Unsupervised Time Series Anomaly Detection Using Time Series Foundational Models

by

Linh K. Nguyen

B.S. Artificial Intelligence and Decision Making & Physics, MIT, 2024

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2025

Authored by:	Linh K. Nguyen Master of Engineering in Electrical Engineering and Computer Science February 14, 2025
Certified by:	Kalyan Veeramachaneni Principal Research Scientist, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair Master of Engineering Thesis Committee

Unsupervised Time Series Anomaly Detection Using Time Series Foundational Models

by

Linh K. Nguyen

Submitted to the Department of Electrical Engineering and Computer Science
on February 14, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

The rapid generation of time series data across a wide array of domains—such as finance, healthcare, and industrial systems—has made anomaly detection a critical task for identifying irregular patterns that could signal significant events like fraud, system failures, or health crises. Traditional approaches to time series anomaly detection, including statistical models like ARIMA and deep learning methods, have proven effective but often require an extensive training phase, which can be both data and time-consuming.

In recent years, the emergence of foundational models, including large language models (LLMs) and specialized time series models, has opened up new possibilities for anomaly detection. These models, pre-trained on vast and diverse datasets, offer the potential to perform tasks with minimal task-specific training. This thesis investigates the feasibility of leveraging these foundational models for time series anomaly detection, with the aim of determining their effectiveness in detecting anomalies without the traditional training requirements. We also aim to investigate whether foundational models pretrained specifically on time series data yield better results compared to large language models (LLMs) that were not pretrained for time series tasks.

Thesis supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist

Acknowledgement

Chapter 4 and parts of chapter 5 are from joint work with Alnegheimish, Nguyen, Berti-Equille, *et al.* [1]. The SigLLM-Detector method was developed and benchmarked by Sarah Alnegheimish. To ensure that the thesis is self-contained and coherent, some notable details of its methodology and results from paper Alnegheimish, Nguyen, Berti-Equille, *et al.* [1] are reiterated in this thesis.

I would like to express my heartfelt gratitude to several individuals and groups who have played pivotal roles throughout the course of my research and the completion of this thesis.

First and foremost, I want to extend my deepest appreciation to my thesis advisor, Kalyan Veeramachaneni. His invaluable research advice and insightful feedback have significantly shaped my work and provided me with clarity and direction

I would like to acknowledge the profound impact of Sarah Alnegheimish's supervision. Her unwavering guidance and mentorship throughout this project have been truly remarkable. The insights she shared, coupled with her patience and encouragement, have profoundly influenced my academic journey and personal growth.

Additionally, I want to acknowledge my experience as a teaching assistant in the Introduction to Machine Learning (6.390) class. The insightful discussion with teaching staff in this course have laid a strong foundation for my understanding of machine learning principles, which has been crucial for my research.

I also want to thank the members of the DAI Lab for their support and collaboration. Their willingness to share knowledge and resources has been invaluable, and I appreciate the sense of community they provided throughout my studies.

Finally, I want to express my gratitude to my family and friends for their support and encouragement. I am fortunate to have such a supportive network that has continually inspired me to pursue my goals with confidence and determination.

In conclusion, I am deeply grateful to all of these individuals and groups for their contributions to my academic and personal journey. Each of them has had a lasting impact on my experience, and I look forward to carrying the lessons learned and the relationships forged into my future endeavors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the era of vast data generation across various domains, the ability to discern anomalies within time series data has become paramount. Time series anomaly detection (AD) serves as a critical tool for detecting irregular patterns, outliers, and deviations from expected behaviors, enabling proactive intervention and maintenance in numerous real-world scenarios. Whether it's monitoring sensor data in industrial settings, detecting fraudulent activities in financial transactions, or identifying anomalies in health monitoring systems, the significance of robust anomaly detection methodologies cannot be overstated.

The domain of anomaly detection has been extensively researched for decades [2]. A broad spectrum of Anomaly Detection (AD) methodologies has been explored, encompassing traditional statistical techniques such as distance-based [3], density-based [4], and isolation-based methods [5], as well as contemporary approaches rooted in machine learning and deep learning paradigms [6]–[9]. Statistical models such as ARIMA excel at modeling linear dependencies and seasonality in time series data, while deep learning methods, including recurrent neural networks (RNNs) and transformers, offer powerful tools for detecting complex, non-linear patterns. However, these methods typically require extensive training, which can be time-consuming and computationally expensive.

Recently, the rise of foundational models–large pre-trained models such as large language

models (LLMs) and time series-specific models (such as `TimesFM`[10], Lag-Llama [11], etc.)– has opened new possibilities in the field of machine learning. These models, trained on vast amounts of data across a variety of domains, have demonstrated an impressive ability to generalize and perform tasks with little to no task-specific training. The potential of such models in time series anomaly detection is an area of growing interest, as they promise to bypass the need for extensive task-specific training while potentially offering superior performance across diverse datasets.

This thesis aims to explore the feasibility of leveraging these foundational models for time series anomaly detection. Specifically, it investigates whether these models can detect anomalies in time series data effectively, without requiring traditional training workflows or fine-tuning.

1.1 Contribution

This thesis aims to investigate the utilization of foundational models to directly detect anomalies in time-series sequences without the need for a resource-intensive training phase. This thesis aims to investigate the potential of two classes of foundational models in anomaly detection:

1. **Time-series Foundational Models**: We explore how foundational models for time series, pretrained on extensive datasets, can effectively identify anomalies in previously unseen data by capturing the underlying time series patterns and detecting deviations from these established norms. We experimented with two time series foundational models `UniTS` and `TimesFM`.

2. **Large Language Models (LLMs)**: We investigate how LLMs can be directly utilized to detect anomalies in time series data by learning normal patterns and identifying deviations from them. This includes exploring different LLM architectures, prompting, and post processing strategies to improve anomaly detection performance. The two

LLMs we experimented with are GPT and MISTRAL.

At the end of this thesis, we also aim to answer the question: do foundational models developed using time series provide better results than the LLMs which were not trained for the time series tasks.

1.2 Thesis Organization

The thesis is structured as follows. Chapter 2 provides a comprehensive review of relevant literature on time series and anomalies, various anomaly detection methods, and an in-depth examination of transformers and foundational models.

Subsequent chapters build upon this foundation, starting with Chapter 3, which focuses on detecting anomalies in signals using time series foundational models. This includes the detailed implemention of *Orion* pretrained pipeline. Chapter 4 extends this discourse to the use of Large Language Models (LLMs) for anomaly detection, detailing methodologies that include anomaly detection through prompting and forecasting. Chapter 5 presents the evaluation of the proposed methods, encompassing dataset descriptions, baseline models, metrics employed for assessment, and a detailed discussion of results. The thesis concludes in Chapter 6 by synthesizing findings, reflecting on implications, and suggesting areas for future research.

Chapter 2

Background and Related Work

2.1 Timeseries and Anomalies

2.1.1 Timeseries Format

In this thesis, a univariate/multivariate time series is represented by a set of integers denoting timestamps, and a/multiple sets of real values observed at each respective timestamps. These values can be of varying numerical magnitudes, which may include both positive and negative values. Below are examples of univariate and multivariate time series.

Table 2.1: Univariate Timeseries where value represents the value of time series.

timestamp	value
1222819200	210
1222840800	125
⋮	⋮
1334905600	400

Table 2.2: Multivariate Timeseries where $v_1, \ldots, v_k$ are k time series and these columns have values for those.

timestamp	v_1	...	v_k
1222819200	210		130
1222840800	125		50
⋮	⋮	...	⋮
1334905600	400		80

2.1.2 Anomalies

In the context of time series data, an anomaly refers to an observation that deviates significantly from the expected pattern or behavior of the data. Even though there are many types of anomalies depending on the sources and domains of the time series data, they can be classified into two broad categories [12]:

- **Point anomalies** (outliers) are abrupt spikes or drops within the time series, identifiable by a single timestamp. When a sequence of consecutive point anomalies occurs, they are termed collective anomalies, and represented by a timestamp interval. Statistical-based outlier detection methods are commonly used to detect these anomalies [13].
- **Contextual anomalies** refer to values that deviate from the expected context within a time series, often appearing as irregular patterns. These anomalies are identified by an interval, defined by both start and end timestamps.

2.2 Anomaly Detection Methods

Over the years, numerous successful anomaly detection methods have emerged, each developed to address the growing need for identifying unusual patterns in various types of data. These methods have evolved significantly, from statistics-based to more advanced machine learning methods.

2.2.1 Statistical Methods

For a long time, researchers have developed methods to systematically detect anomalous sequences in time series data. One of the simplest techniques is static thresholding, which triggers an alert when a data point surpasses the predefined range. Nonetheless, this method often struggles to identify contextual anomalies. To enhance thresholding, various statistical

techniques have been introduced, including Statistical Process Control (SPC)[14], where data points are flagged as anomalies if they do not meet the criteria of statistical hypothesis testing. However, this approach still relies on human expertise to establish prior assumptions.

Another prevalent technique involves utilizing regression models to break down a time series into its underlying components: trend, seasonality, and residuals, with the aim of identifying anomalies within the residuals. This methodology was exemplified in the Autoregressive Moving Average (ARMA) approach [15]. Building on this concept, the Autoregressive Integrated Moving Average (ARIMA) model [16] endeavors to estimate these residuals while effectively addressing the issue of non-stationarity in the data. By incorporating differencing to stabilize the mean of the time series, ARIMA enhances the model's ability to capture underlying patterns and improve anomaly detection performance.

2.2.2 Machine Learning Methods

The recent rise of machine learning techniques has led to the development of additional approaches for anomaly detection that use deep-learning models. Deep-learning anomaly detection methods can be broadly categorized into two kinds: **prediction-based** and **reconstruction-based**.

Prediction-based methods generally involve training a model to forecast future values within the time series using sliding window segments and detect anomalies based on defined deviations from the observed values [17]. A widely recognized architecture for prediction-based anomaly detection is the Long Short-Term Memory Network with Dynamic Thresholding (LSTM-DT) [18]. In addition to LSTM-DT, other prominent methods include Hierarchical Temporal Memory and Bayesian Networks [19]

On the other hand, in **reconstruction-based** methods, a deep learning model, such as recurrent neural networks (RNNs) is trained to recognize a pattern sequence and use an estimator to forecast the expected values. Then, we can detect the anomalies based on the discrepancies between the forecasted signal and the real one [20]. Several promi-

nent **reconstrunction-based** anomaly detection methods include: LSTM Auto-Encoders (LSTM-AE) [21], LSTM Variational Auto-Encoders (LSTM-VAE) [22], and time-series anomaly detection Generative Adversarial Networks (Tad-GAN) [23].

Overall, the core idea behind deep learning models—whether prediction-based or reconstruction-based—is to produce an "expected" signal that reflects the original signal's pattern without anomalies. This process is illustrated in Figure 2.1. Ultimately, this approach generates a sequence of "errors" for each time point, indicating the likelihood of that point being an anomaly.

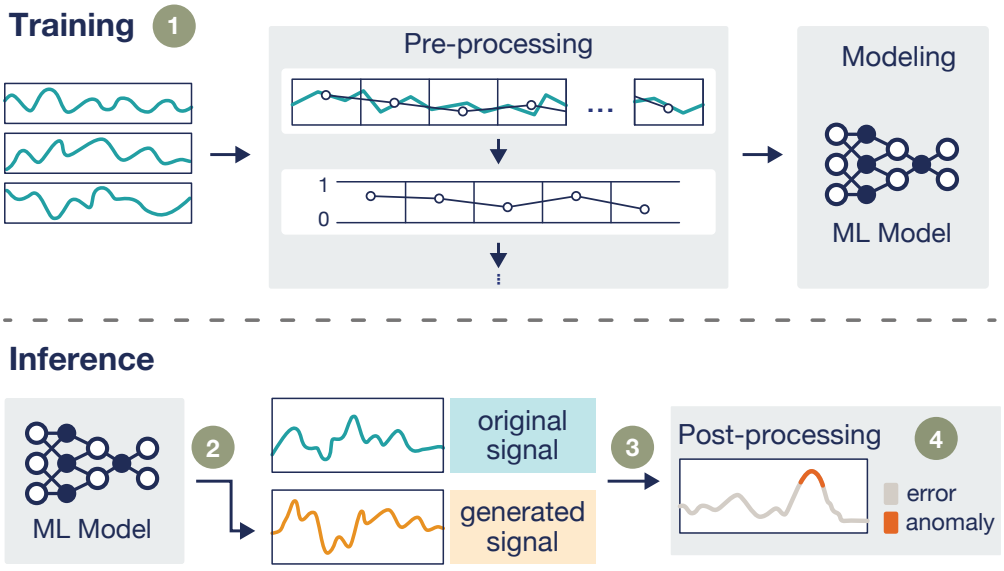

Figure 2.1: General principle of how machine learning models find anomalies in an unsupervised setting. Step 1: Apply a sequence of preprocessing operations and train a machine learning model to learn the pattern of the data. This is the most time-consuming step; Step 2: Use the trained model to generate another time series; Step 3: Quantify the error between what the model expects and the original time series value; Step 4: Use this discrepancy to extract anomalies.

Alnegheimish [12] detailed *Orion* - a framework for constructing anomaly detection pipeline. A previous thesis Song [24] have provided an overview of this framework. For readers' convenience, we included an adapted section from Song [24] on *Orion* overview in Section A.1

2.3 Transformer and Foundational Models.

In the recent years, the emergence of foundational models has significantly transformed the landscape of artificial intelligence (AI) and machine learning (ML). These models are designed to be pre-trained on large datasets and subsequently used for inference on new data across different domains, not limited to natural language processing (NLP) but including computer vision, audio processing, and time series analysis. Among these foundational models, the transformer architecture represents a pivotal advancement, especially in handling sequential data.

2.3.1 The Transformer Architecture

Introduced by Vaswani et al. [25], the transformer model uses a unique self-attention mechanism that enables it to process input data in parallel, overcoming the limitations of recurrent neural networks (RNNs). Key features of the transformer include:

- **Self-Attention**: This mechanism allows the model to focus on relevant parts of the input sequence when making predictions, capturing complex dependencies and contextual relationships with greater accuracy.
- **Positional Encoding**: Transformers incorporate positional encoding to retain the order of sequences, a crucial aspect for tasks where the arrangement of inputs matters.
- **Multi-Head Attention**: By having multiple attention heads, transformers can learn diverse representations and capture various features from the input simultaneously.
- **Feed-Forward Networks and Layers**: These contribute to the model's depth and capacity, processing attention outputs for further non-linear transformations.
- **Residual Connections**: Enhancing training stability, these connections help transfer gradients effectively, addressing the challenges of deep network training.

2.3.2 Foundational Models

Foundational models refer to a broad category of models that are generally large, pre-trained, and versatile. They can be adapted for different tasks, such as language processing, audio processing, time series analysis, etc.

Foundational models are typically trained on a large collection of sequences, $U = U_1, U_2, ...U_i, ..., U_N$, where $U_i = (u_1, u_2, ..., u_j, ..., u_{ni})$ and each token, u_i, belongs to vocabulary set V. Foundational models encode an autoregressive distribution, in which the probability of each token is only dependent on the previous tokens in the sequence, $p_\theta(U_i) = \prod_{n_i} p_\theta(u_j|u_{0:j1})$. The parameters, θ, are learned by maximizing the probability of the entire dataset set sequences $p_\theta(\mathcal{U}) = \prod_{i=1}^{N} p_\theta(U_i)$. Thus, in the inference phase, the next token is directly sampled from the learned conditional probability distribution.

The most important class of foundational models is Large Language Model (LLM). It has been observed that LLMs, such as GPT [26] or LLAMA-2 [27], exhibit "emergent abilities," which stem from the intricate interplay among the model's components and are not directly programmed or designed [28]. The most intriguing among "emergent abilities" is *in-context learning* [29], in which a model (through example demonstration, *few-shot learning*, or no example, *zero-shot learning* [30]) can perform a text-formatted task without training the model parameters on any task-specific data. Zero-shot and few-shot learning abilities enhance the versatility and adaptability of LLMs, making them valuable for a wide range of applications and domains where rapid adaptation and generalization are necessary.

2.4 Related work

2.4.1 Transformers for timeseries

The transformer's self-attention mechanism and ability to capture long-range dependencies make it appealing to time series modeling tasks. These capabilities enable the model to focus

on relevant historical data points and identify essential trends, making it especially effective for complex time-dependent data. For instance, the Temporal Fusion Transformer (TFT) combines attention mechanisms with recurrent structures to enhance multivariate time series forecasting while maintaining interpretability [31].

Additionally, the ability of transformers to analyze long sequences makes them adept at representation learning of multivariate time series, which can then be leveraged for downstream tasks such as reconstructing, forecasting and classification [32], [33].

The advantages of using transformers in time series tasks include scalability, flexibility, and state-of-the-art performances. Comprehensive studies such as Wen, Zhou, Zhang, *et al.* [34] illustrate the potentials and limitations of these models. As the field advances, the potential for transformers to address increasingly complex time series challenges continues to grow, highlighting the need for ongoing research and exploration in this direction.

Recently, there have been a significant number of foundational time series models being released. Like the LLMs, foundational time-series models are also pretrained using the transformer architecture, but instead of being trained on the language data, they were trained on a large corpus of time-series data. ForecastPFN [35] pre-trains a basic encoder-decoder transformer with one multi-head attention layer and two feedforward layers on a synthetically generated time series dataset. Similarly, TimeGPT was pretrained on a large collection of publicly available time series datasets. Lag-Llama [11] is a decoder-only Llama-2 model pretrained on a large corpus of real time series data from diverse domains. Moreover, Chronos [36] adopts a T5 architecture, and parses time series data into text to pretrain their model. Most of these models were developed with the objective of creating a time series foundation model for time series forecasting.

In this research, we study the two foundational time-series models `UniTS` [37] and `TimesFM` [10]. We especially focus on their ability to detect anomaly on new data that wasn't included in their training corpus.

2.4.2 LLM for timeseries

Recently, there has been significant interest in leveraging LLMs within the realm of time series analysis. Due to the similarity between predicting the next word in a sentence and predicting the next value in a time series, much attention has been directed towards time series forecasting. Among the noteworthy initiatives is LMMTime [38], which utilizes GPT [26], and Llama-2 [27] models to forecast time series data in a zero-shot manner. Another notable contribution is PromptCast [39], which introduces a prompt-based approach to forecasting, reframing the task as a question-answering problem.

In this research, we work strictly with two LLMs that have been pre-trained on text: GPT-3.5 [26] and an open source model using Mistral [40].

Chapter 3

Anomaly Detection using Time Series Foundational Models

We introduce a new class of anomaly detection pipeline in *Orion*: the pretrained pipeline. This pipeline leverages the capabilities of foundational time series models for forecasting, enabling us to avoid the traditional training phase, which is often time-consuming and resource-intensive. For instance, the deep learning model TadGAN typically takes an average of 180 seconds to both train and perform inference on each of the 80 NASA signals. More importantly, using pretrained model is more scalable since no training is required for specific signals. In other words, we do not have to train a model for different signals from different applications.

In this chapter, we will provide implementation details and practical usage examples of the pretrained pipeline.

3.1 Models

The two open-access foundational time series models we explored are `UniTS`[37] and `TimesFm`[10]. The datasets that were used to pretrained these models is detailed in section A.2 of Appendix.

3.1.1 `UniTS`

`UniTS`, introduced by Gao, Koker, Queen, *et al.* [37], offers a single framework that unifies predictive and generative time series tasks using task tokenization. The author provided the code and datasets for pretraining on the Github repository https://github.com/mims-harvard/UniTS?tab=readme-ov-file. We used the provided code and datasets to pretrain the model, then saved the model checkpoint for later inference.

3.1.2 `TimesFM`

`TimesFM` is a decoder-only foundation model for time-series forecasting that achieves near state-of-the-art zero-shot accuracy across diverse datasets, introduced by Das, Kong, Sen, *et al.* [10]. We did not need to pretrain `TimesFM` model since the model checkpoint is available at https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2. We downloaded the checkpoint file, then used it for inference. The authors also provided the code and datasets on the Github repository https://github.com/google-research/timesfm for pretraining the model.

3.2 *Orion* pretrained pipeline

3.2.1 High-level overview

As depicted in Figure 2.1, the first step in a typical machine learning (ML) pipeline involves training an ML model on a collection of time series data. However, since we utilize a pretrained time series model that is capable of forecasting without a formal training phase, we can transition directly to the inference phase.

Given a univariate time series $\mathbf{X} = (x_1, x_2, \ldots, x_T)$, we first segment each time series into rolling windows characterized by predetermined lengths w and step size of 1; i.e., the time series $\mathbf{X}$ is segmented and turned into a set $\{(x^i_{1\ldots w})\}_{i=1}^{N}$, where w is the window size and N is the number of windows ($N = T - w$). Here, we assume that anomalies do not happens

early in the time series, so we can ignore the first window.

For each given window ($x^i_{1...w}$), we query the Time Series pretrained model to predict the next value $\hat{x}^i_{w+1}$. The reconstructed timeseries $\hat{\mathbf{X}} = (\hat{x}_{w+1}, \hat{x}_{w+2}, \ldots, \hat{x}_T)$ have length of $T - w$.

We next compute the discrepancy between $\mathbf{X}$ and $\hat{\mathbf{X}}$. A large discrepancy indicates the presence of an anomaly. We denote this discrepancy as an error signal e by computing point-wise residuals, given their simplicity and ease of interpretation. We explore the usage of absolute difference suggested by Hundman et al. [18] $e_t = |x_t - \hat{x}_t|$. Moreover, we explore how other functions, such as squared difference $e_t = (x_t - \hat{x}_t)^2$ will help reveal the location of anomalies. More complex functions that capture the difference between two signals, such as dynamic time warping [41] can be used. However, Geiger, Liu, Alnegheimish, *et al.* [23] shows that discrepancies found with absolute difference function are sufficient for this purpose.

Moreover, we apply an exponentially weighted moving average to reduce the sensitivity of the detection algorithm [18]. Error values that surpass the threshold are considered anomalous. We use a sliding window approach to compute the threshold to help reveal contextual anomalies that are abnormal compared to the local neighborhood. As such, we assign the window size and step size to $T/3$ and $T/10$ respectively. We set a static threshold for each sliding window as four standard deviations away from the mean. These hyperparameters were chosen based on preliminary empirical results that agree with previous settings in other approaches [18], [23], [42].

3.2.2 Primitives and pipelines

The *Orion* pretrained pipeline consists of eight primitives:

`time_segments_aggregate`

This primitive creates an equi-spaced time series by aggregating values over fixed specified interval.

- **Input:** `x` which is an 1-dimensional sequence of values.
- **Output:**
  - `x` sequence of aggregated values, one column for each aggregation method.
  - `index` sequence of index values.

`SimpleImputer`

This `sklearn` primitive is an imputation transformer for filling missing values. It replaces missing values using a chosen statistic (mean, median, or most frequent), or using a constant.

- **Input:** `x` which is an 1-dimensional sequence of values.
- **Output:** `x` which is a transformed version of **Input**.

`StandardScaler`

This `sklearn` standardize features by removing the mean and scaling to unit variance.

- **Input:** `x` which is an 1-dimensional sequence of values.
- **Output:** `x` which is a transformed version of **Input**.

`rolling_window_sequences`

This primitive generates many sub-sequences of the original sequence. It uses a rolling window approach to create the sub-sequences out of time series data.

- **Input:**
  - `x` sequence to iterate over.
  - `index` array containing the index values of `x`.
- **Output:**

- `x` array of rolling windows from the inputted sequence.
- `y` target sequences.
- `index` first index value of each rolling window.
- `target_index` first index value of each target sequence.

`TimesFM` or `UniTS`

This primitive is used to forecast using Time Series pretrained model (`TimesFM` or `UniTS`).

- **Input:** `x` n-dimensional array containing the input sequences in rolling windows.
- **Output:** $\hat{y}$ predicted series.

`regression_errors`

This primitive computes an array of errors comparing the prediction and expected output.

- **Input:**
  - `y` ground truth.
  - $\hat{y}$ predicted values.
- **Output:** `errors` array of errors.

`find_anomalies`

This primitive finds anomalous intervals and its score from sequence of errors

- **Input:**
  - `errors` array of errors
  - `target_index` indices of the sequence
- **Output:** `anomalies` array containing start-timestamp, end-timestamp, and score for each anomalous sequence.

The `Orion` pretrained pipeline is shown in Figure 3.2. Here is a breakdown of how the input and output dimensions change through each step of the pipeline:

1. Input: `x` a time series of length `T`.
2. `time_segments_aggregate` will transform `x` to be of length `T'`. In the case where `x` is already regularly sampled `T` = `T'`. For easier notation, we assume that `x` is regularly sampled.
3. `SimpleImputer` will produce an imputed version of `x` and is of length `T`.
4. `StandardScaler` will produce a scaled version of `x` and is of length `T`.
5. `rolling_window_sequences` will create `x` windows of dimension (`N`, `w`, 1), where `N` = `T-w`. Here we always assume `step_size` = 1. however we can also work with various step sizes. In addition, the primitive will construct `y` of dimension (`N`, 1).
6. `TimesFM or UniTS` will predict next value $\hat{y}$ which is of dimension (`N`, 1) because we only forecast for one single value.
7. `regression_errors` will also produce `errors` of dimension `N`.
8. `find_anomalies` will generate `anomalies` of form $(t_s, t_e)^n$ where t_s and t_e are start and end timestamp of each anomalous interval respectively, and now we have n detected anomalous intervals.

Users can either use the default hyperparameter values or customize them according to their needs. Here is an example of how to use the `TimesFM` pipeline, the usage for `UniTS` pipeline is similar:

```
from orion.data import load_signal
from mlblocks import MLPipeline
#load signal and pipeline
data = load_signal("exchange-2_cpm_results")
pipeline = MLPipeline("timesfm")
#set hyperparameters
hyperparameters = {
    "mlstars.custom.timeseries_preprocessing.time_segments_aggregate#1": {
        "interval": 3600
    }
}
pipeline.set_hyperparameters(hyperparameters)
#detect anomalies
context = pipeline.fit(data)
context["anomalies"] #the detected anomalies
```

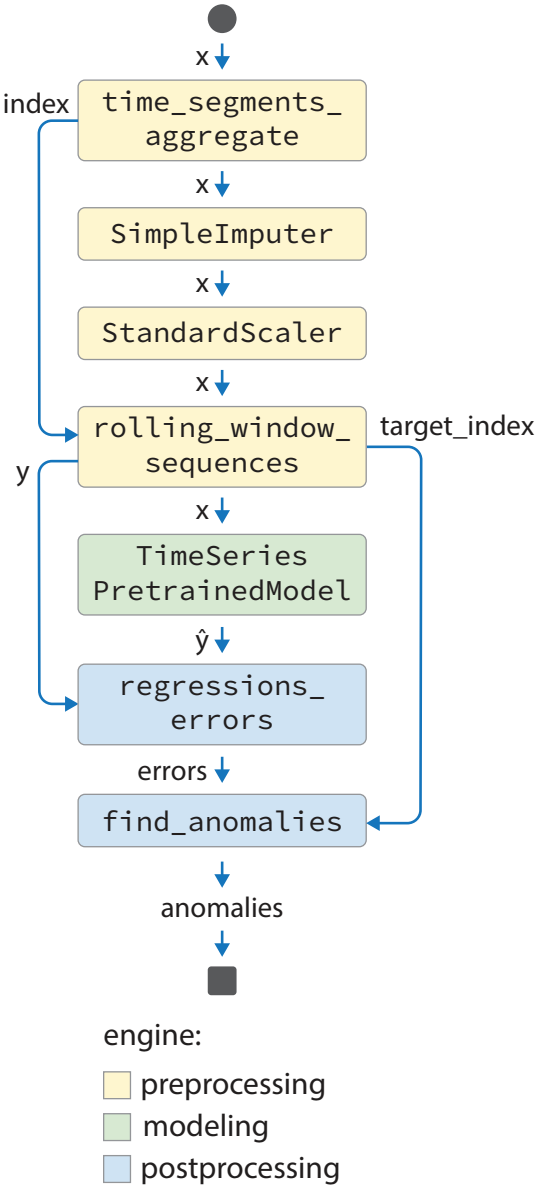

Figure 3.1: Graphic representation of an *Orion* pipeline with a pretrained TimeSeries model. Output format and structure for each data processing block here are explained in text before.

Chapter 4

Anomaly Detection using Large Language Models

This chapter, except section 4.4, is from joint work with Alnegheimish, Nguyen, Berti-Equille, et al. *[1].*

As mentioned in Chapter 2, there has been a growing interest in exploring the potential of Large Language Models (LLMs) for time series analysis. Their autoregressive capabilities have been demonstrated to be suitable for time series forecasting [38]. This leads to the question: can LLMs tackle more complex tasks such as anomaly detection?

4.1 Problem Statement

In this chapter, we work strictly with LLMs that have been pre-trained on text, particularly a proprietary model using GPT-3.5 [26] and an open source model using MISTRAL [40]. Our main objective is to determine whether LLMs have the ability to directly uncover anomalies in time series data. Referring back to Figure 2.1, our methodology focuses on Step 2 onwards – primarily the inference phase. To our knowledge at the time of working Alnegheimish, Nguyen, Berti-Equille, *et al.* [1], there is no other work that utilizes large language models as zero-shot anomaly detectors for time series data. We explore two avenues for accomplishing

this task: (a) Through the paradigm of prompt engineering; (b) By leveraging LLMs' ability to forecast time series in *zero-shot* without any additional data or fine-tuning.

Given that LLMs process strings rather than numerical values, we opted not to use *Orion*. Instead, we developed a new framework called SigLLM, which extends the principles of *Orion* to work effectively with LLMs. This framework incorporates both pre-processing and post-processing primitives that facilitate the conversion between numbers and strings. SigLLM consists of end-to-end pipelines that leverage LLMs for time series anomaly detection. In the following sections, we will discuss both the high-level overview and the algorithmic implementation details of the SigLLM framework.

4.2 Time Series representation

Time series data can take many different forms. In this thesis, we define a univariate time series as $\mathbf{X} = (x_1, x_2, \ldots, x_T)$, where $x_t \in \mathbb{Z}_{\geq 0}$ is the value at time step t, and T is the length of the series. To make a time series LLM-ready, we transform the univariate time series $\mathbf{X}$ into a sequence of values that is tokenized. We follow a sequence of reversible steps, beginning with scaling, quantization, and processing the time series into segments using rolling windows, and ending with tokenizing each window. We detail these steps below.

Scaling. Time series data includes values of varying numerical magnitudes, and may include both positive and negative values. To standardize the representation and optimize computational efficiency, we subtract the minimum value from the time series $x_{s_t} = x_t - min(x_1, x_2, \ldots, x_T)$, resulting in a new time series $\mathbf{X}_s = (x_{s_1}, x_{s_2}, \ldots, x_{s_T})$, where $x_{s_t} \in \mathbb{R}_{\geq 0}$. In other words, we introduced a mapping function: $\mathcal{E} : \mathbb{R} \to \mathbb{R}_{\geq 0}$. This eliminates the need to handle negative values separately.

Other scaling methods, such as min-max scaling, can be utilized to achieve the same goal. However, reducing the set of possible values to a smaller range (e.g. [0, 1]), may cause a loss of information in the quantization step. On the other hand, increasing the range will mean

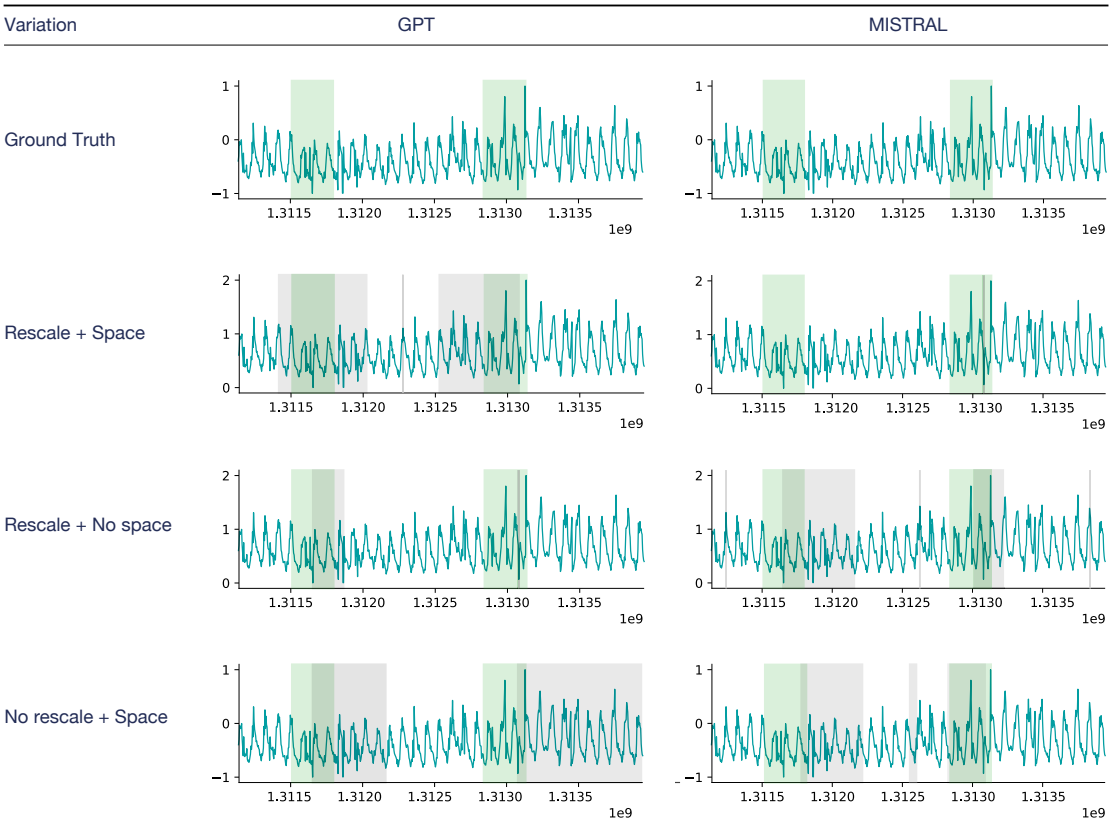

Figure 4.1: Visualizing the output of large language models (GPT and MISTRAL) under different variations of the transformation process. Each row depicts the `exchange-2_cpm_results` signal from the AdEx dataset, where the x-axis shows the timestamp and the y-axis is the signal value. The first row indicates the ground truth anomalies present in the time series (highlighted in green). The remaining rows indicate whether scaling and inserting space between digits has occurred during the conversion from signal to text. The gray intervals highlight the anomalies detected under these conditions; thus, we would like to maximize the overlap between the green and gray intervals. Overall we find that "scaling + space" is the configuration that yields a better output for GPT; and "scaling + no space" is better for MISTRAL.

there are more digits to tokenize. With our approach, we simply shift the range of the signal values, which allows us to reduce the number of individual digits that need to be tokenized while maintaining the original gaps between pairs of entries. Moreover, by projecting the values into a non-negative range, we eliminate the need for sign indicator "-/+" and save an additional token.

Quantization. Unlike the finite set of vocabulary words used to train LLMs (32k vocab tokens for MISTRAL) [1], the set of scaled time series values x_{s_t} is infinite, and cannot be processed by language models. Therefore, time series that are to be used with LLMs are generally quantized [36], [38]. We use the rounding method, as proposed in in [38]. Because in some cases the number of decimal digits are redundant given a fixed precision, we round each value up to a predetermined number of decimals, and subsequently scale to an integer format to avoid wasting tokens on the decimal point. Hence, the input time series becomes $\mathbf{X}_q = (x_{q_1}, x_{q_2}, \ldots, x_{q_T})$, where $x_{q_t} \in \mathbb{Z}_{\geq 0}$. Below is an example of this operation :

$$0.2437, 0.3087, 0.002, 0.462 \rightarrow \text{"244,309,2,462"}$$

Overall, we use 2 mapping functions: the scaling function noted $\mathcal{E} : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$ and the quantization function noted $\mathcal{Q} : \mathbb{R}_{\geq 0} \rightarrow \mathbb{Z}_{\geq 0}$. Because both mapping functions are reversible up to a certain number of precision digits, we can always reconstruct the input time series: $\mathcal{E}^{-1}\left(\mathcal{Q}^{-1}\left(x_{q_t}\right)\right) \approx x_t$

Rolling windows. Because there is an upper limit on the context length input to LLMs (e.g., MISTRAL has an upper limit of 32k tokens and GPT-3.5-turbo has a limit of 16k tokens), and there are constraints on GPU memory, a rolling windows technique is employed to manage input data that exceeds these thresholds. This method involves segmenting each time series into rolling windows characterized by predetermined lengths and step sizes; i.e., a processed time series $\mathbf{X}_q$ is segmented and turned into a set $\{\left(x^i_{q_{1\ldots w}}\right)\}_{i=1}^{N}$, where w is the window size and N is the number of windows. For a cleaner notation, we refer to the set as

[1] The exact vocabulary size for GPT-3.5-turbo has not been released by OpenAI.

$\{(x^i_{1\ldots w})\}_{i=1}^N$. We drop q in the notation from this point on, as all the input is now quantized.

Tokenization. Different tokenization schemes vary in how they treat numerical values. Several open-source LLMs, such as LLAMA-2 [27] and MISTRAL [40], utilize the SentencePiece Byte-Pair Encoding tokenizer [27], which segments numbers into individual digits. However, the GPT tokenizer tends to segment numbers into chunks that may not correspond directly with the individual digits [43]. For instance, the number 234595678 is segmented into chunks [234, 595, 678] and assigned token IDs [11727, 22754, 17458]. Empirical evidence suggests that this segmentation impedes the LLM's ability to learn patterns in time series data [38]. To make sure GPT tokenizes each digit separately, we adopt the approach introduced by [38], which inserts spaces between the digits in a number.

Continuing with the running example:

"244,309,2,462" → "2 4 4 , 3 0 9 , 2 , 4 6 2"

Where each digit is now encoded separately.

Figure 4.1 shows how different preprocessing steps affect the output of the model. Overall, we find that scaling reduces the number of tokenized digits, and yields better results than not scaling. Moreover, GPT performs better with added space between digits, while MISTRAL does not. These results accord with the forecasting representation presented in Gruver, Finzi, Qiu, *et al.* [38].

4.3 Methods

Given a univariate time series $\mathbf{X} = (x_1, x_2, \ldots, x_T)$, and assuming there exists a set of anomalies of varied length $\mathbf{A} = \{(t_s, t_e)^i \mid 1 \leq t_s < t_e \leq T\}_{i=1}^m$ that is unknown *a priori*, our goal is to find a set of m anomalous time segments, where t_s and t_e represent the start and end time points of an anomalous interval. We introduce two fundamentally different methods that can be used for anomaly detection with LLMs: SIGLLM-PROMPTER and

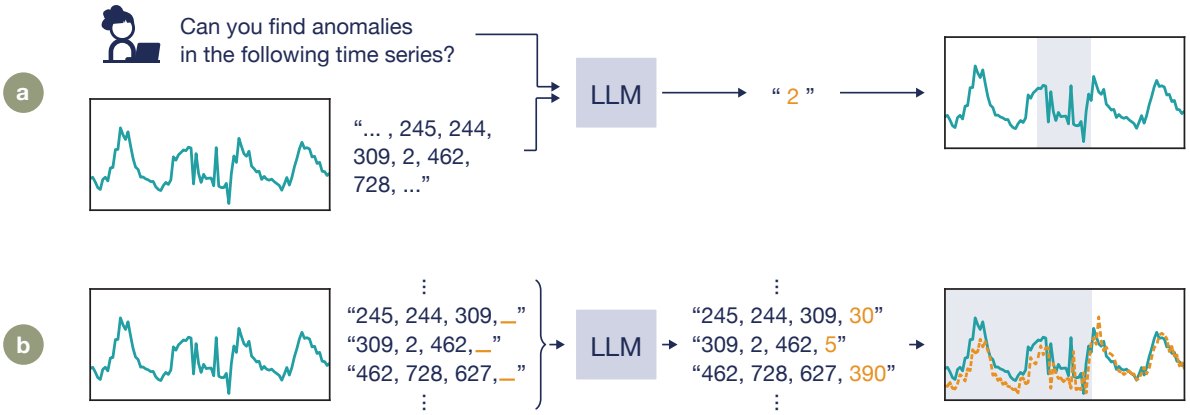

Figure 4.2: Anomaly detection methods in the SIGLLM framework. (a) SIGLLM-PROMPTER: a prompt engineering approach to elicit large language models to identify parts of the input which are anomalies. (b) SIGLLM-DETECTOR: a forecasting approach to use large language models as forecasting methods. SIGLLM-DETECTOR then finds discrepencies between the original and forecasted signal, which indicate the presence of anomalies.

SIGLLM-DETECTOR, as visualized in Figure 4.2.

4.3.1 Finding Anomalies through Prompting

As depicted in Figure 4.2, this pipeline involves querying the LLMs directly for time series anomalies through a text prompt (as shown below) concatenated with the processed time series window $u^i_{1...k} \coloneqq \text{prompt} \oplus (x^i_{1...w})$, where k is the total length of the input after concatenation. LLMs will output the next token u_{k+1} sampled from an autoregressive distribution conditioned on the previous tokens $p_\theta(u_{k+1}|u_{1...k})$.

Following a series of experiments, as shown in Table 4.1, we iterated over trials #5 and arrived at the following prompt for our study:

"You are an exceptionally intelligent assistant that detects anomalies in time series data by listing all the anomalies. Below is a sequence, please return the anomalies in that sequence. Do not say anything like 'the anomalous indices in the sequence are', just return the numbers. Sequence: {the input sequence $(x_{1...w})$}."

Under this prompt, the LLM generates a list of *values* it delineates as point-wise anomalies. It is noteworthy that the GPT-3.5-turbo model is capable of directly outputting anoma-

Table 4.1: Examples of prompts used in SigLLM-Prompter with their respective observed output. $\{x_{1..w}\}$ is a placeholder of the actual signal values in the given window.

Trial	Prompt	Observed Output
1	$\{x_{1..w}\}$. Find the anomalies of the time series above.	(1) generating code with generic stack overflow code for anomaly detection in python with numpy's `convolve` [2] or sklearn's `IsolationForest` [3]. (2) could not find anomalies (3) produced a vague answer about common approaches to finding anomalies
2	Find the range of indices that are anomalous in this series $\{x_{1..w}\}$ or Given this series $\{x_{1..w}\}$. Find the range of indices that are anomalous	(1) producing a list of indices (2) generating code similar to trial #1 (3) could not find anomalies (4) produced a vague answer about common approaches to finding anomalies (5) asked 'do you have any criteria or specific method in mind' (6) confirmed that anomalies are values deviating significantly from the mean. After confirming, the model digressed from the topic
3	Find the anomalous indices in this series $\{x_{t_s-100..t_e+100}\}$. where t_s and t_e is the index of where the anomalies starts and ends, respectively.	(1) producing a list of indices (2) could not find anomalies
4	The anomaly indices in timeseries_1 = $\{x_{1..w}\}_1$ is: $\{t_{1..k}\}_1$ The anomaly indices in timeseries_2 = $\{x_{1..w}\}_2$ is: $\{t_{1..k}\}_2$ The anomaly indices in timeseries_3 = $\{x_{1..w}\}_3$ is:	(1) producing a list of indices (2) claimed anomalies of timeseries_3 had been given (3) could not find anomalies (4) outputted 'Whoa, that's quite a lengthy time series! What can I help you with regarding this data'
5	You are a helpful assitant that performs time series anomaly detection. The user will provide a sequence and you will give a list of indices that are anomalous in the sequence. The sequence is represented by decimal strings separated by commas. Please give a list of indices that are anomalous in the following sequence without producing any additional text. Do not say anything like 'the anomalous indices in the sequence are', just return the numbers. Sequence: $\{x_{1..w}\}$	GPT-3.5-turbo: (1) producing a list of indices (2) occasionally, words like 'Index:' were included (3) sometimes, the output indices exceeded sequence length Mistral: (1) produced a list of **values**.

lous indices using the prompt presented in Table 4.1, while Mistral lacks this ability, as shown in trial #5. To maintain consistency across our experiments, we conducted experiments on both models using the same prompt mentioned above.

As explained in Section III.A, we adopt the rolling windows method, segmenting the time series into rolling windows before inputting it into the LLMs. For each window, we generate 10 samples from the output probability distribution. For each sample containing values deemed anomalous by LLMs, we collect all indices of the window corresponding to those values. Then, the 10 lists of indices are merged together: if an index appears in at least α percent of the total number of samples, it is considered an anomaly. Finally, the lists of detected anomalies from each window are combined to get the final prediction using a similar criterion: an index is considered an anomaly if it appears in at least β percent of the total number of overlapping windows, which are estimated by dividing the window size by the step size. Here, α and β are hyperparameters, which can be tuned to improve performance.

[2] https://numpy.org/doc/stable/reference/generated/numpy.convolve.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

4.3.2 Finding Anomalies through Forecasting

As depicted in Figure 2.1, the first step in a typical ML pipeline involves training an ML model on a collection of time series. From [38], pretrained LLMs are capable of forecasting time series, allowing us to jump straight to the inference phase. Details of preprocessing, forecasting, and post-processing phases of SigLLM-Detector can be found in Alnegheimish, Nguyen, Berti-Equille, *et al.* [1]

4.4 SigLLM primitives and pipelines

In addition to the primitives developed in *Orion*, we introduce several new primitives in Sigllm that facilitate the pre-processing and post-processing of data, ensuring compatibility with LLMs. The new pre-processing primitives include:

`Float2Scalar`

This primitive converts float values into scalar upto certain decimal points.

- **Input:** `x` which is an n-dimensional sequence of values in float type.
- **Output:** `x` which a transformed version in scalar.

`RollingWindow`

This primitive is a simplified version of *Orion*'s `rolling_window_sequences`. This primitive is used in SigLLM-Prompter to create sub-sequences out of the original sequence using a rolling window approach.

- **Input:** `x` which is an 1-dimensional sequence to iterate over
- **Output:**
  - `x` array of rolling windows from the inputted sequence.

- `first_index` first index value of each rolling window.

`Format_as_string`

This primitive converts each sequence of scalar values into strings.

- **Input:** `x` which is an n-dimensional sequence of values
- **Output:** `x.str` which is a string representation version of `X`

In the modeling phase we create 2 primitives: one for detecting anomalies used in SigLLM-Prompter and forecasting timeseries used in SigLLM-Detector using LLM:

`LLM.detect`

This primitive prompts an LLM to detect the anomalies.

- **Input:** `x.str` input sequence
- **Output:** `y` detected anomalous value

`LLM.forecast`

This primitive prompts an LLM to forecast the next steps.

- **Input:** `X` input sequence
- **Output:** `y_hat` predicted sequence

The post-processing primitives include:

`format_as_integer`

This primitive converts each sequence of string values into integers.

- **Input:** `y` which is sequence of string values

- **Output:** `y_int` which is an integer representation version

For post-processing phase in SigLLM-Prompter pipeline, we design the below primitives:

`Val2Idx`

This primitive converts integer values into the indices they appear in the sequence.

- **Input:**
  - `y_int` sequences of anomalous values
  - `x` the input sequences outputted by `RollingWindow` 4.4
- **Output:** `y_idx` sequences of anomalous indices

`find_anomalies_in_windows`

This primitive merges all the samples output by the LLM into a list of anomalous indices for each window: an index is deemed anomalous if it appears in at least α percent of the total number of samples.

- **Input:** `y_idx` n-dimensional array of multiple anomalous indices sequences
- **Output:** `y_win` array of each window's anomalous indices sequences

`merge_anomalous_sequences`

This primitive combines all windows to get the final anomalous indices prediction: an index is deemed anomalous if it appears in at least β percent of the total number of overlapping windows.

- **Input:**
  - `y_win` array of anomalous indices sequences in each window

    - `first_index` first indices of each input sequence outputted by `RollingWindow` 4.4

- **Output:** `y_ano` anomalous indices of the input timeseries.

`format_anomalies`

This primitive uses padding to convert a list of anomalous indices into an array containing the start-timestamp, end-timestamp, and score (which is always 0 in SigLLM-Prompter) for each anomalous sequence that was found.

- **Input:**
    - `y_ano` sequence os anomalous indices
    - `timestamp` sequence of timestamps of the input series
- **Output:** `anomalies` array containing start-timestamp, end-timestamp, and score for each anomalous sequence.

In the post-processing phase of the SigLLM-Detector pipeline, we have developed the following primitives:

`Scalar2Float`

After transforming the raw time series entries into integers during the pre-processing phase 4.4, we now employ this primitive to convert the integer values predicted by the LLM back to float type.

- **Input:** `y_hat` sequence of integer values.
- **Output:** `y_hat` sequences in float form.

`aggregate_rolling_window`

This primitive aggregates multiple prediction samples into one prediction.

- **Input:**
  - `y_hat` n-dimensional sequence of forecasted values
  - `agg` aggregation method, "median" by default
- **Output:** `y_hat` one-dimensional output sequence depicting the aggregated value of forecasts.

The end-to-end pipelines for SigLLM-Prompter and SigLLM-Detector, which incorporate both *Orion* primitives and our custom primitives, are illustrated in Figure 4.3. Here is a breakdown of how the input and output dimensions change through each step of theSigLLM-Prompter pipeline:

1. Input: `x` a time series of length `T`.
2. `time_segments_aggregate` will transform `x` to be of length `T'`. In the case where `x` is already regularly sampled `T` = `T'`. For easier notation, we assume that `x` is regularly sampled.
3. `SimpleImputer` will produce an imputed version of `x` and is of length `T`.
4. `Float2Scalar` will transform `x` from float to integer values while keeping the dimension as `T`.
5. `RollingWindow` will generate prompting samples which can be `x` windows and the dimension would be (`N`, `w`, 1) where `N`= (`T-w`) // `s` with `w` as window size and `s` as step size.
6. `Format_as_string` will convert `x` into its string representation `x.str` for each window which retains the dimension `(N, w, 1)`.

7. LLM will generate `y` which are predicted anomalous values, `y` has dimension (`N`, `n`, `k`), where `n` is number of sample (`n`=10 in this case), and $1 \leq$ `k` $\leq$ `T` is the number of anomalous values the LLM produces, which can be different from sample to sample.

8. `format_as_integer` produces `y_int` which has the same dimesion as `y`.

9. `Val2idx` produces `y_idx` also has the same dimesion as `y`.

10. `find_anomalies_in_windows` produces `y_win` with dimension (`N`, `q`) with `q` is the number of anomalous indices in each window, and its value can vary from window to window.

11. `merge_anomalous_sequences` produces `y_ano` with dimension `K`, which is the number of anomalous indices in the entire sequence.

12. `find_anomalies` will generate `anomalies` of form $(t_s, t_e)^n$ where t_s and t_e are start and end timestamp of each anomalous interval respectively, and now we have n detected anomalous intervals.

Both pipelines take a raw signal, which includes timestamps and their corresponding values, as input. We have set some default hyperparameters values; however, users and easily change those values. This is an example of how to utilize the SigLLM-Prompter pipeline to identify anomalies in a signal; the usage for SigLLM-Detector is similar.

```
from orion.data import load_signal
from mlblocks import MLPipeline
#load signal and pipeline
data = load_signal("exchange-2_cpm_results")
pipeline = MLPipeline("gpt_prompter")
#set hyperparameters
hyperparameters = {
    "mlstars.custom.timeseries_preprocessing.time_segments_aggregate#1": {
        "interval": 3600
    },
    "sigllm.primitives.prompting.anomalies.find_anomalies_in_windows#1": {
        "alpha": 0.5
    },
    "sigllm.primitives.prompting.anomalies.merge_anomalous_sequences#1": {
        "beta": 0.5
    }
}
pipeline.set_hyperparameters(hyperparameters)
#detect anomalies
context = pipeline.fit(data)
context["anomalies"] #the detected anomalies

```

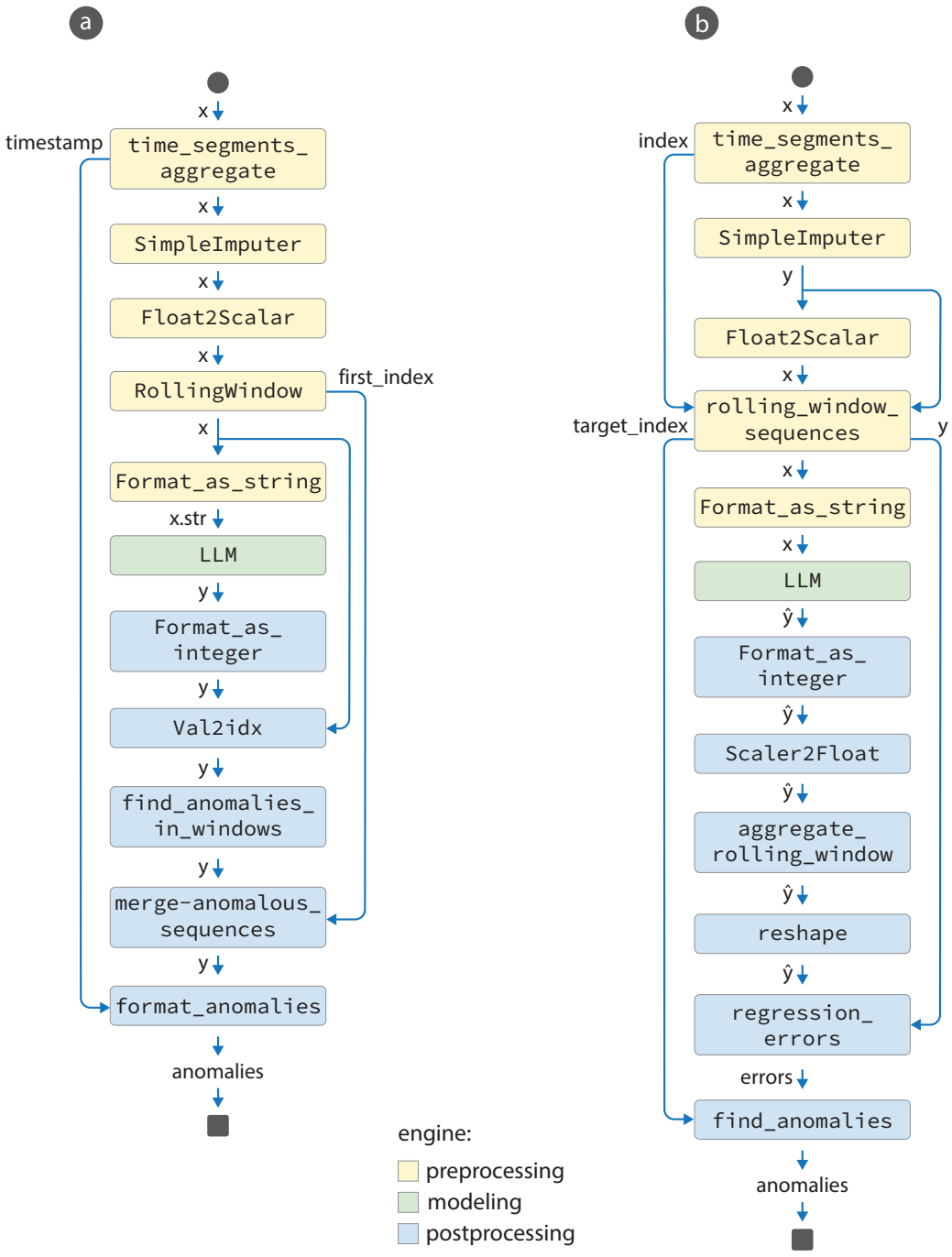

Figure 4.3: Graphic representations of (a) SigLLM-Prompter pipeline and (b) SigLLM-Detector pipeline with any generic LLM. Output format and structure for each data processing block here are explained in text before.

Chapter 5

Evaluation

Sections 5.1 and 5.2, together with subsection 5.3.2, 5.4,2 are from joint work with Alnegheimish, Nguyen, Berti-Equille, et al. *[1].*

In this chapter, we assess our two frameworks and seek to answer the following research questions:

- **RQ1** How do time series foundational models compare to baseline methods? What are their failure cases and why?
- **RQ2** How does SigLLM compare to baseline methods? What are their failure cases and why?
- **RQ3** How do foundational models that were pretrained on time series, `UniTS` and `TimesFM`, compare to LLMs?

5.1 Datasets

We examined SigLLM and `TimesFM` & `UniTS` on 11 datasets with known ground truth anomalies. These datasets were gathered from a wide range of sources, including a satellite telemetry signal corpus from **NASA** [1] that includes two sub-datasets: SMAP and MSL;

[1] https://github.com/khundman/telemanom

Yahoo S5 [2], which contains four sub-datasets: A1, which is based on real production traffic to Yahoo systems, and three others (A2, A3, and A4) which have been synthetically generated; and **NAB** [3], which includes multiple types of time series data from various application domains. We consider five sub-datasets: Art, AWS, AdEx, Traf, and Tweets. In total, these datasets contain 492 univariate time series and 2,349 anomalies. The properties of each dataset, including the number of signals and anomalies, the average signal length, and the average length of anomalies, are presented in Table 5.1. The table makes clear how properties differ between datasets; for instance, the NASA and NAB datasets contain anomalies that are longer than those in Yahoo S5

Table 5.1: Dataset Summary: 492 signals and 2349 anomalies.

Dataset	# Sub-datasets	# Signals	# Anomalies	Avg. Length
NASA	2	80	103	8686 ± 5376
Yahoo S5	4	367	2152	1561 ± 140
NAB	5	45	94	6088 ± 3150
Total	11	492	2349	

Table 5.2 provides an overview of the anomalies type in each of the 11 benchmark datasets. The majority of anomalies in Yahoo S5's A3 & A4 datasets are point anomalies, whereas, **NASA** and **NAB** datasets mostly contain contextual anomalies.

Table 5.2: Overview of anomalies in 11 benchmark datasets.

	NASA		Yahoo S5				NAB				
No. of	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
Point Anomalies (*len = 1*)	0	0	68	33	935	833	0	0	0	0	0
Contextual Anomalies (*len>1*)	36	67	110	167	4	2	6	11	30	14	33
Anomalous Points	7766	54696	1669	466	943	837	2418	795	6312	1560	15651
Total Points	132046	562800	94866	142100	168000	168000	24192	7695	67644	15662	158511

[2] https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70
[3] https://github.com/numenta/NAB

5.2 Baseline Models and Metrics

5.2.1 Models

We compared SigLLM and `TimesFM` & `UniTS` to state-of-the-art models in unsupervised time series anomaly detection. This includes a variety of models similar to the ones considered in [42], [44]:

- Classic statistical methods including `ARIMA`, `Matrix Profiling (MP)`, and a simple `Moving Average (MAvg)`.
- Deep learning models currently considered state-of-the-art, including `LSTM DT` which is a forecasting-based model, `LSTM AE`, `VAE`, and `TadGAN` which are reconstruction-based models, and `AER` which is a hybrid between forecasting and reconstruction.
- `AnomalyTransformer (AT)`, a transformer architecture model for anomaly detection.
- `MS Azure`, an anomaly detection service.

These models use a wide range of underlying detection methods, which increases our anomaly detection coverage overall.

5.2.2 Metrics

We utilized anomaly detection-specific metrics for time series data [44], [45]. Namely, we looked at the F1 score under overlapping segement A.1.2, under which both partial and full anomaly detection are considered correct identification.

5.3 Hyperparamters and Computation

5.3.1 Orion pretrained pipelines

Hyperparameters. We set the rolling window size to 250 for `UniTS`, and 256 for `TimesFM` (that was because the input context length of `TimesFM` model has to be a multiple of 32). Since we did one-step forecasting, we set both the step size value and the prediction horizon to be 1.

Computation. For the `TimesFM` pipeline, we use the `timesfm-1.0-200m` model hosted by Huggingface[4]. While the Das, Kong, Sen, *et al.* [10] paper expertimented with models of 17M, 70M, and 200M parameters, only the 200M model is publicly available.

The Github repository[5] of the `UniTS` model provided accesss to two model checkpoints `units_x128_pretrain_checkpoint` and `units_x32_pretrain_checkpoint` that have 3.5M and 370K parameters respectively. They also provided the dataset as detailed in Table A.2 and code to pretrain a zero-shot forecasting specific model of size 1M parameters. We tested the performance of those three models on 10 selected signals[6], and found that the zero-shot forecasting model has the best performance[7] despite not being the largest model. Therefore, we chose the zero-shot forecasting model for benchmarking. The model checkpoint is available for download at https://sintel-orion.s3.amazonaws.com/pretrained/units.pth

5.3.2 SigLLM

Hyperparameters. For SigLLM-Prompter, GPU capacity means that the maximum input window length of SMAP and MSL is 500 values (for other datasets, it is 200 values). We chose a step size such that, on average, a value was contained in 5 overlapping windows

[4] https://huggingface.co/google/timesfm-1.0-200m

[5] https://github.com/mims-harvard/UniTS?tab=readme-ov-file

[6] We chose ten **NASA** timeseries: S-1, M-1, P-1, E-1, A-1, B-1, C-1, D-1, M-6, M-7.

[7] The F-1 scores of the zero-shot forecasting model, `units_x128_pretrain_checkpoint`, and `units_x32_pretrain_checkpoint` are 0.45, 0.35, and 0.34 respectively.

(i.e, 100 steps for SMAP and MSL, and 40 for others). The SigLLM-Prompter approach originally produced an extremely high number of anomalies. We introduced the α and β hyperparameters to filter the end result. We performed an ablation study to test multiple combinations of α and β values on the F1 score. For some windows, the LLMs consistently outputted more than half of the window values as anomalous; thus, we discarded the predicted results of all windows containing all 10 samples, which was more than 50% of the windows. Fig 5.1 shows detection F1 scores from different combinations of α and β values. We observed that on all datasets, for Mistral, $\alpha = 0.4$ and $\beta = 0.9$ yielded the best F1 score; for GPT, $\alpha = 0.2$ and $\beta = 0.9$ yielded the best F1 score as shown in Fig 5.1.

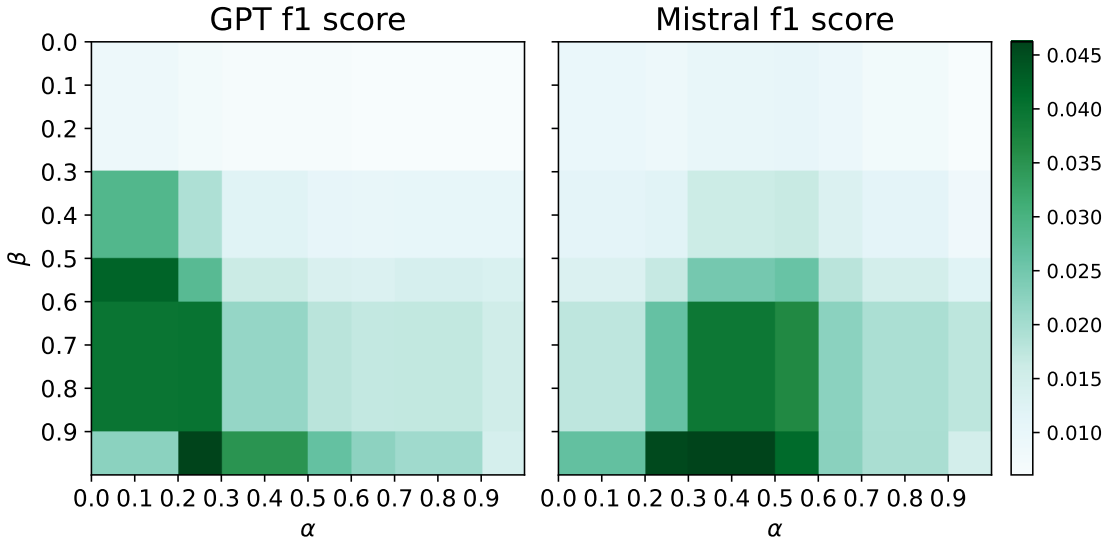

Figure 5.1: Optimizing the choice α and β values based on the average F1 scores on all datasets.

For SigLLM-Detector, we set the window size to 140 and the step size to 1. With a rolling window strategy of step size 1, we set the horizon to 5. For aggregation function, we chose the 5^{th}-percentile, and 95^{th}-percentile values of the predicted distribution to recreate a one-dimensional signal. These hyperparameter values were informed by ablation study in Alnegheimish, Nguyen, Berti-Equille, *et al.* [1]

Computation. For GPT, we used GPT-3.5-turbo due to its superior performance on time series data (demonstrated by Gruver, Finzi, Qiu, *et al.* [38]) and its affordability. For

Table 5.3: Benchmark Summary Results depicting F1 Score.

	NASA		Yahoo S5				NAB						
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	$\mu \pm \sigma$	Avg elapsed(s)
AER	**0.587**	**0.819**	**0.799**	**0.987**	**0.892**	0.709	**0.714**	**0.741**	0.690	**0.703**	0.638	**0.753 ± 0.109**	25.0
LSTM DT	0.471	0.726	0.728	0.985	0.744	0.646	0.400	0.468	0.786	0.585	0.603	0.649 ± 0.161	17.1
ARIMA	0.525	0.411	0.728	0.856	0.797	0.686	0.308	0.382	0.727	0.467	0.514	0.582 ± 0.176	193.9
MP	0.474	0.423	0.507	0.897	0.793	**0.825**	0.571	0.440	0.692	0.305	0.343	0.570 ± 0.193	33.9
TadGAN	0.560	0.605	0.578	0.817	0.416	0.340	0.500	0.623	0.818	0.452	0.554	0.569 ± 0.142	171.6
LSTM AE	0.545	0.662	0.595	0.867	0.466	0.239	0.667	**0.741**	0.500	0.500	0.475	0.569 ± 0.158	13.5
VAE	0.494	0.613	0.592	0.803	0.438	0.230	0.667	0.689	0.583	0.483	0.533	0.557 ± 0.143	27.7
AT	0.400	0.266	0.571	0.565	0.760	0.576	0.414	0.430	0.500	0.371	0.287	0.467 ± 0.138	11.1
MAvg	0.171	0.092	0.713	0.356	0.647	0.615	0.222	0.408	**0.880**	0.157	**0.776**	0.458 ± 0.266	N/A
MS Azure	0.051	0.019	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176	0.243 ± 0.225	1.9
UniTS	0.533	0.554	0.644	0.706	0.015	0.076	0.364	0.458	0.667	0.686	0.551	0.478 ± 0.237	21.2
TimesFM	0.533	0.644	0.628	0.551	0.029	0.052	0.4	0.475	0.783	0.537	0.563	0.472 ± 0.235	13.1
SigLLM-Prompter Mistral	0.160	0.154	0.194	0.235	0.338	0.336	0.370	0.268	0.000	0.135	0.257	0.223 ± 0.104	_ _
SigLLM-Prompter GPT	0.049	0.110	0.143	0.078	0.157	0.195	0.154	0.194	0.133	0.133	0.197	0.133 ± 0.076	_ _
SigLLM-Detector	0.429	0.431	0.615	0.828	0.376	0.363	0.400	0.362	0.727	0.480	0.762	0.525 ± 0.167	_ _

Mistral, we used the publicly available model hosted by HuggingFace [8]

UniTS, TimesFM, and Mistral were used on an Intel i9-7920X 24 CPU core processor and 128GB RAM machine with 2 dedicated NVIDIA Titan RTX 24GB GPUs. On the other hand, GPT was queried through an API key hosted by OpenAI. For benchmarking, we use Intel Xeon processor of 10 CPU cores (9 GB RAM per core) and one NVIDIA Volta V100 GPU with 32 GB memory.

5.4 Qualitative and Quantitative Performance

The total F1 scores of *Orion* pretrained and SigLLM pipelines in comparison with other pipelines are shown in Table 5.3. In this section, we will go into more detail about how the newly introduced pipelines perform compared to existing unsupervised anomaly detection pipelines, and their limitations.

5.4.1 *Orion* pretrained pipelines

The average F1 scores for the UniTS and TimesFM pipelines are closely aligned, falling within one standard deviation of each other. Additionally, the performance across different datasets is fairly comparable. Figure 5.2 shows examples of anomalies identified by

[8] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

UniTS and TimesFM.

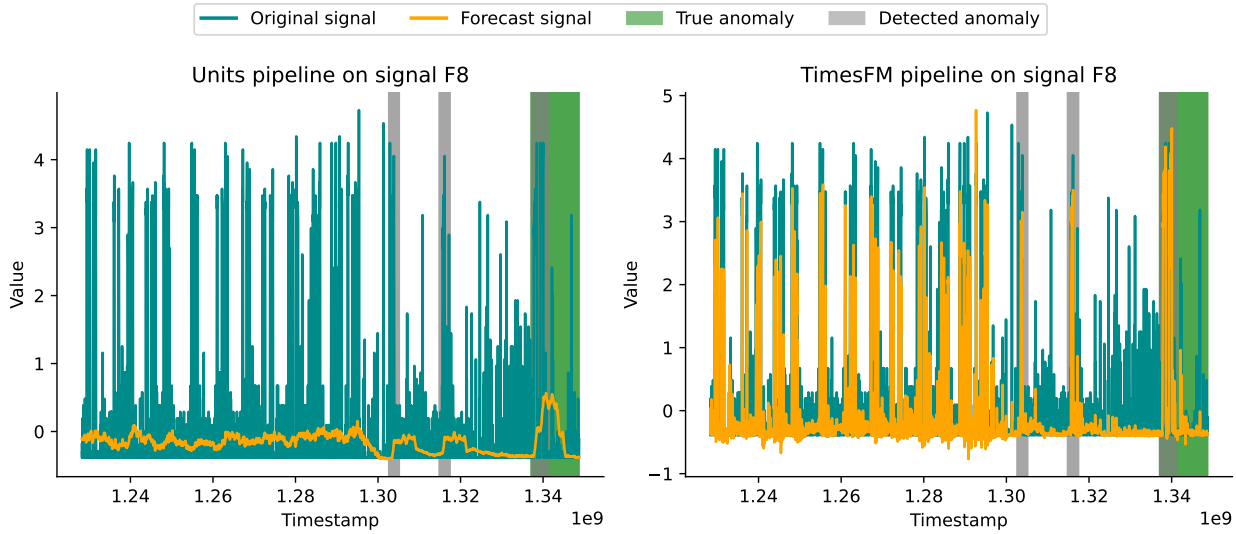

Figure 5.2: Examples of anomalies successfully identified by *Orion* pretrained pipelines. (Left) Despite only being able to capture the trend but not the periodicity of the signal, the UniTS pipeline can still detect one true anomaly. (Right) TimesFM model is better at forecasting the signal, both in terms of trends and periodicity. However, it still could not detect any more anomaly.

How do time series foundation models compare to baseline approaches?

Table 5.3 indicates that the mean F1 score of the *Orion* pretrained pipeline is comparable to that of the Moving Average method, suggesting that UniTS and TimesFM perform similarly to the baseline on average. However, when examining performance at the dataset level, UniTS and TimesFM either exceed or match the baseline in most cases, with the exceptions being Yahoo A3 and A4. This observation points to opportunities for enhancing the performance at these specific datasets.

What are the failure cases and why?

UniTS' poor performance on Yahoo's A3 and A4 datasets suggests that the UniTS pipeline is not able to detect point anomalies. As we have shown in Table 5.1, Yahoo's A3 and A4 have mostly point anomalies. When we take a closer look at how UniTS model forecasts the next values, we notice that it could capture the trend of the signal, but struggles to forecast

its periodicity. Therefore, the reconstruction errors are large even when at non-anomalous intervals. When applying an exponentially weighted function, as mentioned in section 3.2, the errors got smoothed out, thus masking the point anomalies. In Figure 5.3, we show that after removing the error smoothing step, the pipeline could capture one more point anomaly in signal A3Benchmark-TS45. A similar behavior is also observed in `TimesFM`. We recommend exploring more hyperparameters tuning to improve point anomaly detection ability for these models.

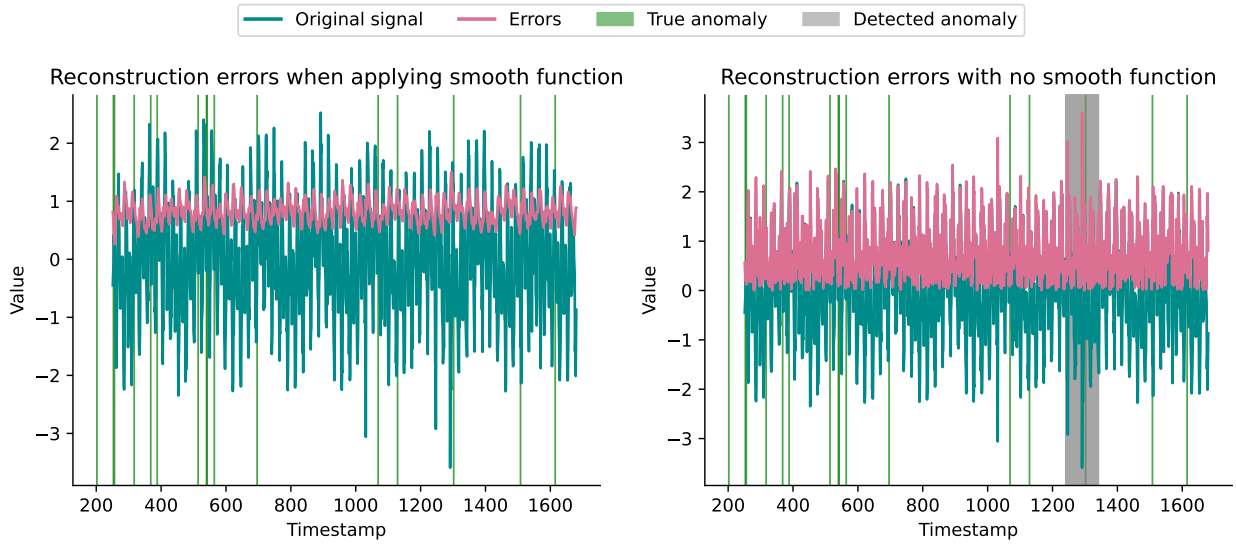

Figure 5.3: The performance of `UniTS` pipeline on an Yahoo A3 data (A3Benchmark-TS45). (Left) The pipeline could not identify any point anomaly if the errors were "smoothed" out using an exponential weighted moving average. (Right) When using the raw point-wise residuals, the pipeline could detect one point anomaly.

5.4.2 SigLLM

A comparison of the SigLLM-Prompter and SigLLM-Detector with GPT and Mistral models is shown in Table 5.4.

Overall, Mistral achieved better results than GPT for the SigLLM-Prompter method, with a 2× improvement in F1 score. In addition, SigLLM-Detector performed better overall than SigLLM-Prompter. We investigate each approach below.

Table 5.4: Summary of Precision, Recall, and F1 Score

	Precision	Recall	F1 Score
SigLLM-Prompter Mistral	0.219 ± 0.108	0.311 ± 0.213	0.223 ± 0.104
SigLLM-Prompter GPT	0.162 ± 0.133	0.245 ± 0.191	0.133 ± 0.076
SigLLM-Detector	**0.613 ± 0.184**	**0.514 ± 0.211**	**0.525 ± 0.167**

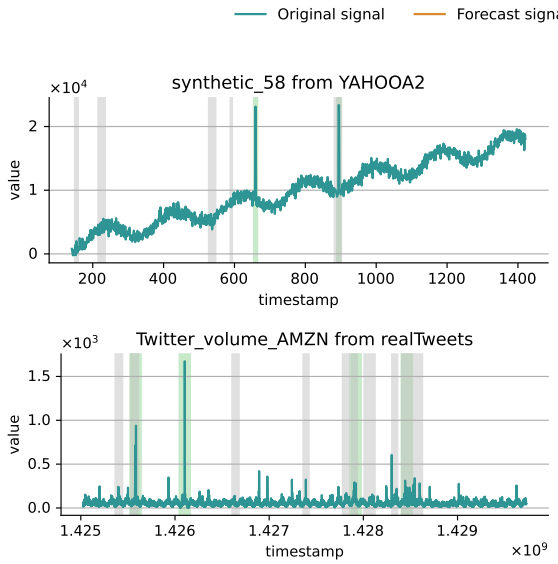

Figure 5.4: Examples of anomalies identified through SigLLM-Prompter. While the model was able to find anomalies, the number of false positives was high, and there were false negatives.

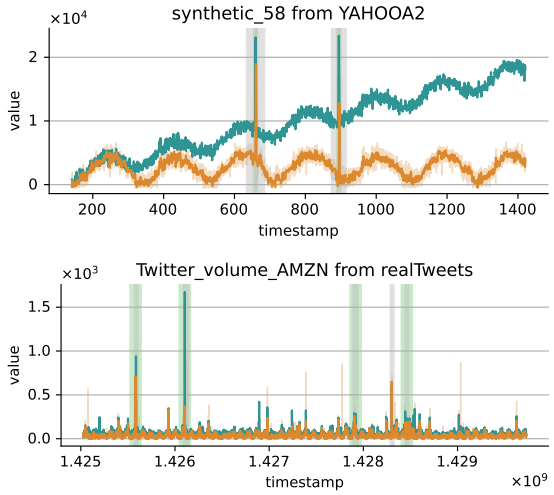

Figure 5.5: Examples of anomalies successfully identified by SigLLM-Detector. Even though the model did not capture the trend present in `synthetic_58`, it still managed to find the anomalous intervals.

How does the SigLLM compare to baseline approaches?

Table 5.3 highlights the F1 score obtained for each of the 11 datasets.

LLM-based methods can perform surprisingly well. Compared to the baseline model `MAvg`, our methods achieved an F1 score 14.6% higher. Moreover, the SigLLM-Detector pipeline alone surpasses `MAvg` performance in 6 out of 11 datasets. We can best see the potential of SigLLM-Detector in the Tweets dataset, where the gap between the LLM's result and the highest result (from `MAvg`) is minimal, at only 1.8%.

What are the failure cases and why?

Figures 5.4 and 5.5 illustrate example outputs for both the SigLLM-Prompter and SigLLM-Detector methods, respectively. We focus on Mistral since it yielded better overall results than GPT, as depicted in Table 5.4.

Even though SigLLM-Detector correctly identified all anomalies in the example shown in Figure 5.5, the forecast itself struggled to capture the non-stationary aspects of the signal, particularly its trend. This is due to the sensitivity of LLMs to context length. A window size larger than 140 is needed to capture this property. While it did not impact the detection in this case, this may explain failure cases in other signals.

SigLLM-Prompter raised a large number of false alarms, with an average precision of 0.219. Using the filtering method described in Section 4.3.1 does not eliminate false positives. An alternative strategy could be to use log probabilities as a measure of confidence for filtering. We recommend exploring this avenue in future work.

5.4.3 How do `UniTS` & `TimesFM` compare to SigLLM?

When comparing the performance of SigLLM-Detector and the *Orion* pretrained pipelines, we could see that on average, SigLLM-Detector performs better. Especially, SigLLM-Detector could detect the point anomalies in Yahoo A3 and A4. However, SigLLM-Detector does not perform as well at the NASA datasets, which could be attributed to LLMs's inability to capture the trend in long sequences. On the other hand, *Orion* pretrained pipelines consistently perform better than SigLLM-Prompter in all datasets, except Yahoo A3 and A4. *Orion* pretrained pipelines also perform better than SigLLM-Prompter on average.

Overally, it is noteworthy that despite being pretrained specifically on time series data, **time series foundational models like `UniTS` & `TimesFM` still do not perform as well as LLMs** that were not trained on time series task. We speculate that this fact could be

attributed to the model size: LLMs' number of parameters is of 10^2 order of magnitude bigger than that of `UniTS` & `TimesFM`). However, more studies need to be done to verify this.

5.5 Computational performance

5.5.1 Runtime

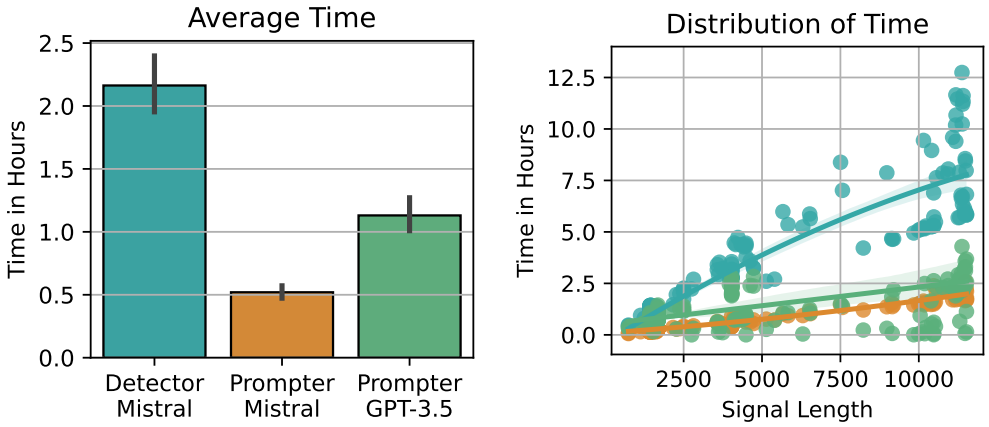

Figure 5.6: Recorded time for SigLLM-Prompter and SigLLM-Detector. (left) On average, SigLLM-Detector takes the longest to infer, almost double the time of SigLLM-Prompter. (right) Distribution of signal length and execution time.

The attractiveness of utilizing foundational models for this task lies in their ability for zero-shot learning, eliminating the need for time-consuming and resource-intensive fine-tuning. However, does this approach genuinely save time and resources when employing foundational models?

The primary constraint in employing foundational models within a pipeline is the inference time. As illustrated in Table 5.3, the inference durations on average for `UniTS` and `TimesFM` are **21.2s** and **13.1s**, respectively. In contrast, traditional deep learning methods such as AER and LSTM DT require model training for each individual signal, yet Time Series pretrained models exhibit comparable inference times.

When examining Large Language Models (LLMs), the inference times are even more pronounced. Figure 5.6 depicts the average duration required for LLMs to generate responses

across various approaches. Waiting between half an hour to two hours for a model output is impractical, particularly since conventional deep learning models require a much less amount of time for training and inference.

5.5.2 Cost

Regarding resource usage, all foundational models (`UniTS`, `TimesFM`, and MISTRAL) were executed in an offline manner, so there is no cost to run them. GPT was operated via an API. The cumulative expense for running SIGLLM-PROMPTER experiments using the `gpt-3.5-turbo-instruct` version totaled approximately $834.11, averaging $1.69 per signal. In a smaller-scale study of SIGLLM-DETECTOR, sampling 22 signals (about 5% of the dataset) incurred a total cost of $95.08, averaging $4.30 per signal, indicating that SIGLLM-DETECTOR is a more costly approach compared to SIGLLM-PROMPTER.

5.6 Discussion

5.6.1 Deployment of `UniTS` Model.

A notable challenge faced in this research is the current status of the `UniTS` model, which is not yet in the deployment phase. Despite the availability of the model architecture and training data, we still need to pretrain the zero-shot forecasting model. Additionally, during the inference phase, we encountered the necessity to modify the original code. The existing code is highly tailored to the model's original dataset, which required us to generalize it to ensure compatibility with a generic data format in `pandas`. This process underscores the complexities involved in adapting established models for diverse applications and highlights the importance of flexibility in code design to facilitate broader usability.

5.6.2 Prompting Challenges.

Over a three-month experimental period, various prompts were employed, as laid out in Table 4.1. It is evident that both GPT and MISTRAL fail to produce the desired responses unless a chat template is applied that attributes roles to the user and the system. Furthermore, to ensure the exclusivity of numerical values in the generated responses, in addition to specifying in the prompt to "just return numbers," we reduced the likelihood of non-numerical tokens appearing in the output generated by the LLMs.

Under the 'find indices' prompt, GPT may generate lists of indices; however, these indices frequently surpass the sequence length. Conversely, MISTRAL yields values instead of indices when utilizing the same prompt. Therefore, for our experiment, we altered the prompt to include "find values" rather than "find indices."

Unlike MISTRAL, GPT outputs a "repetitive prompt" error when presented with a series of identical values within a window. This happened particularly for NASA datasets (there are 23 signals in SMAP and 13 in MSL with this error, affecting up to 85% of the windows). In this experiment, we deemed such windows as having no detectable anomalies, obtaining a true positive of zero.

5.6.3 Addressing Memorization.

Large language models are trained on a vast amount of data. The training data for most models – for instance, those provided by OpenAI – is completely unknown to the general public, which makes evaluating these models a nuanced problem. Given that large language models, especially GPT models [46], are notorious for memorizing training data [47], how do we ensure that there was no data and label leakage for the benchmark datasets used? We posit that our transformation of the time series data into its string representation is unique, essentially making the input time series different from its original form and reducing the chances of blatant memorization. Moreover, unlike with the forecasting task, the task

of anomaly detection is not inherent to the training convention used, which is next token prediction.

As for the `TimesFM` and `UniTS` models, we have shown in Chapter A.2 that their training datasets do not include our testing data.

Chapter 6

Conclusion

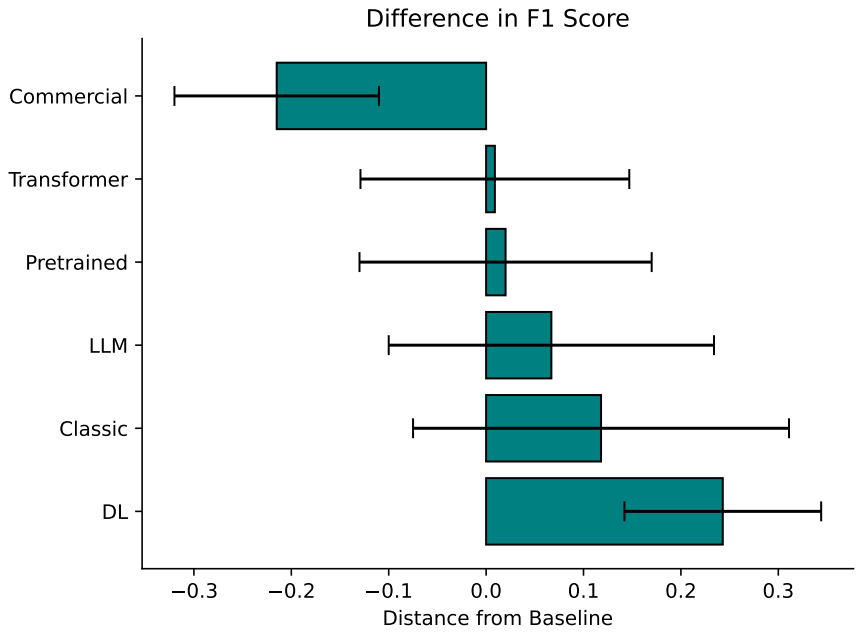

Figure 6.1: F1 Score performances of different model types, compared to a moving average baseline. Each category represents a collection of models that fall under that group. For *classic* models, we consider ARIMA and Matrix Profiling; for *Deep Learning (DL)*, we utilize AER and LSTM DT; for *transformer* anomaly detection models, we look at Anomaly Transformer; lastly, for the *commercial* category, we compare to MS Azure.

6.1 Conclusion

In conclusion, this thesis investigates the use of foundational models for anomaly detection with no prior learning. We introduce a new pipeline class in *Orion*: the pretrained pipeline, which leverages a Time Series Foundational model for signal forecasting, followed by anomaly detection without requiring a formal training phase. Additionally, we present SigLLM, an innovative framework that transforms signals into text, allowing large language models (LLMs) to process time series data.

Both the *Orion* pretrained pipelines and SigLLM surpass the baseline model, with SigLLM demonstrating superior performance that closely rivals classical methods when evaluated on 492 signals, falling short of deep learning models by a factor of just $\times 1.2$, as illustrated in Figure 6.1. Surprisingly, even though time series foundational models such as `UniTS` and `TimesFM` are specifically pretrained on time series data, they do not outperform large language models (LLMs) that were not explicitly trained for time series tasks. Nevertheless, a significant drawback of LLMs is their restricted context window size and long inference time. In SigLLM, we employ rolling windows to segment the time series into smaller parts, which proves to be both inefficient and costly.

6.2 Future Directions

6.2.1 Multivariate time series

The *Orion* pretrained pipelines are capable of supporting multivariate time series anomaly detection. In contrast, the SigLLM framework is limited to handling univariate time series. At this stage, we do not possess a module for converting multivariate signals into text suitable for input into LLMs.

6.2.2 Post modeling

In anomaly detection, post-processing strategies play a crucial role in identifying the locations of anomalies. Moving forward, additional post-processing functionalities can be tested to assist SigLLM-Prompter in filtering out false alarms. Likewise, a comprehensive investigation into error functions can enhance the detection of anomalies in both SigLLM-Detector and the *Orion* pretrained pipelines.

6.2.3 Experiment with new model

As large language models (LLMs) continue to progress, an increasing number of them will be able to accommodate larger context sizes. Our SigLLM framework is versatile and can be adapted to various models. Consequently, in the future, advancements in model capabilities may address the context size issue, potentially resulting in more efficient runtime and improved accuracy in results.

Appendix A

Appendix

A.1 *Orion* for Time Series Anomaly Detection

This section is adapted from Song [24].

In most anomaly detection approaches, the data in its raw form cannot be effectively modeled and must undergo several preprocessing steps to become suitable for modeling and scoring. Thus, there arises a need to conceptualize anomaly detection as a sequential workflow that consists of distinct modules for preprocessing, modeling, and post-processing, rather than as a single transformation process [48]. This concept underpins the design of the library *Orion*[1], which builds comprehensive end-to-end pipelines for anomaly detection. In the following sections, we will provide an overview of the *Orion* framework [12].

A.1.1 Primitives and pipelines

At the core of the *Orion* framework is the concept of **primitives** and **pipelines**. This idea emerges from the Machine Learning Blocks (*MLBlocks*) framework introduced by Xue [49], Santiago [50], and Smith, Sala, Kanter, *et al.* [51], with the goal to reconcile multiple data science software tools that are often scattered across multiple libraries, depending on

[1] https://github.com/sintel-dev/Orion

the type of data and the stage of the workflow. For instance, the *pandas*[2] library supports prepossessing and visualization, but it does not have modeling capacities. On the other hand, libraries such as `scikit-learn`[3] and *XGBoost*[4] provide extensive modeling options but fall short in offering a comprehensive suite of preprocessing functions. This dichotomy highlights the need for an integrated approach, where users can seamlessly transition between preprocessing, modeling, and other necessary steps within the data science process, thereby enhancing efficiency and coherence in their workflows.

Primitives

An acceptable *MLBlocks* primitive is a `Python` object that can be imported and must meet the following criteria:

- It can be either a function or a class.
- If it is a class, it may include a fitting stage where the primitive receives training data to learn from, which can be executed with a single method invocation. Function primitives do not include a fitting stage.
- It must include a producing stage, wherein the primitive takes in data and returns either a transformation of the input data or a new dataset derived from it, such as a prediction set. This producing stage should be executed with a single function or method call.
- It may include hyperparameters, which are additional arguments that can be supplied to modify or control the behavior of the fitting and producing stages, either through the function call or the class constructor.

Below are several examples of *MLBlocks* primitives[5] from some popular libraries:

[2] https://pandas.pydata.org/docs/index.html
[3] https://scikit-learn.org/stable/
[4] https://xgboost.readthedocs.io/en/stable/
[5] https://mlbazaar.github.io/MLBlocks/advanced_usage/primitives.html

Table A.1: Examples of primitives

Primitive	Type	Fit	Produce
`sklearn.preprocessing.StandardScaler`	class	fit	transform
`sklearn.ensemble.RandomForestClassifier`	class	fit	predict
`keras.applications.resnet50.preprocess_input`	function	-	-

Pipelines

Pipelines are comprehensive, end-to-end programs that consist of primitives collaborating to learn from training data and subsequently make predictions on new data. In the case of anomaly detection, a pipeline is fitted on the training dataset and is then prompted to detect anomalies in the new signal. Users can either use the pre-written anomaly detection pipelines in *Orion* or write their own pipelines following the procedure of *MLBlocks*. A typical pipeline usually includes a pre-processing, modeling, and post-processing phase. Some available pipelines in *Orion* are: Long-Short-Term-Memory models (LSTMs) [18], AutoEncoders (AE) [21], Variational AutoEncoders (VAE) [22], Generative Adversarial Networks (GANs) [23], and Transformers [33], [52].

A.1.2 Benchmark and metrics

Orion also supports benchmarking the pipelines's performance on a collection of different time series datasets from reputable data sources - **NASA**[6], **Yahoo**[7], and **NAB**[8].

The metrics that are traditionally used in classification tasks like *precision*, *recall*, and *F1-score* cannot account for the case when data is not regularly sampled [45]. Therefore, instead of simply counting the number of true positives, false positives, false negatives, and true negatives, *Orion* introduces two methods to enable the fair computation of metrics without restrictions on the data: **weighted segment** and **overlapping segment**[12].

[6] https://github.com/khundman/telemanom
[7] https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70
[8] https://github.com/numenta/NAB

Weighted segment

In this evaluation approach, we compute numbers of true positives, false positives, false negatives, and true negatives not based on flagged anomalous and true anomalous timestamps, but based on the segment, then weigh each segment by its time range.

Overlapping segment

This is a more lenient approach, which is inspired by Hundman, Constantinou, Laporte, *et al.* [20], which rewards the model even if it only detects a subset of anomaly. In chapter 5, we will use this metric for benchmarking.

A.2 Time Series Foundational Models' pretrained data

A.2.1 `TimesFm`

`TimesFM` model was trained on a large-scale time-series corpus consisting of real-world and synthetic data. The real-world data included:

- Google Trends [9]: Search interest data for approximately 22,000 queries over 15 years (2007-2021), at various granularities (hourly, daily, weekly, monthly).
- Wiki Pageviews [10]: Hourly page view data for all Wikimedia pages from January 2012 to November 2023, aggregated to different granularities.
- Other real-world public datasets: The M4 dataset [53], Electricity dataset, Traffic dataset [54], and Weather dataset [54] were also included.

In terms of synthetic data, the authors generated signals using ARIMA processes that include seasonal patterns, trends, and step functions.

[9] https://trends.google.com/
[10] https://en.wikipedia.org/wiki/Wikipedia:Pageview_statistics

A.2.2 `UniTS`

The zero-shot forecasting model was trained using the forecast and classification datasets shown in Table A.2.

Table A.2: Summary of `UniTS` training data

Name	# Variables	Task
NN5 [55]	111	Forecast
ECL [56]	321	Forecast
ETTh1 [54]	7	Forecast
Exchange [57]	8	Forecast
Traffic [58]	862	Forecast
Weather [59]	21	Forecast
Heartbeat [60]	61	Classification
JapaneseVowels [61]	12	Classification
PEMS-SF [62]	963	Classification
SelfRegulationSCP2 [63]	7	Classification
SpokenArabicDigits [64]	13	Classification
UWaveGestureLibrary [65]	3	Classification
ECG5000 [66]	1	Classification
NonInvasiveFetalECGThorax1 [67]	1	Classification
Blink [68]	4	Classification
FaceDetection [69]	144	Classification
ElectricDevices [70]	1	Classification
Trace [71]	1	Classification
FordB [72]	1	Classification
MotionSenseHAR [73]	12	Classification
EMOPain [74]	30	Classification
Chinatown [72]	1	Classification
MelbournePedestrian [72]	1	Classification
SharePriceIncrease [75]	1	Classification

A.3 SigLLM's extra results

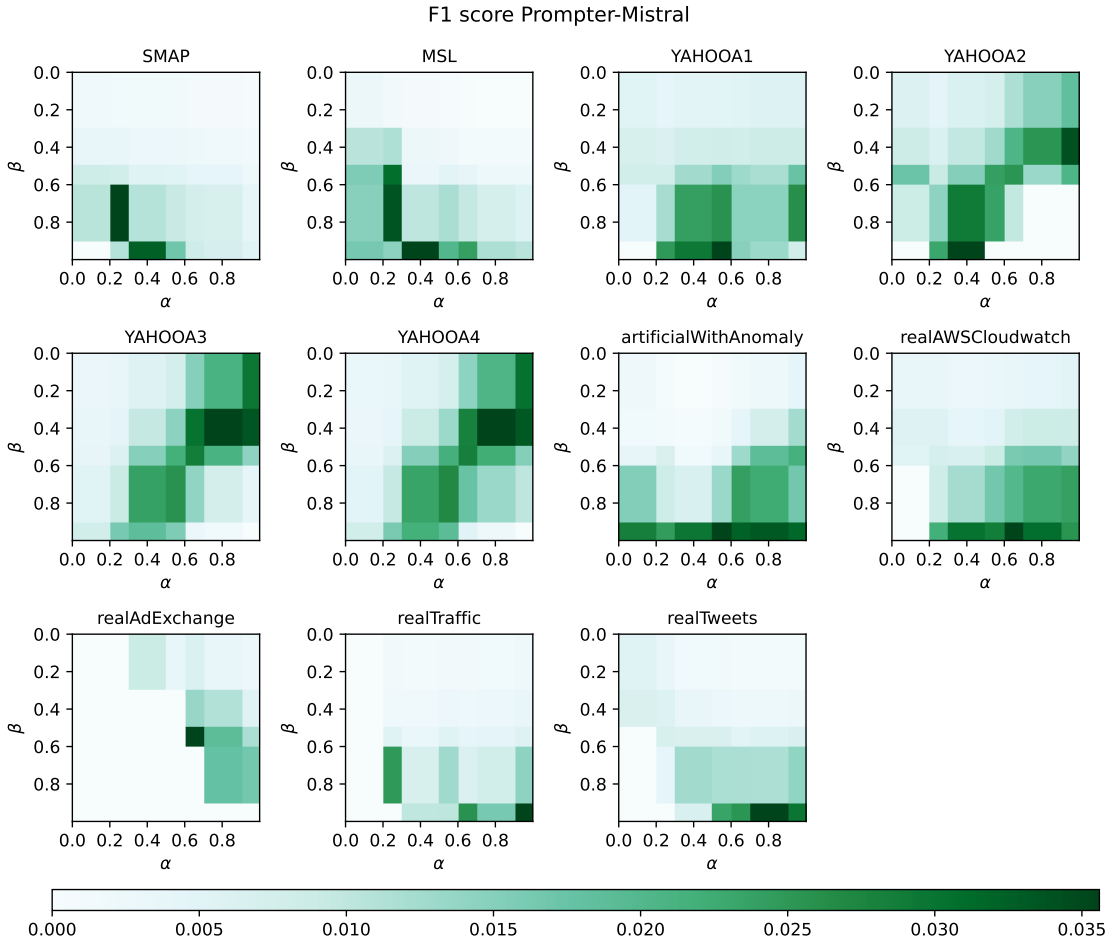

Figure A.1: SigLLM-Prompter Mistral F1 score versus α and β for each dataset.

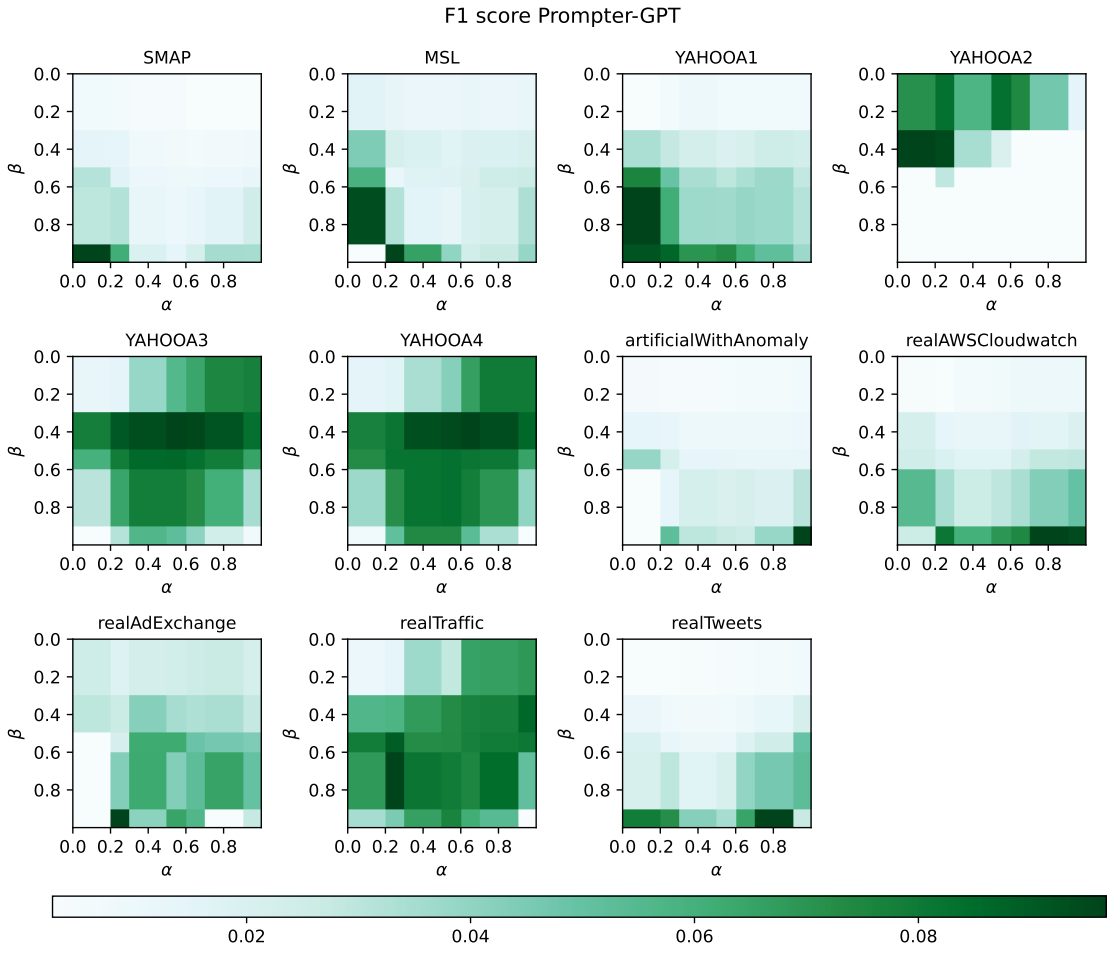

Figure A.2: SigLLM-Prompter GPT F1 score versus α and β for each dataset.

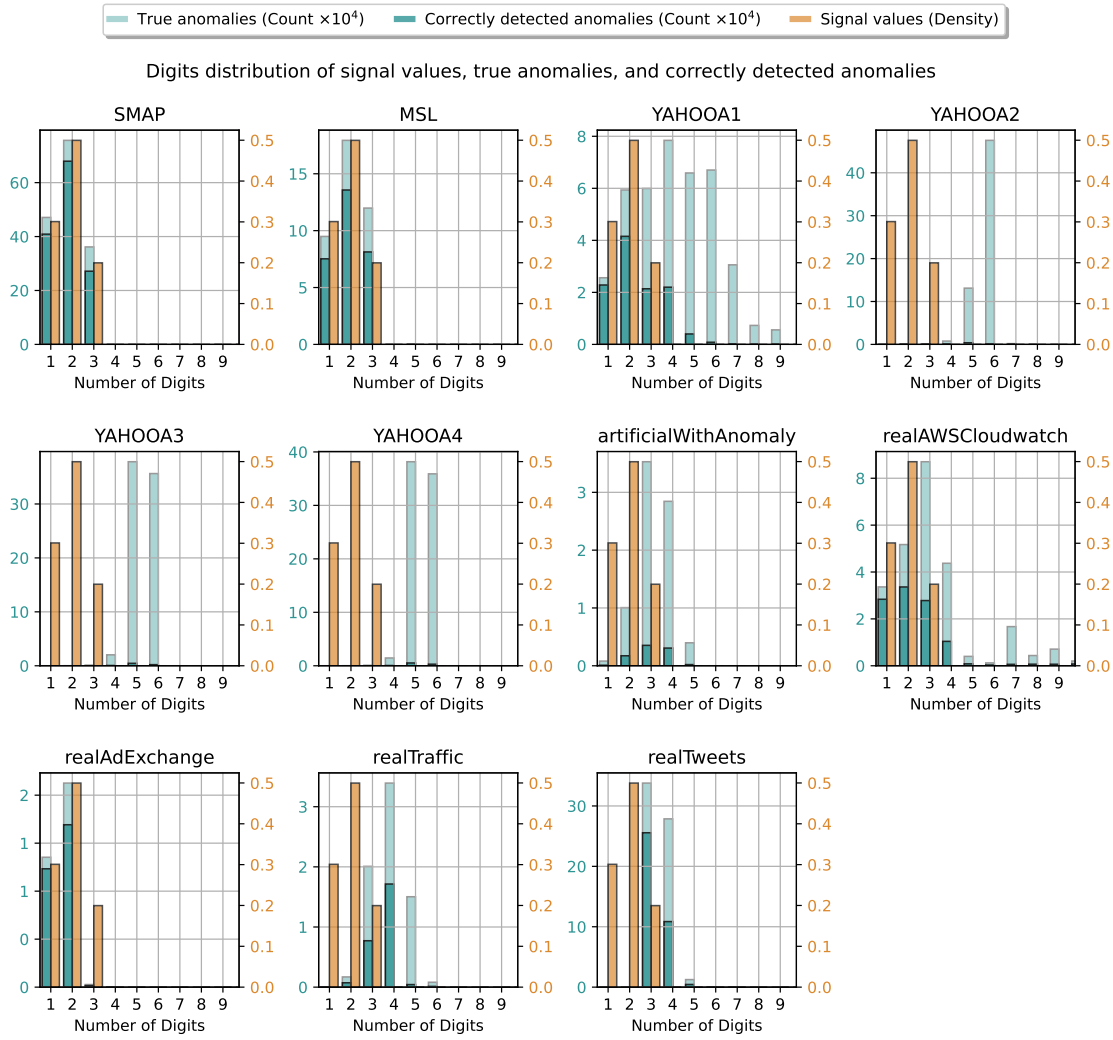

Figure A.3: Digits distribution of signal values, true anomalies, and correctly detect anomalies.

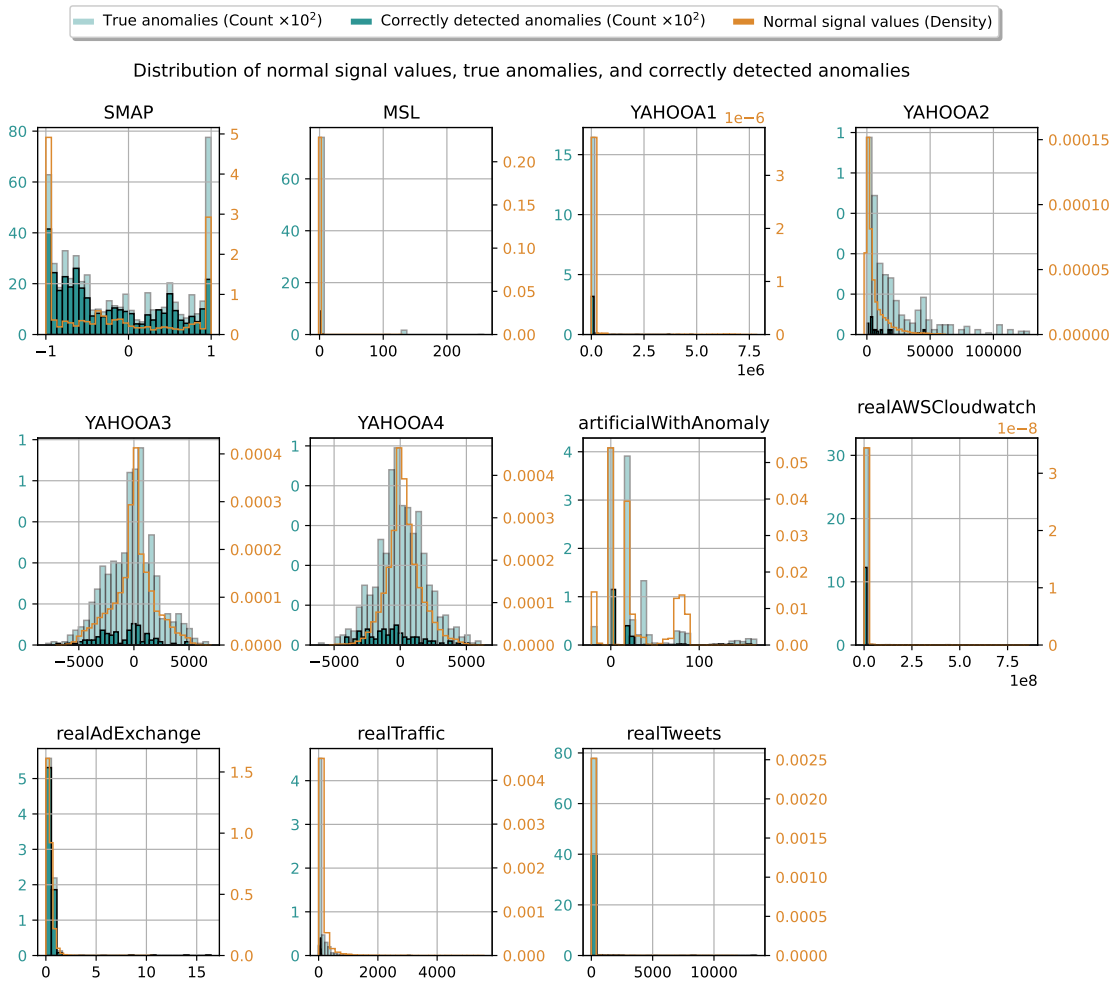

Figure A.4: Distribution of normal signal values, true anomalies, and correctly detected anomalies.

References

[1] S. Alnegheimish, L. Nguyen, L. Berti-Equille, and K. Veeramachaneni, *Large language models can be zero-shot anomaly detectors for time series?* 2024. arXiv: 2405.14755 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2405.14755.

[2] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2014. DOI: 10.1109/TKDE.2013.184.

[3] W. A. Chaovalitwongse, Y.-J. Fan, and R. C. Sachdeo, "On the time series K-nearest neighbor classification of abnormal brain activity," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 37, no. 6, pp. 1005–1016, 2007. DOI: 10.1109/TSMCA.2007.897589.

[4] R. Zhang, S. Zhang, S. Muthuraman, and J. Jiang, "One class support vector machine for anomaly detection in the communication network performance data," in *Proceedings of the 5th Conference on Applied Electromagnetics, Wireless and Optical Communications*, ser. ELECTROSCIENCE'07, Tenerife, Canary Islands, Spain: World Scientific, Engineering Academy, and Society (WSEAS), 2007, pp. 31–37, ISBN: 9789606766251.

[5] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[6] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'19/IAAI'19/EAAI'19, Honolulu, Hawaii, USA: AAAI Press, 2019, ISBN: 978-1-57735-809-1. DOI: 10.1609/aaai.v33i01.33011409. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33011409.

[7] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19, Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2828–2837, ISBN: 9781450362016. DOI: 10.1145/3292500.3330672. [Online]. Available: https://doi.org/10.1145/3292500.3330672.

[8] Y. Yuan, G. Xun, F. Ma, Y. Wang, N. Du, K. Jia, L. Su, and A. Zhang, "Muvan: A multi-view attention network for multivariate temporal data," in *2018 IEEE International Conference on Data Mining (ICDM)*, 2018, pp. 717–726. DOI: 10.1109/ICDM.2018.00087.

[9] V. Jacob, F. Song, A. Stiegler, B. Rad, Y. Diao, and N. Tatbul, *Exathlon: A benchmark for explainable anomaly detection over time series*, 2021. arXiv: 2010.05073 [cs.LG].

[10] A. Das, W. Kong, R. Sen, and Y. Zhou, *A decoder-only foundation model for time-series forecasting*, 2024. arXiv: 2310.10688 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.10688.

[11] K. Rasul et al., "Lag-Llama: Towards foundation models for time series forecasting," *arXiv preprint arXiv:2310.08278*, 2023.

[12] S. Alnegheimish, "Orion–a machine learning framework for unsupervised time series anomaly detection," Ph.D. dissertation, Massachusetts Institute of Technology, 2022.

[13] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, pp. 85–126, Oct. 2004. DOI: 10.1023/B:AIRE.0000045502.10941.a9.

[14] D. Zheng, F. Li, and T. Zhao, "Self-adaptive statistical process control for anomaly detection in time series," *Expert Syst. Appl.*, vol. 57, no. C, pp. 324–336, Sep. 2016, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2016.03.029. [Online]. Available: https://doi.org/10.1016/j.eswa.2016.03.029.

[15] E. Hannan, *Multiple Time Series* (Wiley Series in Probability and Statistics). Wiley, 2009, ISBN: 9780470317136. [Online]. Available: https://books.google.com.vn/books?id=R5IB1Vja3j4C.

[16] E. H. Pena et al., "Anomaly detection using forecasting methods ARIMA and HWDS," in *2013 32nd International Conference of the Chilean Computer Science Society (sccc)*, IEEE, 2013, pp. 63–66.

[17] N. P. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:207227428.

[18] K. Hundman et al., "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 387–395.

[19] N. Ding, H. Gao, H. Bu, H. Ma, and H. Si, "Multivariate-time-series-driven real-time anomaly detection based on bayesian network," *Sensors*, vol. 18, no. 10, 2018, ISSN: 1424-8220. DOI: 10.3390/s18103367. [Online]. Available: https://www.mdpi.com/1424-8220/18/10/3367.

[20] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '18, ACM, Jul. 2018. DOI: 10.1145/3219819.3219845. [Online]. Available: http://dx.doi.org/10.1145/3219819.3219845.

[21] P. Malhotra et al., "LSTM-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[22] D. Park et al., "A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1544–1551, 2018.

[23] A. Geiger, D. Liu, S. Alnegheimish, A. Cuesta-Infante, and K. Veeramachaneni, "Tadgan: Time series anomaly detection using generative adversarial networks," in *2020 IEEE International Conference on Big Data (IEEE BigData)*, IEEE, 2020, pp. 33–43. DOI: 10.1109/BigData50022.2020.9378139.

[24] G. Song, *Modeling control signals for reconstruction-based time series anomaly detection*, 2024.

[25] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017.

[26] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato,

R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.

[27] H. Touvron, L. Martin, K. Stone, *et al.*, *Llama 2: Open foundation and fine-tuned chat models*, 2023. arXiv: 2307.09288 [cs.CL].

[28] J. Wei, Y. Tay, R. Bommasani, *et al.*, *Emergent abilities of large language models*, 2022. arXiv: 2206.07682 [cs.CL].

[29] M. Hahn and N. Goyal, *A theory of emergent in-context learning as implicit structure induction*, 2023. arXiv: 2303.07971 [cs.CL].

[30] T. B. Brown, B. Mann, N. Ryder, *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[31] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, 2020. arXiv: 1912.09363 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1912.09363.

[32] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, *A transformer-based framework for multivariate time series representation learning*, 2020. arXiv: 2010.02803 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2010.02803.

[33] S. Tuli et al., "Tranad: Deep transformer networks for anomaly detection in multivariate time series data," *Proc. VLDB Endow.*, vol. 15, no. 6, pp. 1201–1214, Feb. 2022, ISSN: 2150-8097. DOI: 10.14778/3514061.3514067.

[34] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, *Transformers in time series: A survey*, 2023. arXiv: 2202.07125 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2202.07125.

[35] S. Dooley et al., "Forecastpfn: Synthetically-trained zero-shot forecasting," in *Advances in Neural Information Processing Systems*, vol. 36, Curran Associates, Inc., 2023, pp. 2403–2426.

[36] A. F. Ansari et al., "Chronos: Learning the language of time series," *arXiv preprint arXiv:2403.07815*, 2024.

[37] S. Gao, T. Koker, O. Queen, T. Hartvigsen, T. Tsiligkaridis, and M. Zitnik, *Units: A unified multi-task time series model*, 2024. arXiv: 2403.00131 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2403.00131.

[38] N. Gruver, M. Finzi, S. Qiu, and A. G. Wilson, "Large language models are zero-shot time series forecasters," *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[39] H. Xue and F. D. Salim, *Promptcast: A new prompt-based learning paradigm for time series forecasting*, 2023. arXiv: 2210.08964 [stat.ME].

[40] A. Q. Jiang et al., *Mistral 7b*, 2023. arXiv: 2310.06825 [cs.CL].

[41] M. Müller, "Dynamic time warping," *Information retrieval for music and motion*, pp. 69–84, 2007.

[42] L. Wong, D. Liu, L. Berti-Equille, S. Alnegheimish, and K. Veeramachaneni, "Aer: Auto-encoder with regression for time series anomaly detection," in *2022 IEEE International Conference on Big Data (IEEE BigData)*, IEEE, 2022, pp. 1152–1161. DOI: 10.1109/BigData55660.2022.10020857.

[43] T. Liu and B. K. H. Low, *Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks*, 2023. arXiv: 2305.14201 [cs.LG].

[44] S. Alnegheimish, D. Liu, C. Sala, L. Berti-Equille, and K. Veeramachaneni, "Sintel: A machine learning framework to extract insights from signals," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22, Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 1855–1865, ISBN: 9781450392495. DOI: 10.1145/3514221.3517910. [Online]. Available: https://doi.org/10.1145/3514221.3517910.

[45] N. Tatbul et al., "Precision and recall for time series," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018.

[46] K. Chang et al., "Speak, memory: An archaeology of books known to ChatGPT/GPT-4," in *EMNLP 2023*. DOI: 10.18653/v1/2023.emnlp-main.453.

[47] S. Biderman et al., "Emergent and predictable memorization in large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[48] M. J. Smith, C. Sala, J. M. Kanter, and K. Veeramachaneni, "The machine learning bazaar: Harnessing the ml ecosystem for effective system development," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20, Portland, OR, USA: Association for Computing Machinery, 2020, pp. 785–800, ISBN: 9781450367356. DOI: 10.1145/3318464.3386146. [Online]. Available: https://doi.org/10.1145/3318464.3386146.

[49] Xue, "A flexible framework for composing end to end machine learning pipelines," Ph.D. dissertation, Jan. 2018.

[50] Santiago, *Machine Learning Blocks*. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2015. [Online]. Available: https://books.google.com.vn/books?id=18qdjwEACAAJ.

[51] M. J. Smith, C. Sala, J. M. Kanter, and K. Veeramachaneni, "The machine learning bazaar: Harnessing the ml ecosystem for effective system development," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD/PODS '20, ACM, May 2020, pp. 785–800. DOI: 10.1145/3318464.3386146. [Online]. Available: http://dx.doi.org/10.1145/3318464.3386146.

[52] J. Xu et al., "Anomaly Transformer: Time series anomaly detection with association discrepancy," in *International Conference on Learning Representations*, 2022.

[53] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "M5 accuracy competition: Results, findings, and conclusions," *International Journal of Forecasting*, vol. 38, no. 4, pp. 1346–1364, 2022, Special Issue: M5 competition, ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2021.11.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207021001874.

[54] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, *Informer: Beyond efficient transformer for long sequence time-series forecasting*, 2021. arXiv: 2012.07436 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2012.07436.

[55] S. B. Taieb, G. Bontempi, A. Atiya, and A. Sorjamaa, *A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition*, 2011. arXiv: 1108.3259 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1108.3259.

[56] A. Trindade, *ElectricityLoadDiagrams20112014*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C58C86, 2015.

[57] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, *Modeling long- and short-term temporal patterns with deep neural networks*, 2018. arXiv: 1703.07015 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1703.07015.

[58] Caltrans, *Pems.traffic.*

[59] G. Woo, C. Liu, D. Sahoo, A. Kumar, and S. Hoi, *Etsformer: Exponential smoothing transformers for time-series forecasting*, 2022. arXiv: 2202.01381 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2202.01381.

[60] C. Liu, D. B. Springer, Q. Li, *et al.*, "An open access database for the evaluation of heart sound algorithms," *Physiological Measurement*, vol. 37, pp. 2181–2213, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:22101272.

[61] M. Kudo, J. Toyama, and M. Shimbo, "Multidimensional curve classification using passing-through regions," *Pattern Recognition Letters*, vol. 20, no. 11, pp. 1103–1111, 1999, ISSN: 0167-8655. DOI: https://doi.org/10.1016/S0167-8655(99)00077-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016786559900077X.

[62] M. Cuturi, "Fast global alignment kernels," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11, Bellevue, Washington, USA: Omnipress, 2011, pp. 929–936, ISBN: 9781450306195.

[63] N. Birbaumer, N. Ghanayim, T. Hinterberger, I. H. Iversen, B. Kotchoubey, A. Kübler, J. Perelmouter, E. Taub, and H. Flor, "A spelling device for the paralysed," *Nature*, vol. 398, pp. 297–298, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:204991968.

[64] M. Bedda and N. Hammami, *Spoken Arabic Digit*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C52C9Q, 2008.

[65] J. Liu, Z. Wang, L. Zhong, J. Wickramasuriya, and V. Vasudevan, "Uwave: Accelerometer-based personalized gesture recognition and its applications," in *2009 IEEE International Conference on Pervasive Computing and Communications*, 2009, pp. 1–9. DOI: 10.1109/PERCOM.2009.4912759.

[66] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet," *Circulation*, vol. 101, no. 23, e215–e220, 2000. DOI: 10.1161/01.CIR.101.23.e215. eprint: https://www.ahajournals.org/doi/pdf/10.1161/01.CIR.101.23.e215. [Online]. Available: https://www.ahajournals.org/doi/abs/10.1161/01.CIR.101.23.e215.

[67] I. Silva, J. Behar, R. Sameni, T. Zhu, J. Oster, G. D. Clifford, and G. B. Moody, "Noninvasive fetal ecg: The physionet/computing in cardiology challenge 2013," in *Computing in Cardiology 2013*, 2013, pp. 149–152.

[68] K. O. Chicaiza and M. E. Benalcázar, "A brain-computer interface for controlling iot devices using eeg signals," in *2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM)*, 2021, pp. 1–6. DOI: 10.1109/ETCM53643.2021.9590711.

[69] R. N. A. Henson, D. G. Wakeman, V. Litvak, and K. J. Friston, "A parametric empirical bayesian framework for the eeg/meg inverse problem: Generative models for multi-subject and multi-modal integration," *Frontiers in Human Neuroscience*, vol. 5, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:7560846.

[70] J. Lines, A. Bagnall, P. Caiger-Smith, and S. Anderson, "Classification of household devices by electricity usage profiles," in *Proceedings of the 12th International Conference on Intelligent Data Engineering and Automated Learning*, ser. IDEAL'11, Norwich, UK: Springer-Verlag, 2011, pp. 403–412, ISBN: 9783642238772.

[71] D. Roverso, "Plant diagnostics by transient classification: The aladdin approach," *International Journal of Intelligent Systems*, vol. 17, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:10733207.

[72] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, *The ucr time series archive*, 2019. arXiv: 1810.07758 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1810.07758.

[73] M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi, "Mobile sensor data anonymization," in *Proceedings of the International Conference on Internet of Things Design and Implementation*, ser. IoTDI '19, ACM, Apr. 2019. DOI: 10.1145/3302505.3310068. [Online]. Available: http://dx.doi.org/10.1145/3302505.3310068.

[74] J. O. Egede, S. Song, T. A. Olugbade, C. Wang, A. Williams, H. Meng, M. Aung, N. D. Lane, M. Valstar, and N. Bianchi-Berthouze, *Emopain challenge 2020: Multimodal pain evaluation from facial and bodily expressions*, 2020. arXiv: 2001.07739 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2001.07739.

[75] M. Middlehurst, P. Schäfer, and A. Bagnall, "Bake off redux: A review and experimental evaluation of recent time series classification algorithms," *Data Mining and Knowledge Discovery*, vol. 38, no. 4, pp. 1958–2031, Apr. 2024, ISSN: 1573-756X. DOI: 10.1007/s10618-024-01022-1. [Online]. Available: http://dx.doi.org/10.1007/s10618-024-01022-1.