{\mathrm{w},b} \frac{1}{N} \sum_{t=1}^{N} (\hat{y}_{t+1} - y_{t+1})^2$$

Despite its simplicity, linear regression remains a commonly used baseline in time series forecasting. Nonetheless, its inability to capture nonlinear relationships fundamentally limits its performance in real-world scenarios.

3.2.2. Support Vector Regression

Support Vector Regression (SVR) extends Support Vector Machines (SVMs) to regression tasks [48, 45]. SVR models the underlying relationship as a linear function of the form $f(x) = \mathrm{w}^\top \mathrm{x} + b$. It aims to estimate w and b such that the predictions are within a margin of tolerance ε around the true observations, while minimizing model complexity (i.e., penalizing $\|\mathrm{w}\|$). The optimization problem is formulated as:

$$\min_{\mathrm{w},b,\xi_i,\xi_i^*} \quad \frac{1}{2}\|\mathrm{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

subject to:

$$y_i - \mathrm{w}^\top \mathrm{x}_i - b \le \varepsilon + \xi_i,$$

$$\mathrm{w}^\top \mathrm{x}_i + b - y_i \le \varepsilon + \xi_i^*,$$

$$\xi_i, \xi_i^* \ge 0, \quad i = 1, \ldots, N.$$

Here, $C > 0$ is a regularization parameter that controls the trade-off between model complexity and training error. ξ_i and ξ_i^* are slack variables representing points outside the ε-tolerance region.

SVR is often reformulated using the kernel trick, where the dot product $\mathrm{x}_i^\top \mathrm{x}_j$ is replaced by a kernel function $K(\mathrm{x}_i, \mathrm{x}_j)$. This allows SVR to operate in high-dimensional, nonlinear feature spaces without explicitly computing the transformation $\phi(\cdot)$. Common kernels include the linear kernel $K(\mathrm{x}_i, \mathrm{x}_j) = \mathrm{x}_i^\top \mathrm{x}_j$, polynomial kernel $K(\mathrm{x}_i, \mathrm{x}_j) = (\mathrm{x}_i^\top \mathrm{x}_j + c)^d$ and Gaussian kernel $K(\mathrm{x}_i, \mathrm{x}_j) = \exp\left(-\gamma \|\mathrm{x}_i - \mathrm{x}_j\|^2\right)$.

Support vector regression serves as a flexible model in the forecasting pool, particularly effective when the data contains smooth nonlinear patterns that linear models fail to capture.

3.2.3. Decision Tree Regression

Decision tree regression is a machine learning model that recursively partitions the input feature space into disjoint regions [9]. The model then fits a simple predictive model in each resulting region, typically the mean of the training samples in that region. At prediction time, the new input is routed through the tree according to the learned decision rules, and the prediction corresponds to the fitted model at the leaf node reached by the input.

During training, the tree is constructed in a top-down manner using a greedy splitting strategy. At each node, the algorithm selects the split that results in the lowest total MSE across the resulting child nodes:

$$\mathrm{MSE}_{\mathrm{split}} = \sum_{i \in \mathrm{left}} (y_i - \bar{y}_{\mathrm{left}})^2 + \sum_{i \in \mathrm{right}} (y_i - \bar{y}_{\mathrm{right}})^2$$

This recursive splitting continues until a stopping criterion is met, such as a maximum tree depth or a minimum number of samples in a leaf.

Decision trees capture nonlinear relationships and feature interactions without requiring feature transformation. Moreover, irrelevant features do not degrade the predictive performance, since they are ignored if they do not improve the splitting criterion.

3.2.4. Random Forest Regression

Random forest regression is an ensemble learning technique that combines outputs of multiple decision trees for improved predictive accuracy [8]. Each tree is trained independently on a random subset from the training set drawn with replacement. Additionally, at each split, only a randomly selected subset of the input features is used. This two-step randomization, over both data and features, introduces diversity among the individual trees, improving the ensemble's generalization performance.

At prediction time, the input is passed through all the trees of the ensemble and the final output of the model is the average of each tree's own individual prediction:

$$\hat{y}_{t+1} = \frac{1}{K} \sum_{k=1}^{K} \hat{y}_{t+1}^{(k)}$$

where $\hat{y}_{t+1}^{(k)}$ is the prediction of the k-th tree and K denotes the total number of trees in the ensemble.

3.2.5. Gradient Boosting Regression

Gradient boosting regression is another ensemble learning technique that constructs a strong predictive model by sequentially combining multiple weak learners, typically decision trees [18]. In contrast to random forests, where trees are trained independently, gradient boosting constructs trees sequentially in a stage-wise manner, where each new tree attempts to correct the errors made by the ensemble so far.

Given a training dataset, the algorithm starts with an initial model $F_0(\mathrm{x})$, typically the mean of the target values. For a differentiable loss function $\mathcal{L}(\cdot)$, this is expressed as:

$$F_0(\mathrm{x}) = \arg\min_{c} \sum_{i=1}^{N} \mathcal{L}(y_i, c)$$

At each boosting iteration $m = 1, 2, ..., M$, a new tree $h_m(\mathrm{x})$ is trained to predict the pseudo-residuals, defined as the negative gradients of the loss function with respect to the current ensemble's predictions. In other words, the tree is constructed to capture the direction in which the model should be updated to reduce the loss. Formally, pseudo-residuals are computed as:

$$r_{m,i} = -\frac{\partial \mathcal{L}(y_i, F_{m-1}(\mathrm{x}_i))}{\partial F_{m-1}(\mathrm{x}_i)}, \quad i = 1, \ldots, N.$$

The model is updated by adding the contribution of the newly trained tree:

$$F_m(\mathrm{x}) = F_{m-1}(\mathrm{x}) + \nu \cdot h_m(\mathrm{x})$$

where $\nu \in (0, 1]$ controls the contribution of each new tree. At prediction time, the final output is the sum of all the individual learners:

$$\hat{y}_{t+1} = F_M(\mathrm{x}) = F_0(\mathrm{x}) + \sum_{m=1}^{M} \nu \cdot h_m(\mathrm{x})$$

3.3. Neural Network Models

Neural networks, introduced in subsection 2.3.1, are particularly valuable in the context of forecasting because of their ability to capture nonlinear relationships and complex temporal dependencies. In this category we include multilayer perceptrons, LSTMs and CNN-LSTM models.

3.3.1. Multilayer Perceptron

A neural network in which every neuron in a given layer is connected to all neurons of the subsequent layer is called a fully connected neural network. When such a network contains one or more hidden layers between the input and output layers, it is referred to as a multilayer perceptron (MLP) [38]. As discussed in subsection 2.3.1, MLPs are capable of modeling complex nonlinear relationships between inputs and outputs. In forecasting tasks, they are trained to approximate the mapping between past observations and future values.

In time series forecasting, the input x is typically a fixed-length window of past observations, and the MLP is trained to predict the value at the next time step. Let $\mathrm{x} \in \mathbb{R}^d$ be an input vector. The output of the l-th layer is given by:

$$\mathrm{h}^{(l)} = \sigma\left(\mathrm{W}^{(l)}\mathrm{h}^{(l-1)} + \mathrm{b}^{(l)}\right),$$

where $\mathrm{W}^{(l)}$ and $\mathrm{b}^{(l)}$ denote the weights and bias vector of layer l, $\sigma(\cdot)$ represents a nonlinear Activation Function, and $\mathrm{h}^{(0)} = \mathrm{x}$. The network parameters (weights and biases) are learned by minimizing a loss function as described in subsubsection 2.3.1.

Although MLPs lack explicit mechanisms for handling sequential dependencies, they are still capable of capturing complex temporal patterns. This makes them particularly suitable in heterogeneous forecasting pools, especially for short input windows with nonlinear relationships.

3.3.2. Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are special neural networks specifically designed to model long-range dependencies in sequential data. They were introduced to address the limitations of standard recurrent neural networks (RNNs) in retaining information over long sequences [26].

Recurrent Neural Networks

Recurrent neural networks are neural network architectures developed to process sequential data by maintaining a hidden state h_t that acts as a memory of the past information. At each time step t, the hidden state h_t is updated based on the current input x_t and the previous hidden state h_{t-1}. This recurrent structure enables RNNs to capture temporal dependencies across sequences, particularly when relevant information is located within short temporal gaps. However, RNNs show weaknesses in learning long-term dependencies due to vanishing and exploding gradient problems during backpropagation through time [3].

LSTM networks address these limitations by introducing a more complex memory mechanism. Instead of relying only on the hidden state, LSTMs utilize a cell state c_t and a hidden state h_t in addition to a gate mechanism to control the information flow.

The hidden state h_t represents the short-term output at time t, while the cell state c_t functions as a long-term memory, holding information accumulated across multiple time steps. At each new time step t, the gate mechanism decides which information is added to or removed from the cell state. Each gate has learnable parameters and uses nonlinear activation functions, allowing selective regulation of information flow.

The Forget Gate is the first operation within an LSTM cell, deciding which information to discard from the cell state. It is computed as:

$$f_t = \sigma\left(W_f\left[h_{t-1}, \mathrm{x}_t\right] + b_f\right).$$

To decide which information to add to the cell state, the input gate layer first uses a sigmoid layer to determine which components of the cell state should be updated:

$$i_t = \sigma\left(W_i\left[h_{t-1}, \mathrm{x}_t\right] + b_i\right),$$

then a tanh layer determines whether to decrement or increment each value in the cell state:

$$\tilde{c}_t = \tanh\left(W_c\left[h_{t-1}, \mathrm{x}_t\right] + b_c\right).$$

By combining the two, the cell state is then updated as:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t.$$

Finally, the output gate applies a sigmoid layer to the cell state to decide which information to output:

$$o_t = \sigma\left(W_o\left[h_{t-1}, \mathrm{x}_t\right] + b_o\right),$$

$$h_t = o_t \odot \tanh\left(c_t\right).$$

3.3.3. Bidirectional Long Short-Term Memory

Bidirectional Long Short-Term Memory (BiLSTM) networks are an extension of standard LSTMs designed to improve the model's ability to capture contextual dependencies by processing the data in both forward and backward directions [41]. BiLSTMs employ two separate LSTM layers. In forecasting tasks, the forward LSTM processes the window in its original order whereas the backward LSTM uses the reverse form of the same sequence. At each time step t, the final hidden output is the concatenation of both hidden states.

BiLSTMs have been shown to outperform unidirectional LSTM models, particularly in natural language processing tasks such as text parsing and next word prediction. Nonetheless, in time series forecasting, the benefits of bidirectional over standard LSTMs are less clear [44].

3.3.4. Convolutional Long Short-Term Memory

Convolutional Long Short-Term Memory (CNN-LSTM) networks combine convolutional neural networks with LSTMs, allowing local pattern detection and long-range temporal modeling [42]. This makes CNN-LSTMs well suited for time series forecasting, where both components are relevant.

In this architecture, a one-dimensional convolutional layer acts as a feature extractor. Given an input time series window $\mathrm{x} \in \mathbb{R}^{T\times 1}$ of length T, the convolutional block applies a series of Convolution Operation followed by an activation function. Formally:

$$\mathrm{z}^{(k)} = \sigma\left(\mathrm{W}_{\mathrm{conv}}^{(k)} * \mathrm{z}^{(k-1)} + \mathrm{b}_{\mathrm{conv}}^{(k)}\right), \quad k = 1, \ldots, K,$$

where $*$ denotes the 1D convolution operation, K denotes the number of convolutional operations and $\sigma(\cdot)$ is a nonlinear activation function. Pooling layers might also be used as part of the convolutional block.

These features are then fed to the LSTM network, which models their sequential dependencies over time:

$$\mathrm{h}_t, \mathrm{c}_t = \mathrm{LSTM}(\mathrm{z}_t, \mathrm{h}_{t-1}, \mathrm{c}_{t-1}), \quad t = 1, \ldots, T.$$

The final hidden state is passed through a fully connected output layer to produce the forecast. CNN-LSTMs are used in our pool for their ability to combine spatial feature extraction with sequential modeling.

4. Autoencoder Embeddings for Adaptive Model Selection and Drift Detection

This chapter introduces the Autoencoder Embeddings for Adaptive Model Selection (AE-AMS) framework, a dynamic model selection approach that leverages latent embeddings rather than traditional raw time-domain representations for region of competence-based model selection. The main motivation behind this approach is the ability of autoencoders to extract meaningful latent similarities that might not be observable in raw time-domain representations.

4.1. Overview of the System

This section presents an overview of the proposed forecasting pipeline, which integrates dynamic model selection with concept drift detection using regions of competence and autoencoder-based latent embeddings. The system consists of three key components: a set of forecasting models, a convolutional autoencoder, and a dynamic selection strategy based on latent similarities.

In the first stage, a heterogeneous pool of forecasters is trained using windowed segments drawn from an early portion of the time series assumed to be stable. In parallel, a convolutional autoencoder is also trained to learn compact representations of these windows.

In the second stage, the system evaluates all forecasters on a separate validation set, also using windowed data. For each validation window, we determine which forecaster achieves the lowest prediction error. The corresponding embedding of that window, produced by the convolutional autoencoder, is then added to the region of competence of the winning model. This process builds a library of embeddings for each model, which reflects regions of the input space where it has demonstrated strong performance. Clustering is also performed in latent space within each model's region of competence.

At prediction time, each incoming input window is encoded using the same autoencoder. The system computes its distance to all stored embeddings. The forecasting model associated with the closest embedding is selected to perform the prediction. Reconstruction errors are also monitored as a mechanism for detecting distribution changes. When such changes are observed, the autoencoder is retrained to reflect the new concept, and the regions of competence are subsequently updated.

4.2. Data Preprocessing

The first step in the AE-AMS framework involves transforming the raw time series into a structured input suitable for the forecasting pipeline. This process includes partitioning the dataset into training, validation, and test subsets; normalizing the values into a consistent scale; and applying a sliding-window segmentation to generate overlapping fixed-length sequences of the series.

4.2.1. Data Partitioning

Let $Y = [y_1, y_2, ..., y_N]$ denote the raw time series, where N is the total number of observations. The series is partitioned into three disjoint subsets:

- **Training set**

$$Y^{\text{train}} = \left[y_1, \cdots, y_{\lfloor 0.5N \rfloor}\right]$$

  The training set contains the first 50% of observations of the raw time series. Both the forecasting models and the convolutional autoencoder are trained on this subset.

- **Validation set**

$$Y^{\text{val}} = \left[y_{\lfloor 0.5N \rfloor+1}, \cdots, y_{\lfloor 0.75N \rfloor}\right]$$

  The validation set contains the subsequent 25% of observations. It is reserved for constructing the RoCs of the forecasting models.

- **Test set**

$$Y^{\text{test}} = \left[y_{\lfloor 0.75N \rfloor+1}, \cdots, y_N\right]$$

  The test set contains the last 25% of observations and is used to evaluate the performance of the proposed method against a baseline approach that performs model selection directly in the raw time domain.

4.2.2. Data Normalization

Normalization is a critical preprocessing step in time series forecasting. It is applied to standardize the scale of the observations, improving the performance and stability of the forecasting models [24]. In this work, z-score normalization is employed to transform the data such that it has zero mean and unit variance.

Definition: Z-score Normalization [24]

For time series $Y = [y_1, y_2, ..., y_N]$ with mean $\bar{Y}$ and standard deviation σ_Y, the transformation is given by:

$$y'_t = \frac{y_t - \bar{Y}}{\sigma_Y}, \quad t = 1, \ldots, N.$$

To prevent data leakage, the normalization statistics ($\bar{Y}$ and σ_Y) are computed using exclusively the training set. This ensures that information from future observations does not influence the model during training or evaluation.

4.2.3. Data Windowing

After splitting and normalizing the raw time series data, each subset is transformed into overlapping input-target pairs using a sliding window approach. For a given window length w and a normalized subset $Y^* = [y_a, y_{a+1}, ..., y_b]$, the i-th window is defined as:

$$x_i = [y_i, y_{i+1}, ..., y_{i+w-1}], \quad a \leq i \leq b - w,$$

with its corresponding one-step-ahead target given by:

$$t_i = y_{i+w}.$$

This results in a set of x_i, t_i pairs for each split. We denote the resulting windowed subsets as:

$$\left(Y_{\text{win}}^{\text{train}},\ T^{\text{train}}\right),\ \left(Y_{\text{win}}^{\text{val}},\ T^{\text{val}}\right),\ \left(Y_{\text{win}}^{\text{test}},\ T^{\text{test}}\right).$$

4.3. Autoencoder Architecture

AE-AMS utilizes a one-dimensional convolutional autoencoder to encode fixed-size windows into compact latent representations. The encoder consists of three convolutional layers, each using the ReLU activation function. Between each layer, batch normalization is applied to stabilize and accelerate training. The first convolutional layer takes a single-channel input window and applies 32 filters using a kernel of size 3 with same padding. The next convolutional layer increases the channels to 64. Before the final layer, a dropout layer is applied, followed by a max pooling layer. The final convolutional layer maps the resulting feature maps to the latent space with a configurable number of channels.

The dropout layer is used for regularization (i.e., to prevent overfitting). By randomly deactivating a certain percentage of neurons, dropout helps prevent the model from memorizing the training data and performing poorly on new data [22]. The max pooling layer, with a kernel size of 2, reduces the sequence length by half, allowing the autoencoder to preserve only the most important information of the input.

The decoder reverses this process. First, a transposed convolution simultaneously upsamples the sequence back to its original length and increases the number of channels from the latent dimension back to 64. A convolutional layer then reduces the channels to 32, followed by a dropout layer. The final convolutional layer restores the data to its original single-channel format.

The autoencoder is trained on the same data as the forecasting models and serves two primary purposes in the system: generating latent representations used for dynamic model selection and reconstruction errors to detect concept drift.

4.4. Training

The convolutional autoencoder and the forecasting models are trained independently using the windowed training set $(Y_{\text{win}}^{\text{train}},\ T^{\text{train}})$. The autoencoder is trained to minimize the mean squared reconstruction error between the input windows and their reconstructions, formally:

$$\mathcal{L}_{\text{AE}} = \frac{1}{M}\sum_{i=1}^{M} \|x_i - \hat{x}_i\|_2^2,$$

where M is the number of training windows, x_i is the input window, and $\hat{x}_i$ the reconstruction produced by the decoder.

Each forecasting model in the pool is trained to minimize the one-step-ahead prediction error. For a forecasting model f_θ with parameters θ, the objective function is:

$$\mathcal{L}_{\text{forecast}} = \frac{1}{M}\sum_{i=1}^{M} \left(t_i - f_\theta(x_i)\right)^2,$$

where t_i denotes the target value associated with x_i.

4.5. RoCs Construction

Within AE-AMS, each forecasting model in the pool is associated with a region of competence, which represents the time series windows where the model has demonstrated superior forecasting performance. Once the forecasting models and the convolutional autoencoder have been trained, the next step consists of constructing these regions for each model.

Given a pool of forecasters $\mathbb{P} = \{f_1, \ldots, f_K\}$ and a windowed validation set $\left(Y_{\text{win}}^{\text{val}},\ T^{\text{val}}\right)$, the RoCs are built as follows. For each validation window $x_i \in Y_{\text{win}}^{\text{val}}$, each model $f_j \in \mathbb{P}$ produces a forecast $\hat{t}_{j,i}$:

$$\hat{t}_{j,i} = f_j(x_i).$$

The predictions are compared against the true target value $t_i \in T^{\text{val}}$ using mean squared error:

$$MSE_{j,i} = \left(\hat{t}_{j,i} - t_i\right)^2.$$

The model with the lowest error on window x_i is declared the winner, formally:

$$j^* = \arg\min_j MSE_{j,i}.$$

The corresponding window x_i is then encoded using the trained convolutional autoencoder and assigned to the RoC of the winning model f_{j^*}:

$$z_i = \text{encoder}(x_i),$$

$$\text{RoC}_{j^*} \leftarrow \text{RoC}_{j^*} \cup \{z_i\}.$$

By the end of the validation phase, each model has built a personalized region of competence that reflects its localized strengths across the validation set. Additionally, k-means clustering is performed on each forecaster's region of competence in latent space. These centers can be used later instead of the full RoC during model selection to reduce computational cost. Moreover, decoding the cluster centers provides visual explanations of the selection process.

Definition: K-means Clustering [34]

K-means clustering is an unsupervised learning algorithm used to partition a dataset into k distinct, non-overlapping clusters based on similarity. Given a dataset of n points, and a fixed integer k, k-means aims to identify a set of clusters $\{C_1, C_2, \cdots, C_k\}$, such that the within-cluster sum of squared errors (WCSS) is minimized:

$$\arg\min_C \sum_{c=1}^{k} \sum_{z_i \in C_c} \|z_i - \mu_c\|^2,$$

where μ_c is the centroid of cluster C_c. The algorithm proceeds iteratively, by first assigning each point to the nearest centroid and recomputing each centroid as the mean of the points assigned to it. This process iterates until cluster assignments stabilize, typically when centroids no longer change or the decrease in WCSS becomes negligible.

For each model f_j, AE-AMS begins by determining the optimal number of clusters k^* using the elbow method [30]. For a set of candidate values $K = \{k_{min}, k_{min} + 1, \ldots, k_{max}\}$, the algorithm computes the

within-cluster sum of squares across all candidate values $k \in K$. The optimal number of clusters k^* is identified as the point of maximum curvature ("elbow"), where adding more clusters results in diminishing returns in terms of WCSS. Once the optimal number of clusters k^* is determined, k-means clustering is performed one last time using k^* and the resulting centroids define the clustered regions of competence $\text{RoC}_j^{\mu} = \{\mu_{j,1}, \mu_{j,2}, \ldots, \mu_{j,k^*}\}$.

4.6. Online Model Selection

At prediction time, AE-AMS performs model selection by encoding the current input window x_i into its latent representation $z_i = \text{encoder}(x_i)$ and comparing it against the regions of competence RoC_j across all the candidate models in the pool. To achieve this, a distance metric is used to quantify the similarity between this embedding and the stored latent representations within each model's RoC.

The system performs online model selection in five stages. Let $\mathbb{P} = \{f_1, \ldots, f_K\}$ denote the pool of forecasting models, x_i denote an input window and $d(\cdot, \cdot)$ is a predefined distance. Real-time prediction is performed via the following steps:

1. **Input Encoding**

   The input window x_i is encoded using the trained convolutional autoencoder:

   $$z_i = \text{encoder}(x_i)$$

2. **Distance Computation**

   For each forecaster $f_j \in \mathbb{P}$, we compute the minimum distance between the encoded window z_i and its $\text{RoC}_j = \{z_1, z_2, ..., z_{N_j}\}$:

   $$D_{j,i} = \min_{z \in \text{RoC}_j} d(z_i, z).$$

3. **Model Selection**

   The model whose RoC is closest to the current input is selected for the forecast:

   $$j^* = \arg\min_j D_{j,i}.$$

4. **Forecasting**

   The selected model f_{j^*} produces the forecast:

   $$\hat{t}_i = f_{j^*}(x_i)$$

5. **Drift Monitoring**

   In parallel, the autoencoder reconstruction error is forwarded to the drift detection module:

   $$AE_{error} = \|x_i - \text{decoder}(z_i)\|^2$$

4.7. Drift Detection

AE-AMS integrates an unsupervised drift detection mechanism, based on Hoeffding's inequality, to adapt to distribution changes. The mechanism relies on monitoring the autoencoder's reconstruction error, identifying statistically significant deviations from its expected range, and triggering model adaptation when concept drift is detected.

First, reference statistics are computed from the validation set. For each validation window, we compute the reconstruction error between the input and its autoencoder reconstruction. These values are then used to estimate the initial reference mean μ_0, defined as:

$$\mu_0 = \frac{1}{N} \sum_{i=1}^{N} e_i,$$

where $e_i = \|x_i - \hat{x}_i\|^2$ represents the reconstruction error of the i-th validation window and N denotes the total number of validation windows. In addition to the reference mean, we also estimate the range parameter R required by Hoeffding's inequality. Specifically, we compute the 1.5th and 98.5th percentiles of the reconstruction error values, denoted $q_l^{(e)}$ and $q_h^{(e)}$. The range is then defined as a scaled interval between these quantiles:

$$R_{\text{recon}} = \gamma \cdot \left(q_h^{(e)} - q_l^{(e)} \right),$$

where γ is a safety parameter that provides an additional margin to ensure robustness against unseen data.

At prediction time, the detector updates a running mean $\hat{\mu}_t$ with each new window and compares it against the reference mean μ_0 using the Hoeffding's inequality. A drift signal is triggered if

$$|\hat{\mu}_t - \mu_0| > \sqrt{\frac{R_{\text{recon}}^2 \ln\left(\frac{2}{\delta}\right)}{2W}},$$

where W denotes the number of recent windows used to compute the running mean and δ denotes the significance level parameter, which specifies the maximum probability of a false alarm. This is equivalent to a confidence level of $1 - \delta$.

Following drift detection, the most recent observations of the time series are extracted to serve as an adaptation set. This set is partitioned into two subsets. The first subset is used to retrain the convolutional autoencoder, enabling it to better capture the new data distribution and produce stable reconstruction errors for post-adaptation drift detection. The second subset is then used to rebuild each model's region of competence following the same procedure described in section 4.5. The previously stored RoCs are discarded and replaced by the newly constructed ones. The size of the adaptation set, the train/validation split ratio, and the choice to append or replace the RoCs are all configurable parameters of the framework.

5. Experiments

In this chapter, we present the experiments conducted to evaluate the proposed framework. We aim to investigate the benefits of using autoencoder-based latent embeddings over raw time-domain representations for dynamic model selection. As part of this evaluation, we also compare different distance metrics to assess their impact on selection performance. Moreover, we explore whether clustering the regions of competence in latent space leads to improvements in forecasting performance or provides higher explainability of the selection process. Lastly, we evaluate whether applying Hoeffding's inequality to the reconstruction error provides more effective concept drift detection than applying it to the mean values of the series.

Accordingly, the experiments are guided by the following research questions:

- **Q1:** Does using latent embeddings improve dynamic model selection compared to using raw time-domain representations, and how does the choice of distance metric influence the selection performance?
- **Q2:** How does using RoC cluster centers instead of the unclustered RoC affect forecasting performance, and can visualizing these centers provide deeper insights into the competence of each model?
- **Q3:** Is monitoring the reconstruction error more effective for detecting concept drift than the mean values of the series in terms of trigger frequency and model performance?

All experiments were implemented in Python using PyTorch and scikit-learn, and the complete code is publicly available on GitHub.

5.1. Experimental Setup

For the experimental evaluation, we use a diverse collection of publicly available time series datasets covering a range of domains such as energy, economics, environment, and healthcare. These datasets were selected to represent different application areas and sampling frequencies. All series are univariate, and the forecasting horizon is fixed to one step ahead. Missing values were handled using Last Observation Carried Forward (LOCF). Table 5.1 provides an overview of the datasets in our experiments.

All datasets are preprocessed as described in section 4.2, which involves z-score normalization, windowing into fixed-size overlapping sequences, and splitting the data into training, validation, and test sets. The window size is treated as a configurable parameter within AE-AMS and is fixed to **10** in all experiments. The training and validation sets are used for training the forecasting models and building the RoCs, respectively. The test set is kept exclusively for evaluation and will be revealed sequentially to simulate real-world conditions. For evaluation, we use the mean squared error, averaged across all prediction steps in the test set. All errors are computed on the z-score normalized series to ensure comparability across datasets.

Table 5.1.: Summary of univariate time series used in the experiments.

Dataset	Size	Freq.	Description
Temperatures	3650	Daily	Minimum daily temperatures, Melbourne (DataMarket) [7]
Births	7306	Daily	Daily female births in the United States (Monash) [21]
SolarPower	4194	Hourly	Solar power production in MW (Monash) [21]
WindPower	8220	Hourly	Wind power production in MW (Monash) [21]
ExchangeRate	6848	Daily	USD exchange rates from the European Central Bank [14]
Treasury	12808	Daily	U.S. treasury yields (FRED) [15]
AirQuality	9358	Hourly	C6H6 measurements in Italy (UCI) [12]
OfficeTemp	7268	Hourly	Office temperature sensor data (NAB) [31]

5.1.1. Forecasting Pool Specification

For our experiments, we construct a pool $\mathbb{P}$ of 15 candidate models, covering statistical, machine learning, and deep learning approaches. This diversity provides AE-AMS with a broad coverage of modeling strengths. A detailed overview of the individual models is presented in chapter 3.

The statistical models include Exponential Smoothing and ARIMA. The ARIMA model is configured with an autoregressive component of order one, without differencing or moving average terms, while the Exponential Smoothing forecaster is implemented as the standard simple exponential smoothing (SES) variant, excluding trend and seasonal components.

The machine learning models comprise linear and kernel-based regressors as well as tree ensembles. Specifically, we include a Linear Regression model, a Support Vector Regressor with a radial basis function kernel, and three tree-based models: a Decision Tree with maximum depth three, a Random Forest with fifty estimators and restricted depth, and a Gradient Boosting Regressor trained with shallow trees and a reduced learning rate. All of these models are implemented using scikit-learn and trained to minimize mean squared error.

The neural forecasting models are implemented in PyTorch and cover fully connected, recurrent, and hybrid architectures. Two Multilayer Perceptrons (MLPs) are used: one with a single hidden layer of size 8 and another with two hidden layers of sizes 16 and 8, both trained with Adam optimization. Recurrent models include two Long Short-Term Memory networks (LSTMs) with one and two layers respectively, and two bidirectional LSTMs (BiLSTMs), all with small hidden sizes and dropout regularization. Finally, two CNN-LSTM hybrids are considered, in which temporal convolutional filters first extract local features before being processed by one or two stacked LSTM layers. All neural models are trained using mean squared error as the loss function.

Each model in the pool is trained independently on the training partition of each dataset, and the validation set is subsequently used to construct its corresponding Region of Competence. To ensure reproducibility, all experiments are conducted with fixed random seeds. The complete configurations of these models are summarized in Appendix A.

5.1.2. AE-AMS Configuration

AE-AMS relies on several global hyperparameters to control clustering, drift detection, and adaptation. For clustering, k-means is applied to each region of competence using the elbow method. The number of

clusters is bounded between 2 and $\lfloor\sqrt{N/2}\rfloor$, where N is the number of latent vectors in that region. Drift detection is performed using Hoeffding's inequality on reconstruction errors with a significance level $\delta = 0.023$ and safety margin $\gamma = 1.85$, where the reference mean is computed from the reconstruction errors of the validation set. When drift is detected, the system adapts using the 25% of the full series length that immediately precedes the drift point. This adaptation set is split into 75% for retraining the autoencoder and 25% for reconstructing the regions of competence. The old RoCs are discarded and replaced with the newly constructed ones to reflect the new distribution, while the running mean reset as the new reference mean and W is reset to zero. At prediction time, a single distance metric is fixed for all similarity computations, while multiple candidate metrics are evaluated later to assess their influence on selection performance. A summary of all hyperparameters can be found in Table 5.2.

The convolutional autoencoder used in AE-AMS is trained on sliding windows of length 10. Training is performed for 30 epochs with the Adam optimizer at a learning rate of 0.001, a batch size of 128, and a dropout rate of 0.3 using mean squared error as the reconstruction loss. The bottleneck of the autoencoder is set to 2 in all experiments. After training, the encoder is employed to produce latent representations of all subsequent windows, which are then used for constructing the regions of competence.

Table 5.2.: AE-AMS hyperparameters used in the experiments.

Parameter	**Value**	**Description**
Sliding window length	10	Length of each input window for forecasting and embeddings.
Latent dimension	2	Size of the autoencoder bottleneck.
Number of clusters	$2 - \lfloor\sqrt{N/2}\rfloor$	Range of k when clustering RoCs with N latent vectors.
Hoeffding's significance level	0.023	Significance level for Hoeffding's inequality.
Safety parameter	1.85	Scales the percentile range of reconstruction errors for robustness.
Adaptation set size	Last 25%	Portion of recent observations used after drift.
Append RoCs	False	Old RoCs replaced (not appended) during adaptation.
Adaptation split	75%/25%	Train/validation split for autoencoder retraining and RoC rebuild.

5.1.3. Comparative Baselines

To assess the effectiveness of AE-AMS, we implement a baseline variant of our framework that operates directly in the raw time domain. In this mode, similarity is computed between the current input window and the stored (non-encoded) windows in each model's region of competence. This setup allows us to isolate the contribution of autoencoder-based representations, as all other components remain identical to the full AE-AMS system described in chapter 4 and Table 5.2.

By default, AE-AMS computes similarities using the full set of latent vectors in each region of competence. To evaluate the impact of clustering, we also implement a variant where the RoC is represented

solely by its cluster centers. This clustered-RoC variant is assessed separately in the experimental evaluation.

For clarity, we use the following naming convention when referring to system variants throughout the evaluation. **AE-AMS** denotes the full autoencoder-based framework, while **RAW-AMS** refers to its raw time-domain variant. Each configuration is paired with a similarity metric, indicated by the suffix **-EUC** (Euclidean), **-DTW** (Dynamic Time Warping), or **-COS** (cosine distance). When the region of competence is represented by cluster centers rather than all stored windows, we add a **C-** prefix to the name. For example, C-AE-AMS-EUC denotes the clustered latent variant evaluated with Euclidean distance.

5.2. Results

This section presents the results of the experimental evaluation, beginning with Research Question **Q1**. The first question investigates whether autoencoder-based latent embeddings provide an advantage over raw time-domain representations for dynamic model selection. Specifically, we examine whether computing similarities in latent space leads to improved forecasting performance. Furthermore, we assess how the choice of similarity metric influences the relative performance of the selection. To address this question, we compare the baseline **RAW-AMS** variants against their corresponding **AE-AMS** counterparts across all datasets and distance metrics. For additional context, we also add **ARIMA** and **SES** models as a reference baseline outside the AMS framework. The forecasting accuracy is measured in terms of mean squared error on the normalized data, with lower values indicating better performance.

Table 5.3 summarizes the aggregated forecasting performance for all dataset-metric combinations. The results show that **AE-AMS consistently outperforms RAW-AMS**: it achieves a lower overall average MSE (0.163 vs. 0.179) and records three times as many wins (18 vs. 6). On the datasets where AE-AMS is superior, it yields an average improvement of 9.1% relative to RAW-AMS, while the cases in which RAW-AMS performs better provide only an improvement of 2.7%. This indicates the latent representation provides a more robust and effective feature space for determining regions of competence and computing similarities, leading to not only more frequent but also larger improvements. Detailed results for each dataset and metric are provided in Table A.4.

Method	Avg. MSE	Wins	Losses	Avg. % Improvement (on wins)
AE-AMS	**0.163**	**18**	6	**9.10%**
RAW-AMS	0.179	6	**18**	2.74%
ARIMA	0.314	–	–	–
SES	0.399	–	–	–

Table 5.3.: Overall forecasting performance across all datasets and distance metrics (*dataset* × *metric*, 24 comparisons in total). Average MSE is reported for all methods, while wins/losses and relative improvements are shown only for AE-AMS against its RAW-AMS counterpart. ARIMA and SES are included as additional reference baselines.

We now focus on the impact of the similarity metric used in the model selection process. Both RAW-AMS and AE-AMS can operate with different distance functions when comparing the current input win-

dow to the stored regions of competence. In our experiments, we evaluate three metrics: **Euclidean distance**, **Dynamic Time Warping** (DTW), and **Cosine distance**. Each metric represents different approaches to measuring similarity: Euclidean distance captures pointwise deviations, DTW handles local temporal misalignments, and Cosine distance focuses on directions rather than magnitude.

Table 5.4 presents the average forecasting performance of AE-AMS and RAW-AMS under different similarity metrics. The results show that cosine distance consistently provides better performance for both variants, reducing the average MSE relative to Euclidean distance by 6.9% for AE-AMS and 7.4% for RAW-AMS. Euclidean distance remains a strong and reliable baseline, while DTW performs worst in both cases with substantially higher errors compared to Euclidean.

DTW is widely considered a highly effective similarity measure for time series [4], however it does not outperform Euclidean distance in our experiments. This can be explained by the data preprocessing and the AE-AMS configuration. The short fixed window length of only 10 observations substantially limits the potential benefits of temporal warping. When combined with z-score normalization, where amplitude differences are removed, there is often little meaningful phase shift within such small segments to correct. Under these conditions, DTW provides little advantage and may even amplify noise (by aligning noise with noise), leading to slightly worse performance than Euclidean distance. Interestingly, cosine distance outperforms both Euclidean distance and DTW in our setting. This observation could be explained by the same conditions causing DTW's underperformance: Since all windows are z-score normalized, magnitude differences across windows provide minimal additional information. As a result, the direction of variation becomes the most informative characteristic, which cosine distance is particularly well-suited to capture. Moreover, in low-dimensional spaces, angular similarity is often more robust to noise and small fluctuations, which explains why cosine distance achieves the best overall performance.

Method	**Metric**	**Avg. MSE**	**% improvement vs. EUC**
AE-AMS	Euclidean	0.165	–
	DTW	0.171	−4.23%
	Cosine	**0.153**	**+6.95%**
RAW-AMS	Euclidean	0.176	–
	DTW	0.197	−11.91%
	Cosine	**0.163**	**+7.40%**

Table 5.4.: Per-metric averages across all datasets. Positive values indicate lower (better) MSE than the Euclidean baseline within the same method.

To complement the average MSE comparison, we also analyze win rates, defined as the proportion of individual windows where AE-AMS achieves lower error than its raw counterpart. To further account for the magnitude of improvements, we also compute weighted win rates. In this variant, each window is weighted by the larger of the two errors. The model with the lowest error on a window receives that weight as a win. The weighted win rate is then calculated as the sum of weights won by a model divided by the total sum of weights across all windows.

The results in Table 5.5 show that AE-AMS achieves overall higher win rates than RAW-AMS across all similarity metrics. The latent variants outperform the raw ones by +2.1 percentage points (pp) in the

unweighted measure and +2.8 in the weighted measure, confirming that latent embeddings provide both frequent and meaningful improvements. In the unweighted measure, DTW has the largest improvement (+4.8), suggesting that latent embeddings shifts decisions toward better models more frequently. However, those wins are not necessarily associated with significant error reductions, as shown in the weighted analysis, where the gain drops to +3.0. Euclidean distance shows only a small improvement (+1.1 and +0.8), together with the highest same-decision rate (44.3%), indicating that both versions make similar selections most of the time. Remarkably, Cosine distance shows the smallest unweighted gain (+0.4) but the largest weighted improvement (+4.7). This indicates that while latent Cosine does not significantly increase the frequency of wins, it is particularly effective at correcting large errors.

Measure	**Metric**	**Raw (%)**	**Latent (%)**	**Δ (pp)**	**Same-decision (%)**
Unweighted	All	49.0	51.0	+2.1	31.1
	DTW	47.6	52.4	+4.8	27.4
	Euclidean	49.4	50.6	+1.1	44.3
	Cosine	49.8	50.2	+0.4	21.5
Weighted	All	48.6	51.4	+2.8	31.1
	DTW	48.5	51.5	+3.0	27.4
	Euclidean	49.6	50.4	+0.8	44.3
	Cosine	47.6	52.4	+4.7	21.5

Table 5.5.: Win-rate and weighted win-rate summaries across all datasets. Values are percentages. Δ is the difference in percentage points between latent and raw.

To statistically compare the forecasting performance of the different methods across datasets, we employ a Critical Difference (CD) diagram based on the Wilcoxon signed-rank test with Holm correction ($\alpha = 0.05$). The test ranks the methods according to their average performance on each dataset, with lower ranks indicating better forecasting accuracy.

Figure 5.1 shows that the three latent-space methods consistently achieve the lowest average ranks, all clustered closely together around the top position. On the other hand, the raw-space methods occupy the higher ranks, indicating a lower performance. However, the horizontal bar covering all methods indicates that the observed differences are not statistically significant at the 0.05 level according to the Wilcoxon signed-rank test. Despite the lack of statistical significance, the clear separation in average ranks between latent and raw methods is consistent with the findings observed in average MSE and win-rate analyses. Among the latent methods, cosine achieves the best overall rank, while DTW performs worst in the raw space. The critical difference reinforces our conclusion that latent embeddings provide a more stable similarity space. However, larger-scale experiments would be needed to establish significance.

We now turn to **Q2**, examining how clustering the regions of competence affects both forecasting performance and explainability. Instead of storing the full set of windows associated with each model, we apply k-means clustering and retain only the resulting centroids. At prediction time, similarities are computed against these centroids rather than the entire RoC. This reduces the computational cost of similarity search, as comparisons are made against a small number of representative vectors instead of all historical windows. Moreover, clustering also has the potential to enhance interpretability: by decoding

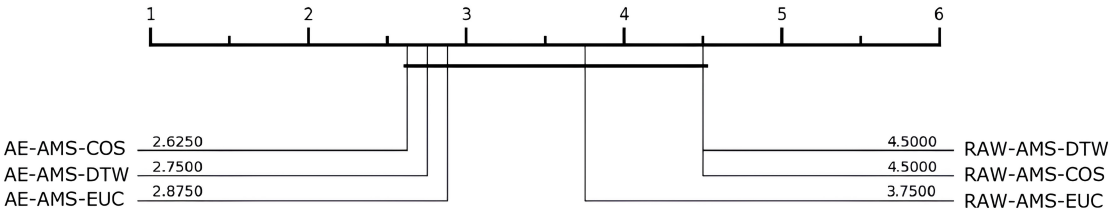

Figure 5.1.: Critical difference diagram comparing forecasting methods across all datasets. Methods connected by a horizontal bar are not significantly different according to the Wilcoxon signed-rank test (with Holm correction, $\alpha = 0.05$).

the latent cluster centers back into the input space using the autoencoder, we obtain patterns that can provide visual insight into the structures where each model is most competent.

Table 5.6 compares the overall forecasting performance of clustered and unclustered model selection. The results show that AE-AMS achieves the lowest average error (0.163), outperforming its clustered variant. AE-AMS dominates C-AE-AMS in 19 out of 24 dataset–metric combinations, while the clustered variant wins in only 5 cases. Furthermore, when AE-AMS is superior, it achieves a mean improvement of 12.32%, notably larger than the 7.96% average gain observed when it is outperformed. These outcomes suggest that while clustering occasionally improves predictions, it overall discards essential information, specifically the diversity present in the unclustered RoC. More importantly, both latent-based methods still surpass RAW-AMS in average error, suggesting that the latent embedding provides a more effective similarity space. Detailed results for each dataset and metric are provided in Table A.5.

Method	**Avg. MSE**	**Wins**	**Losses**	**Avg. % Improvement (on wins)**
C-AE-AMS	0.176	5	**19**	7.96%
AE-AMS	**0.163**	**19**	5	**12.32%**
RAW-AMS	0.179	–	–	–
ARIMA	0.314	–	–	–
SES	0.399	–	–	–

Table 5.6.: Overall forecasting performance across all datasets and distance metrics (*dataset* × *metric*, 24 comparisons in total). Average MSE is reported for all methods, while wins/losses and relative improvements are shown only for AE-AMS against its C-AE-AMS counterpart. ARIMA, SES and RAW-AMS are included as additional reference baselines.

We further compare C-AE-AMS to its unclustered counterpart on all metric configurations through a critical difference diagram using the Wilcoxon signed-rank test with Holm correction ($\alpha = 0.05$). The results, shown in Figure 5.2, demonstrate a clear separation between the two modes. All three AE-AMS variants are grouped toward the lower average ranks, while all three C-AE-AMS variants occupy higher

ranks. Although the test indicates that no statistically significant differences are present at the chosen confidence level, it still provides strong evidence that **AE-AMS outperforms C-AE-AMS** in terms of forecasting performance. It is worth noting that cosine distance remains the strongest metric in both modes, achieving the best rank overall in AE-AMS and the best rank within the clustered variants. This suggests that the degradation in performance is primarily due to the clustering step rather than the choice of metric.

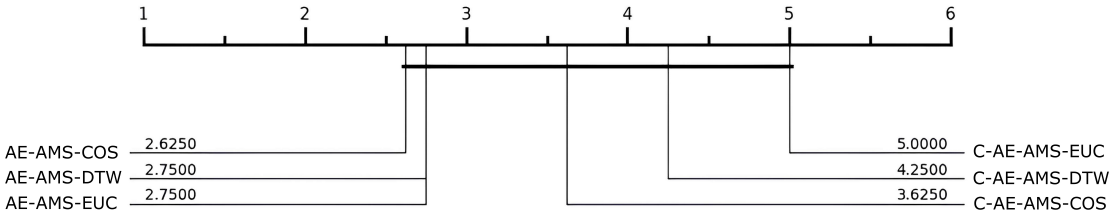

Figure 5.2.: Critical difference diagram comparing forecasting methods across all datasets. Methods connected by a horizontal bar are not significantly different according to the Wilcoxon signed-rank test (with Holm correction, $\alpha = 0.05$).

Although C-AE-AMS fails in terms of predictive performance, from an efficiency perspective clustering provides significant improvements. Table 5.7 highlights that C-AE-AMS achieves a substantial reduction in computational cost compared to its unclustered counterpart. On average across all datasets, runtime decreases from 63.4 seconds to 5.4 seconds on average per evaluation. DTW, in particular, shows the biggest improvement, where clustering reduces the cost from over 2 minutes to less than 7 seconds. Although less pronounced, Cosine and Euclidean distances show similar reductions. These results demonstrate a clear trade-off between efficiency and predictive performance.

Metric	**AE-AMS (s)**	**C-AE-AMS (s)**
DTW	121.35	**6.96**
Cosine	54.78	**5.14**
Euclidean	13.92	**4.02**
Overall	63.35	**5.37**

Table 5.7.: Average runtime per metric across all datasets for AE-AMS and C-AE-AMS.

We now shift our focus to the interpretability of the latent clusters. By decoding the cluster centers back into the input space, we can visually inspect them to assess whether they form meaningful patterns that characterize the regions of input space where each model performs well. As additional context, we also include cluster visualizations in which one cluster center is shown together with its members. This allows us to evaluate whether clustering enhances the explainability of the selection process or helps explain the weak performance of C-AE-AMS.

We observed that the quality of the decoded clusters is not consistent across datasets. Some datasets produce highly expressive clusters with interpretable patterns, while others collapse into nearly flat or parallel structures. This distinction is reflected in the performance of the clustered autoencoder-based model selection. To illustrate this, we take a closer look at two datasets: Births, where clustering results in a minor improvement, and Treasury, where it leads to a significant degradation. The combined results are displayed in Table 5.8.

Dataset	%Δ (C vs AE)	Wins (C vs AE)
Births	**+1.45%**	**1**
Treasury	-28.21%	0

Table 5.8.: Relative change of C-AE-AMS vs AE-AMS. Wins are counted out of the 3 metrics

The Births dataset illustrates well-structured clusters. As shown in Figure 5.3, the cluster centers produced by each model display distinct temporal characteristics such as variations in peak amplitude, phase shifts, oscillatory behavior, and overall shapes. These patterns are easy to interpret and suggest that the clustering has successfully grouped windows into meaningful clusters.

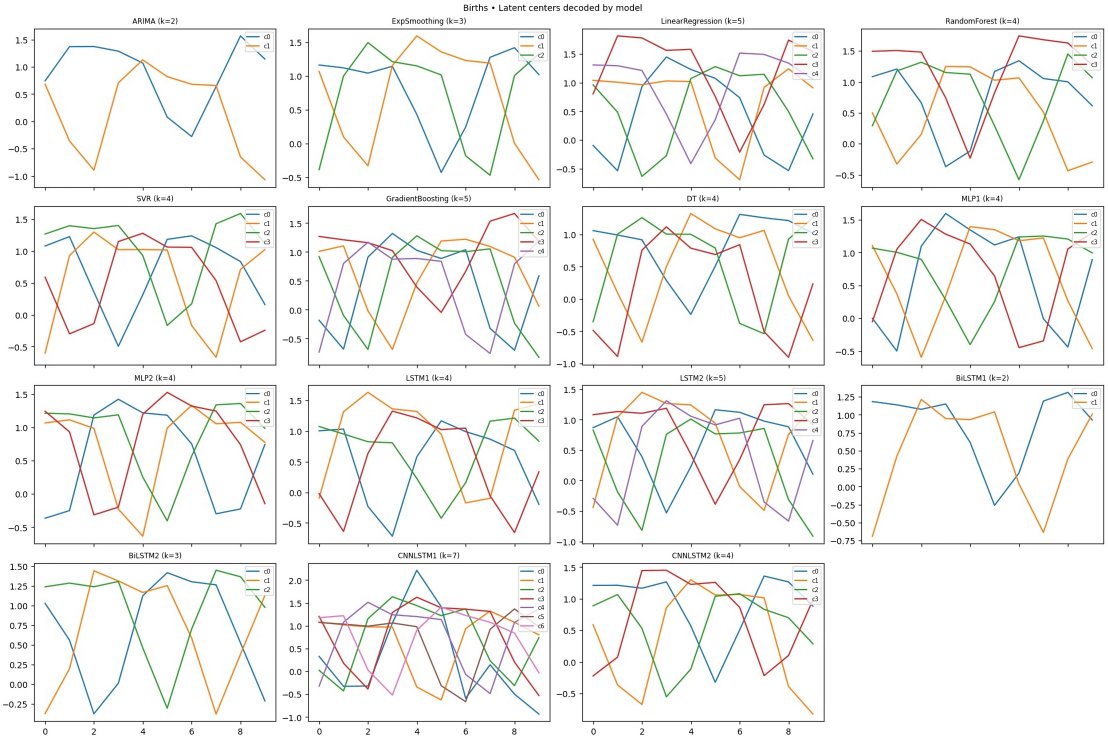

Figure 5.3.: Decoded cluster centers of the RoCs for the Births dataset, separated by forecasting model.

The cluster visualizations, presented in Figure 5.4, confirm that the centers provide meaningful representations of a model's competence in the input space. The quality of the clusters is also reflected in

slight performance gain observed in Table 5.8.

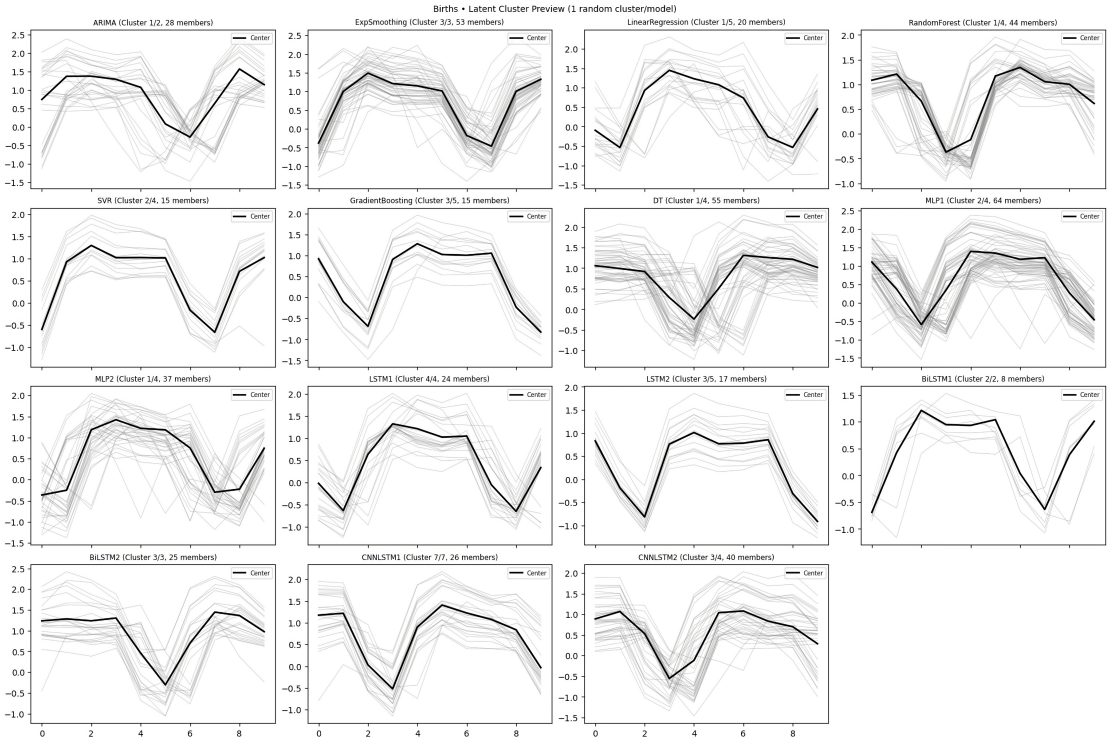

Figure 5.4.: Cluster previews of the RoCs for the Births dataset, separated by forecasting model. Each subplot shows one randomly selected cluster with its decoded center (bold) and members (faint).

The Treasury dataset, in contrast, illustrates the limitations of clustering for explainability. As shown in Figure 5.5, the decoded clusters primarily consist of flat and parallel structures with little temporal variation. This makes it difficult to interpret them as specific regions of model competence. In this case, clustering substantially degrades forecasting performance, as shown in Table 5.8, where C-AE-AMS leads to forecasting performance drops of more than 28% compared to AE-AMS across the three metrics. This example highlights how the effectiveness of clustering depends strongly on the underlying variability of the data: when temporal dynamics are limited, cluster centers simplify into unclear patterns, leading to poor interpretability and weak predictive performance.

One notable exception is observed in the ExchangeRate dataset, where C-AE-AMS outperforms AE-AMS across all metrics despite flat centroids. Because these patterns closely resemble those of the Treasury dataset, we interpret this as an anomaly rather than evidence that flat prototypes are effective. A likely explanation is that ExchangeRate has smaller and more balanced RoCs across models, so clustering still preserves a reasonable diversity. In contrast, Treasury shows highly oversized and unbalanced RoCs, which results in non-representative centroids. More generally, datasets with low temporal variability are more affected by this issue, as redundant windows accumulate disproportionately in certain models, which results in unbalanced RoCs and flatter centroids.

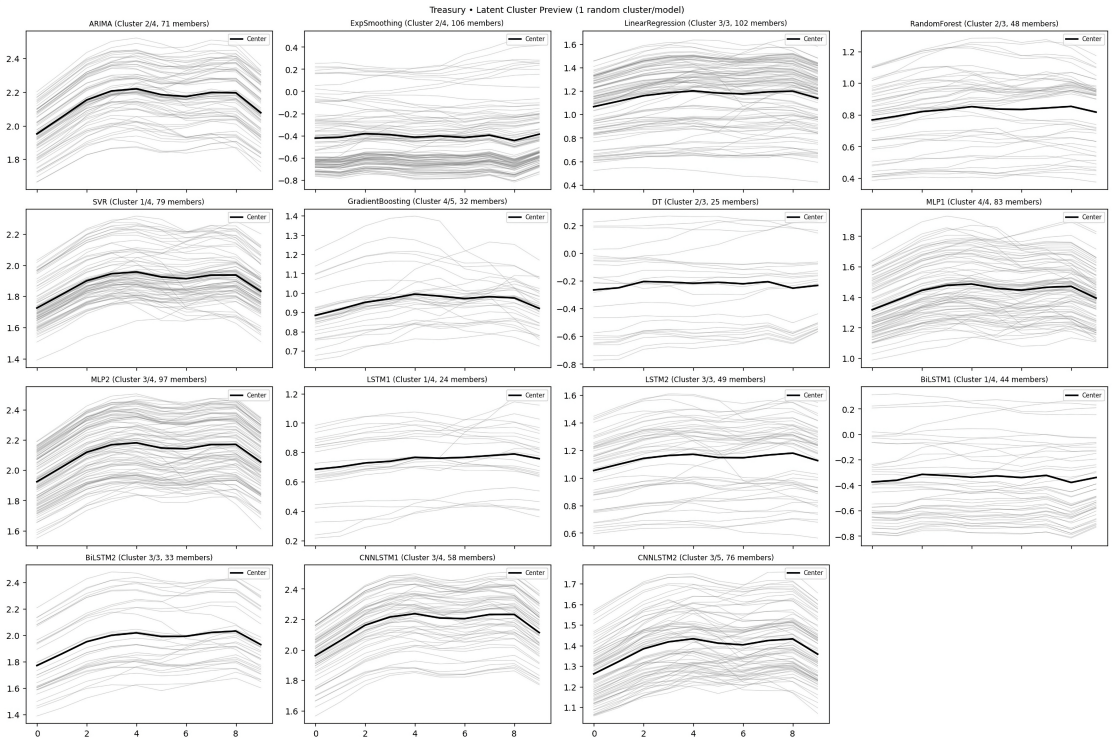

Figure 5.5.: Cluster previews of the RoCs for the Treasury dataset, separated by forecasting model. Each subplot shows one randomly selected cluster with its decoded center (bold) and members (faint). The flat structures illustrate the limited interpretability of clusters on this dataset.

To conclude our analysis of clustering, Figure 5.6 illustrates how the cluster centers can be used to explain why a particular model was selected for a given window. For the Births dataset, both centers follow the shape of the original window, giving a clear explanation for the model choice. In contrast, for the Treasury dataset, the centers and the window do not align, providing little explainability.

In the next experiment, we address **Q3** by comparing two variants of the Hoeffding-based drift detection mechanism described in section 4.7, one monitoring the mean of each window and the other monitoring its reconstruction error. The goal is to assess whether monitoring the reconstruction error enables more effective drift detection than monitoring the raw series mean. Specifically, we examine how frequently the detector triggers drift alarms and how much forecasting performance improves when the model adapts to the new distribution. Because drift is not present across all datasets, we restrict our analysis to three datasets where clear distributional changes occur. Moreover, since this experiment also involves a grid search over adaptation hyperparameters, restricting the datasets keeps computational costs manageable. For this experiment, all model selection is performed using cosine distance and a lower safety parameter for the mean-based detector $\gamma = 1.25$. A lower value was chosen to balance sensitivity such that both detectors trigger a comparable number of drift alarms.

Table 5.9 summarizes the results of the two drift detectors compared to a baseline without drift detection across the three datasets. On the Births dataset, both detectors improve forecasting performance relative

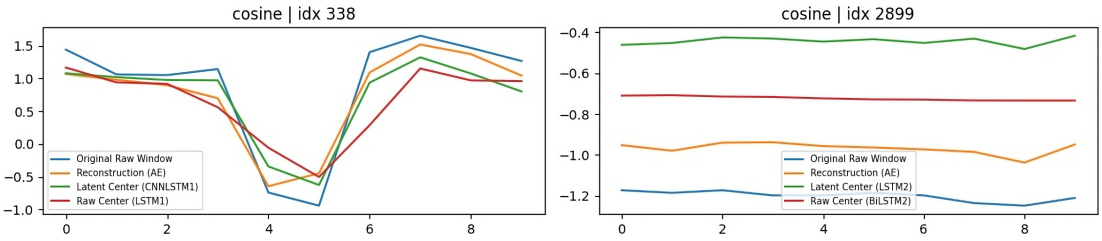

Figure 5.6.: Random test window (with its reconstruction) matched to the closest raw and latent centers using cosine distance. Example shown for Births (left) and Treasury (right).

to the baseline. The mean-based detector signals 4 drift points, while the reconstruction-error detector only triggers twice. This results in a slightly lower MSE for the mean variant but at a higher runtime cost. On the OfficeTemp dataset, both detectors trigger once, yet only the reconstruction-error detector results in a lower MSE compared to the baseline. The mean-based detector increases the error, suggesting a false or mistimed drift alarm. On the SolarPower dataset, the reconstruction-error detector remains inactive, while the mean-based detector triggers frequently (11 alarms), which lowers the forecasting error but introduces substantial computational cost. Overall, the reconstruction error provides more cautious and stable drift detection, whereas the mean-based detector is more sensitive and its effectiveness varies across datasets.

Table 5.9.: Comparison of drift detectors (cosine metric) across datasets.

Dataset	Detector	MSE	Runtime (s)	Drifts
Births	None	0.3607	50.50	0
	Recon Error	0.2552	**38.61**	2
	Mean	**0.2459**	60.86	**4**
OfficeTemp	None	0.1407	50.61	0
	Recon Error	**0.1324**	42.07	**1**
	Mean	0.1922	**28.82**	**1**
SolarPower	None	0.2365	**18.36**	0
	Recon Error	0.2365	18.75	0
	Mean	**0.2155**	71.76	**11**

We now take a closer look at the Births dataset by examining the raw series alongside the reconstruction error and MSE over time. As shown in Figure 5.7, the reconstruction error detector triggers two alarms, both aligned with shifts in the signal, leading to noticeable reductions in MSE after adaptation. Toward the end of the series, however, the detector remains inactive despite visible changes in pattern. This is due to the reconstruction error being too stable after adaptation to be statistically significant. The mean-based detector triggers 4 times, some of which don't correspond to clear shifts. This detector still achieves slightly better MSE than recon error, but at the cost of potentially more false positives and a

higher runtime.

For OfficeTemp, shown in Figure 5.8, the reconstruction-error detector raises a single alarm around $t = 750$, which matches with a clear dip in the signal and a pronounced peak in the reconstruction error. This timely adaptation results in improved forecasting performance, as shown in Table 5.9. As observed in the Births dataset, the reconstruction-error detector fails to detect the following shifts. In contrast, the mean-based detector triggers once at the very beginning of the series and fails to detect any structural changes.

In the SolarPower dataset, presented in Figure 5.9, the two detectors behave very differently. The reconstruction-error variant never triggers a drift signal. The autoencoder captures the underlying day–night cycle with good accuracy. Consequently, reconstruction error does not reflect the distributional shifts, and no drifts are signaled. On the other hand, the mean-based detector raises signals often, largely aligning with shifts and day–night changes visible in the signal. This translates into lowest MSE but at a substantial runtime cost due to repeated retraining.

Overall, the reconstruction-error detector performs better on datasets with clear structural breaks, while the mean-based detector shows higher sensitivity on very sharp recurring shifts. This suggests that the most suitable drift signal is dataset-dependent, with reconstruction error offering greater robustness when abrupt regime changes occur.

Finally, we also conduct a small robustness check to evaluate how different adaptation strategies affect forecasting and drift detection. We perform a grid search over the **adaptation set size**, **adaptation split** and whether to **append** or replace RoCs. The parameter ranges used for the evaluation are presented in Table 5.10. We only use the reconstruction-error detector and keep all other settings fixed as in the previous experiment. The full grid search results are provided in Table A.6.

The results show that using a small adaptation split (retraining percentage of 25%) consistently destabilizes the detector, producing excessive drift alarms without improving accuracy, whereas values of 50% or 75% are more stable. Similarly, very large adaptation set sizes such as 50% lead to repeated drift triggers, while 10% performs reasonably better and 25% being the most stable choice overall. On average, appending new windows to the old RoC increases the forecasting error. Because each input is compared directly against all stored RoC vectors rather than aggregated centroids, appending only amplifies noise, which increases both computational cost and results in weaker forecasting performance. It is also worth noting that for the SolarPower dataset, no parameter configuration resulted in a drift signal.

Parameter	**Range**
Adaptation set size	{0.50, 0.25, 0.10}
Adaptation split	{0.75, 0.50, 0.25}
Append option	{False, True}

Table 5.10.: Grid search parameters for RoC adaptation.

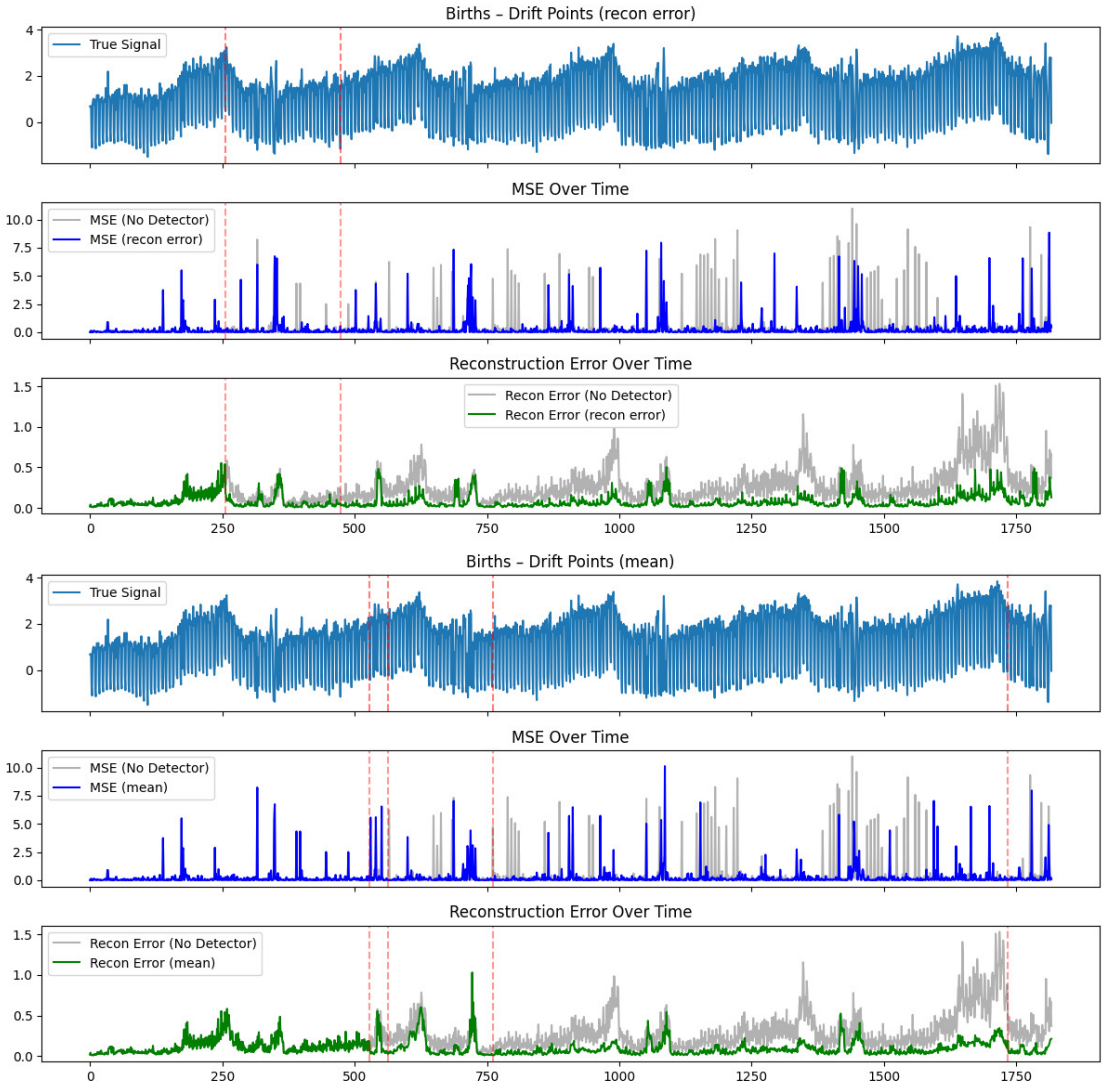

Figure 5.7.: True signal, forecasting errors (MSE), and reconstruction errors on the Births dataset. Gray shows results without drift detection; colored plots show results with the reconstruction-error detector (top) and mean-based detector (bottom).

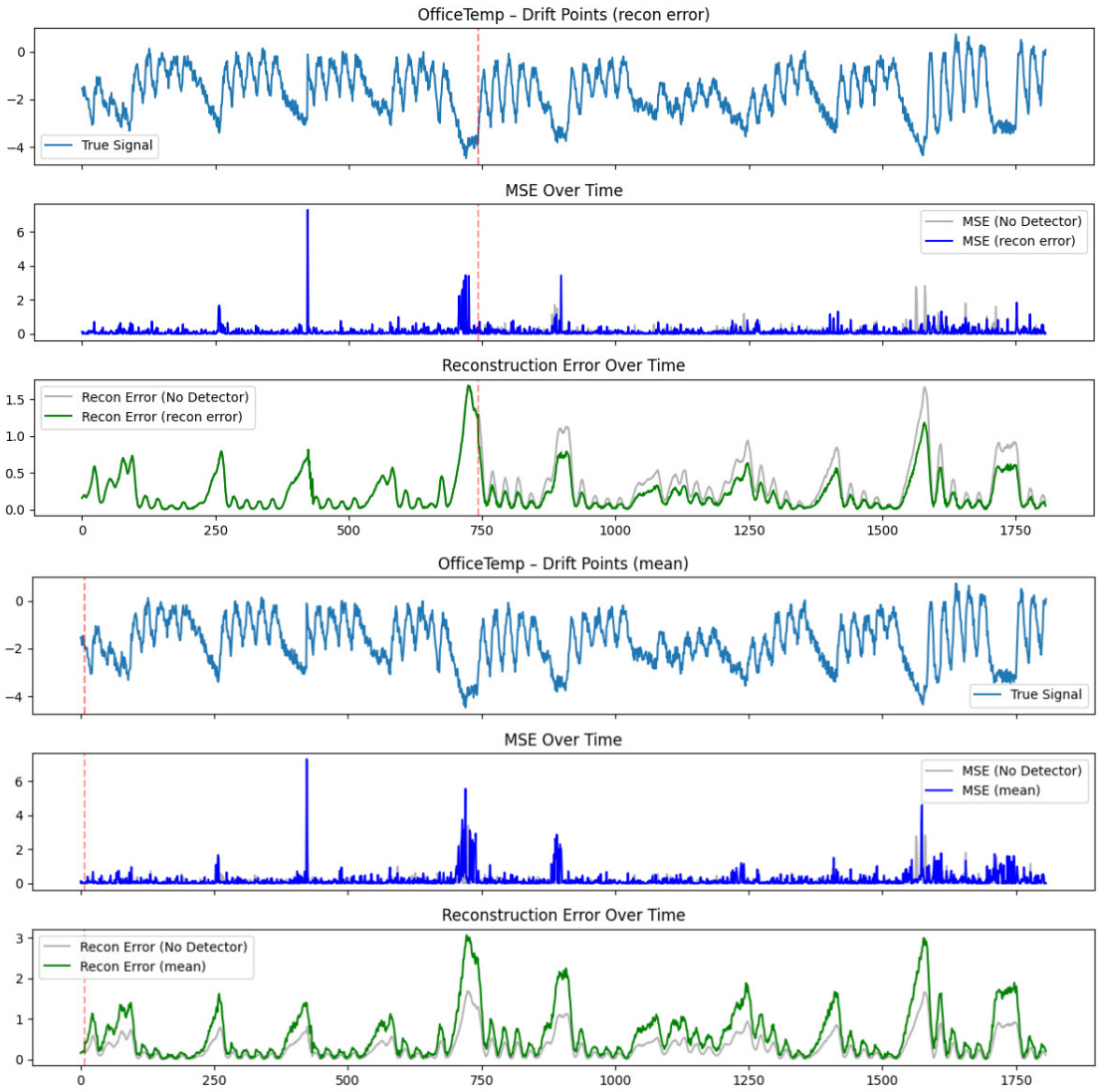

Figure 5.8.: True signal, forecasting errors (MSE), and reconstruction errors on the OfficeTemp dataset. Gray shows results without drift detection; colored plots show results with the reconstruction-error detector (top) and mean-based detector (bottom).

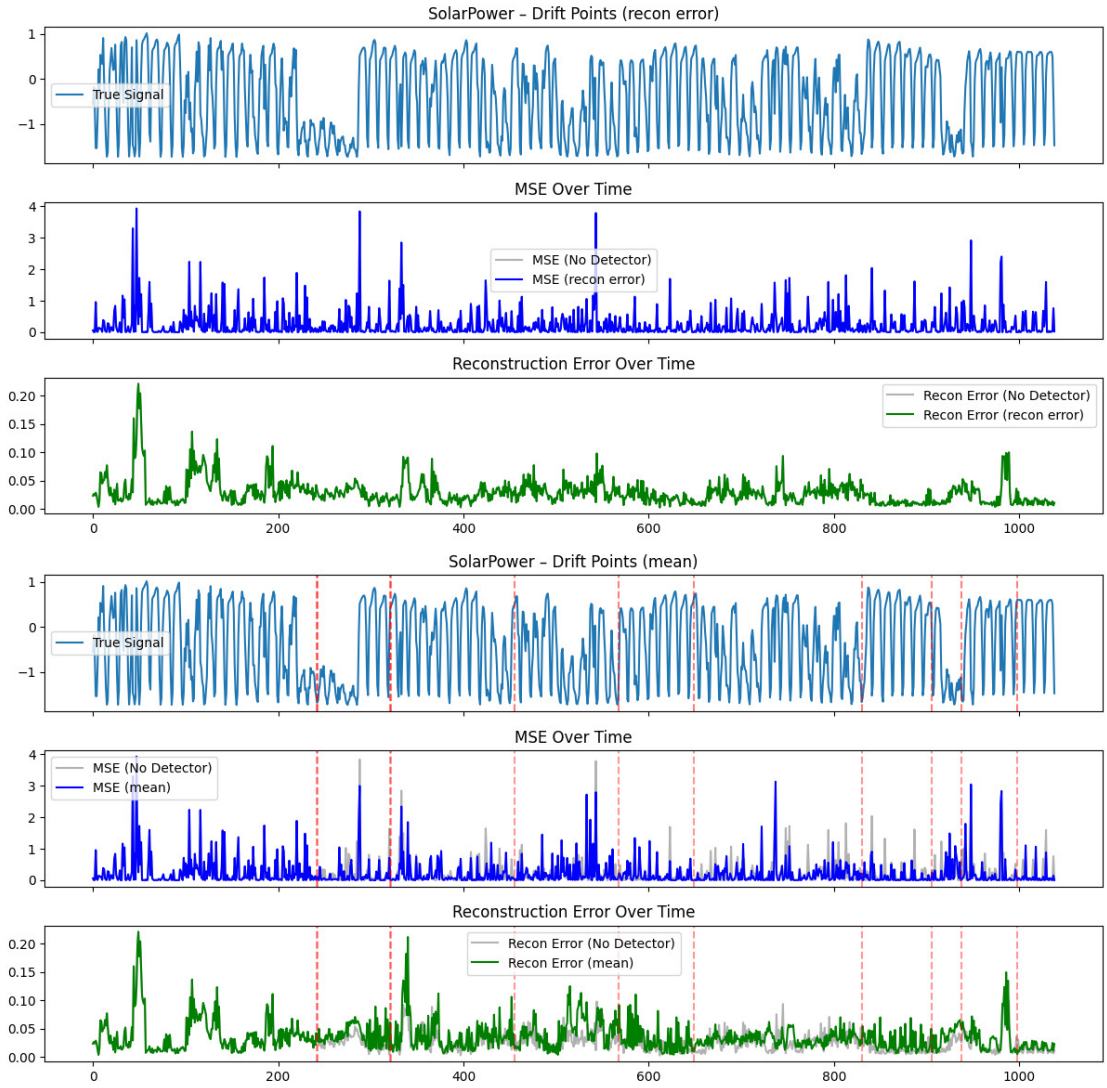

Figure 5.9.: True signal, forecasting errors (MSE), and reconstruction errors on the SolarPower dataset. Gray shows results without drift detection; colored plots show results with the reconstruction-error detector (top) and mean-based detector (bottom).

6. Further Discussion

This thesis introduced the AE-AMS framework for dynamic model selection and drift detection in the context of time series forecasting. We explored the use of autoencoder embeddings as the feature space in which regions of competence are defined and similarities are measured. Historical windows from a validation set are encoded using a convolutional autoencoder and assigned to forecasting models according to their performance. Online selection is then performed by comparing input windows to these regions in the latent space. We also gained deeper insights into the selection process and the competence of each forecasting model by clustering the regions of competence in the latent space and decoding the cluster centers back into the input space. Lastly, we investigated the use of reconstruction errors to update the regions of competence through unsupervised concept drift detection.

Our results show that learned representations, such as autoencoder embeddings, can provide an effective feature space for guiding dynamic model selection, leading to both performance gains and new interpretability opportunities. While our experiments focused on univariate time series, an important extension is to test the effectiveness of latent representations on multivariate time series, since they introduce complex cross-dependencies that latent spaces are well suited to capture. Furthermore, our evaluation was limited to eight datasets, with only three used for drift detection. Another limitation is that the range of the monitored statistic was not recalculated after drift detection for both detectors, which may have affected the accuracy of subsequent adaptations. Addressing these limitations is an important step toward validating the generalizability of our approach.

Exploring hybrid representations that combine raw time-domain similarity with learned latent spaces could make model selection even more robust and reliable, offering a more balanced trade-off between efficiency, accuracy, and robustness compared to either latent- or raw-only selection. Additionally, experimenting with other embedding methods, such as variational autoencoders with structured latent spaces or transformers with attention-based representations, presents a promising direction for future work to enhance the quality of the regions of competence and improve the identification of meaningful similarities between time series segments. Future research should also consider extending the framework to ensemble strategies by weighting each model using the similarities to the learned representations. By exploring these opportunities and addressing the challenges, representation learning has the potential to enhance established model selection strategies and offer a clear next step for dynamic model selection.

A. Appendix

Table A.1.: Statistical forecasters and key hyperparameters.

Name	Parameter	Value
ARIMA	order	(1, 0, 0)
ExpSmoothing	–	default

Table A.2.: Linear, kernel, and ensemble forecasters with key hyperparameters.

Name	Parameter	Value
LinearRegression	–	default
SVR	kernel	rbf
	C	0.5
	ϵ	0.05
DT	max_depth	3
	min_samples_split	3
	min_samples_leaf	2
RandomForest	n_estimators	50
	max_depth	3
	min_samples_split	4
	min_samples_leaf	2
GradientBoosting	n_estimators	50
	max_depth	2
	learning_rate	0.05

Table A.3.: Neural network forecasters and their hyperparameters.

Parameter	Model 1	Model 2
MLP		
hidden_layers	(8)	(16, 8)
activation	relu	relu
solver	adam	adam
learning_rate_init	10^{-3}	10^{-3}
max_iter	300	400
early_stopping	True	True
n_iter_no_change	10	10
LSTM		
hidden_size	16	32
num_layers	1	2
dropout	0.0	0.1
lr	10^{-3}	10^{-3}
epochs	30	30
batch_size	64	64
BiLSTM		
hidden_size	8	16
num_layers	1	2
dropout	0.0	0.1
lr	$3 \cdot 10^{-4}$	10^{-3}
epochs	30	30
batch_size	64	64
CNN-LSTM		
conv_channels	(8)	(16, 16)
kernel_size	3	3
lstm_hidden_size	16	32
lstm_num_layers	1	2
dropout	0.0	0.1
lr	10^{-3}	10^{-3}
epochs	30	30
batch_size	64	64

Table A.4.: Detailed comparison of RAW-AMS and AE-AMS across all datasets and metrics.

Dataset	Metric	Raw MSE	Latent MSE	Disagreement (%)	% Improvement
Temperatures	DTW	0.342	0.317	83.500	7.370
	Euclidean	0.337	0.322	65.560	4.450
	Cosine	0.330	0.332	75.080	−0.440
Births	DTW	0.485	0.350	68.020	27.830
	Euclidean	0.334	0.287	55.860	14.130
	Cosine	0.292	0.279	76.280	4.340
SolarPower	DTW	0.225	0.210	79.600	6.700
	Euclidean	0.219	0.200	61.790	8.580
	Cosine	0.237	0.236	74.010	0.550
WindPower	DTW	0.085	0.086	76.820	−0.900
	Euclidean	0.087	0.086	66.410	0.840
	Cosine	0.088	0.084	74.820	4.260
ExchangeRate	DTW	0.002	0.002	73.030	4.500
	Euclidean	0.002	0.002	52.290	−0.440
	Cosine	0.002	0.002	86.490	18.800
Treasury	DTW	0.004	0.004	64.510	−3.310
	Euclidean	0.004	0.004	48.120	−2.560
	Cosine	0.004	0.004	87.840	−7.340
Air Quality	DTW	0.123	0.106	73.560	13.610
	Euclidean	0.118	0.107	51.290	9.820
	Cosine	0.118	0.107	73.780	10.070
OfficeTemp	DTW	0.312	0.298	61.700	4.660
	Euclidean	0.309	0.308	44.050	0.120
	Cosine	0.234	0.181	79.750	22.480

Table A.5.: Detailed comparison of C-AE-AMS (cluster) against AE-AMS (latent) and RAW-AMS across all datasets and metrics.

Dataset	Metric	Cluster MSE	%Δ vs AE	%Δ vs RAW
Temperatures	DTW	0.331	−4.640	3.060
	Euclidean	0.332	−3.160	1.430
	Cosine	0.331	0.410	−0.030
Births	DTW	0.303	13.440	37.530
	Euclidean	0.311	−8.450	6.870
	Cosine	0.289	−3.430	1.050
SolarPower	DTW	0.234	−11.480	−4.010
	Euclidean	0.253	−26.140	−15.310
	Cosine	0.243	−3.070	−2.510
WindPower	DTW	0.088	−2.840	−3.770
	Euclidean	0.090	−4.070	−3.190
	Cosine	0.088	−5.100	−0.630
ExchangeRate	DTW	0.002	12.070	16.020
	Euclidean	0.002	9.660	9.260
	Cosine	0.002	4.210	22.220
Treasury	DTW	0.005	−33.850	−38.280
	Euclidean	0.005	−33.230	−36.650
	Cosine	0.005	−18.040	−26.710
Air Quality	DTW	0.141	−32.450	−14.420
	Euclidean	0.139	−29.900	−17.140
	Cosine	0.123	−15.020	−3.440
OfficeTemp	DTW	0.333	−11.780	−6.570
	Euclidean	0.332	−7.800	−7.670
	Cosine	0.242	−33.620	−3.580

Table A.6.: Grid search results for drift adaptation parameters. MSE is rounded to three decimals.

Dataset	Adapt. set size	Adapt. split	Append	σ	MSE	Drifts
Births	0.50	0.75	False	0.023	0.279	11
	0.50	0.75	True	0.023	0.308	9
	0.50	0.50	False	0.023	0.286	34
	0.50	0.50	True	0.023	0.285	15
	0.50	0.25	False	0.023	0.297	125
	0.50	0.25	True	0.023	0.304	153
	0.25	0.75	False	0.023	0.274	23
	0.25	0.75	True	0.023	0.282	9
	0.25	0.50	False	0.023	0.286	28
	0.25	0.50	True	0.023	0.359	1
	0.25	0.25	False	0.023	0.256	70
	0.25	0.25	True	0.023	0.313	52
	0.10	0.75	False	0.023	0.280	16
	0.10	0.75	True	0.023	0.327	30
	0.10	0.50	False	0.023	0.268	49
	0.10	0.50	True	0.023	0.249	31
	0.10	0.25	False	0.023	0.274	78
	0.10	0.25	True	0.023	0.295	138
OfficeTemp	0.50	0.75	False	0.023	0.142	4
	0.50	0.75	True	0.023	0.149	8
	0.50	0.50	False	0.023	0.150	43
	0.50	0.50	True	0.023	0.145	38
	0.50	0.25	False	0.023	0.136	17
	0.50	0.25	True	0.023	0.141	145
	0.25	0.75	False	0.023	0.131	2
	0.25	0.75	True	0.023	0.146	2
	0.25	0.50	False	0.023	0.131	1
	0.25	0.50	True	0.023	0.147	1
	0.25	0.25	False	0.023	0.157	5
	0.25	0.25	True	0.023	0.136	10
	0.10	0.75	False	0.023	0.172	2
	0.10	0.75	True	0.023	0.133	2
	0.10	0.50	False	0.023	0.132	4
	0.10	0.50	True	0.023	0.131	2
	0.10	0.25	False	0.023	0.133	13
	0.10	0.25	True	0.023	0.137	11

List of Figures

Acknowledgments

While writing this thesis, I used ChatGPT to improve the readability of the text and to suggest alternative structures for presenting certain sections. Its use was strictly linguistic and structural. All research content, analyses, and conclusions remain entirely my own.

I would like to sincerely thank Dr. Amal Saadallah for introducing me to this research topic and for her guidance throughout the thesis. Her enthusiasm for the subject made this experience both enriching and inspiring. I am profoundly grateful to my mother for her constant support throughout my bachelor studies, and to my friend Adam K. for always being there when I needed him.

Bibliography

[1] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. 1993. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms*, David B. Lomet (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 69–84.

[2] Sylvain Arlot and Alain Celisse. 2010. A survey of cross-validation procedures for model selection. *Statistics Surveys* 4 (2010), 40–79. https://doi.org/10.1214/09-SS054

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166.

[4] Donald J. Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (Seattle, WA) *(AAAIWS'94)*. AAAI Press, 359–370.

[5] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern Recognition and Machine Learning*. Springer, New York, NY.

[6] George E. P. Box and Gwilym M. Jenkins. 1976. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, CA.

[7] Paul Brabban. 2017. Daily Minimum Temperatures in Melbourne (1981–1990). https://www.kaggle.com/datasets/paulbrabban/daily-minimum-temperatures-in-melbourne Originally hosted on DataMarket.

[8] Leo Breiman. 2001. Random forests. *Machine Learning* 45, 1 (2001), 5–32. https://doi.org/10.1023/A:1010933404324

[9] Leo Breiman, J. H. Friedman, Richard A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Chapman and Hall/CRC. https://doi.org/10.1201/9781315139470

[10] Peter J. Brockwell and Richard A. Davis. 2016. *Introduction to Time Series and Forecasting* (3 ed.). Springer, New York, NY. https://doi.org/10.1007/978-3-319-29854-2

[11] Rafael M. O. Cruz, Robert Sabourin, and George D. C. Cavalcanti. 2018. Dynamic classifier selection: recent advances and perspectives. *Information Fusion* 41 (2018), 195–216. https://doi.org/10.1016/j.inffus.2017.09.010

[12] Saverio De Vito, Emanuele Massera, Matteo Piga, Luigi Martinotto, and Giulio Di Francia. 2008. Air Quality Dataset. https://archive.ics.uci.edu/dataset/360/air+quality UCI Machine Learning Repository. Retrieved August 26, 2025.

[13] Rick Durrett. 2019. *Probability: Theory and Examples* (5 ed.). Cambridge University Press, Cambridge. https://doi.org/10.1017/9781108591034

[14] European Central Bank. 2025. Exchange Rates: USD to EUR (EXR.D.USD.EUR.SP00.A). https://data.ecb.europa.eu/data/datasets/EXR/EXR.D.USD.EUR.SP00.A Retrieved August 26, 2025.

[15] Federal Reserve Bank of St. Louis. 2025. 10-Year Treasury Constant Maturity Minus 2-Year Treasury Constant Maturity [T10Y2Y]. https://fred.stlouisfed.org/series/T10Y2Y Retrieved August 26, 2025, from FRED, Federal Reserve Bank of St. Louis.

[16] David A. Freedman. 2009. *Statistical Models: Theory and Practice* (2 ed.). Cambridge University Press.

[17] Isvani Frías-Blanco, José del Campo-Ávila, Gustavo Ramos-Jiménez, Rafael Morales-Bueno, Ernesto Ortiz-Díaz, and Yailé Caballero-Mota. 2015. Online and non-parametric drift detection methods based on Hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering* 27, 3 (2015), 810–823. https://doi.org/10.1109/TKDE.2014.2345382

[18] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics* 29, 5 (2001), 1189–1232. https://doi.org/10.1214/aos/1013203451

[19] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with drift detection. In *Advances in Artificial Intelligence – SBIA 2004*. Springer, Berlin, Heidelberg, 286–295. https://doi.org/10.1007/978-3-540-28645-5_29

[20] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *Comput. Surveys* 46, 4 (2014), 44. https://doi.org/10.1145/2523813

[21] Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I. Webb, Rob J. Hyndman, and Pablo Montero-Manso. 2021. Monash Time Series Forecasting Archive. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, Joaquin Vanschoren and Sai-Kit Yeung (Eds.). https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/eddea82ad2755b24c4e168c5fc2ebd40-Abstract-round2.html

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press, Cambridge, MA. http://www.deeplearningbook.org/

[23] James D. Hamilton. 1994. *Time Series Analysis*. Princeton University Press, Princeton, NJ.

[24] Jiawei Han, Micheline Kamber, and Jian Pei. 2012. *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann, Waltham, MA.

[25] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507. https://doi.org/10.1126/science.1127647

[26] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[27] Charles C. Holt. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting* 20, 1 (2004), 5–10. https://doi.org/10.1016/j.ijforecast.2003.09.015

[28] Rob J. Hyndman and George Athanasopoulos. 2018. *Forecasting: Principles and Practice* (2 ed.). OTexts, Melbourne, Australia. https://otexts.com/fpp2/

[29] Maciej Jaworski, Leszek Rutkowski, and Plamen Angelov. 2020. Concept Drift Detection Using Autoencoders in Data Streams Processing. In *Artificial Intelligence and Soft Computing*, Leszek Rutkowski, Rafał Scherer, Marcin Korytkowski, Witold Pedrycz, Ryszard Tadeusiewicz, and Jacek M. Zurada (Eds.). Springer International Publishing, Cham, 124–133.

[30] David J. Ketchen and Christopher L. Shook. 1996. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic Management Journal* 17, 6 (1996), 441–458. https://doi.org/10.1002/(SICI)1097-0266(199606)17:6<441::AID-SMJ819>3.0.CO;2-G

[31] Alexander Lavin and Subutai Ahmad. 2015. The Numenta Anomaly Benchmark. In *Proceedings of the IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 38–44. https://doi.org/10.1109/ICMLA.2015.141

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[33] Bryan Lim and Stefan Zohren. 2021. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A* 379, 2194 (2021), 20200209. https://doi.org/10.1098/rsta.2020.0209

[34] J. MacQueen. 1967. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1. University of California Press, 281–297.

[35] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2018. The M4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting* 34, 4 (2018), 802–808. https://doi.org/10.1016/j.ijforecast.2018.06.001

[36] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. 2011. Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In *Artificial Neural Networks and Machine Learning – ICANN 2011*. Springer, Berlin, Heidelberg, 52–59. https://doi.org/10.1007/978-3-642-21735-7_7

[37] Florian Priebe. 2019. *Dynamic Model Selection for Automated Machine Learning in Time Series*. M.Sc. thesis. Technische Universität Dortmund, Dortmund, Germany.

[38] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–536. https://doi.org/10.1038/323533a0

[39] Amal Saadallah. 2023. Online Explainable Model Selection for Time Series Forecasting. In *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, Piscataway, NJ, USA, 1–10. https://doi.org/10.1109/DSAA60987.2023.10302609

[40] Amal Saadallah, Matthias Jakobs, and Katharina Morik. 2021. Explainable Online Deep Neural Network Selection Using Adaptive Saliency Maps for Time Series Forecasting. In *Machine Learning and Knowledge Discovery in Databases. Research Track*. Springer, 404–420. https://doi.org/10.1007/978-3-030-86486-6_25

[41] M. Schuster and K.K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681. https://doi.org/10.1109/78.650093

[42] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. 2015. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 28. 802–810.

[43] Robert H. Shumway and David S. Stoffer. 2017. *Time Series Analysis and Its Applications: With R Examples* (4 ed.). Springer, Cham. https://doi.org/10.1007/978-3-319-52452-8

[44] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. 2019. The Performance of LSTM and BiLSTM in Forecasting Time Series. In *2019 IEEE International Conference on Big Data (Big Data)*. 3285–3292. https://doi.org/10.1109/BigData47090.2019.9005997

[45] Alex J. Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Statistics and Computing* 14, 3 (2004), 199–222. https://doi.org/10.1023/B:STCO.0000035301.49549.88

[46] Jerome Friedman Trevor Hastie, Robert Tibshirani. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2 ed.). Springer, New York, NY. https://doi.org/10.1007/978-0-387-84858-7

[47] Alexey Tsymbal. 2004. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin* 106 (2004), 58.

[48] Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer, New York. https://doi.org/10.1007/978-1-4757-2440-0

[49] Peter R. Winters. 1960. Forecasting sales by exponentially weighted moving averages. *Management Science* 6, 3 (1960), 324–342. https://doi.org/10.1287/mnsc.6.3.324

[50] Qiuyan Xiang, Lingling Zi, Xin Cong, and Yan Wang. 2023. Concept Drift Adaptation Methods under the Deep Learning Framework: A Literature Review. *Applied Sciences* 13, 11 (2023). https://doi.org/10.3390/app13116515

[51] Indrė Žliobaitė. 2010. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784* (2010). https://doi.org/10.48550/arXiv.1010.4784

Eidesstattliche Versicherung

(Affidavit)

Attary, Ilias

Name, Vorname
(surname, first name)

238079

Matrikelnummer
(student ID number)

■ Bachelorarbeit
(Bachelor's thesis)

☐ Masterarbeit
(Master's thesis)

Titel
(Title)

Autoencoder Embeddings for Adaptive Model Selection and Drift Detection

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 25.08.2025

Ort, Datum
(place, date)

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 25.08.2025

Ort, Datum
(place, date)

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**