Machine Learning Systems for Unsupervised Time Series Anomaly Detection

by
Sarah Alnegheimish

B.S., King Saud University (2017)
S.M., Massachusetts Institute of Technology (2022)

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

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2025

© 2025 Sarah Alnegheimish. All rights reserved.
The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by:	Sarah Alnegheimish Department of Electrical Engineering and Computer Science August 15, 2025
Certified by:	Kalyan Veeramachaneni Principal Research Scientist Laboratory for Information and Decision Systems Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Students

Machine Learning Systems for Unsupervised Time Series Anomaly Detection

by

Sarah Alnegheimish

Submitted to the Department of Electrical Engineering and Computer Science
on August 15, 2025, in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract

Modern assets – from launched satellites to electric vehicles – output dense, multivariate time series data that must be monitored for deviations from "normal" behavior. This monitoring task is referred to as time series anomaly detection. The current state of the industry still depends on fixed or heuristic thresholds that often drown operators in false alarms, and can miss the subtle, context-dependent faults that matter most. This thesis addresses unsupervised time series anomaly detection as an end-to-end problem, asking how we can learn, evaluate, and deploy models that judiciously flag anomalies while remaining intuitive to the end user.

This thesis provides contributions in the form of both algorithms and systems. First, it introduces three models that enlarge the design space of unsupervised time series anomaly detection: `TadGAN`, which leverages adversarial reconstruction; `AER`, which unifies predictive and reconstructive objectives in a single hybrid score; and `MixedLSTM`, which explicitly incorporates interdependencies to improve anomaly detection in multivariate time series. We propose two range-based evaluation metrics that quantify detection quality over temporal intervals. Second, it presents our system Orion, which abstracts anomaly detection pipelines as directed acyclic graphs of reusable primitives, providing user-friendly APIs and enabling interactive visual inspection. Building on this infrastructure, OrionBench performs periodic, fully reproducible benchmarks, producing leaderboards that align research innovations with the needs of end users. Third, the thesis explores a new paradigm – foundation models for unsupervised time series anomaly detection – by formulating SigLLM, which employs large language models and time series foundation models for zero-shot anomaly detection via prompting and forecasting. This paradigm indicates a promising path to developing scalable models for anomaly detection. Finally, beyond evaluating our systems on publicly available datasets, we provide extensive experiments on two industrial case studies that demonstrate improved detection accuracy and practical usability of our system.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist
Laboratory for Information and Decision Systems

Acknowledgments

الحمدلله أولاً وآخراً

I dedicate this thesis to my family, without whom I could not have completed this journey. This work would also not have been possible without the many people I met who generously invested their time in our collaborations and conversations.

I owe my deepest gratitude to my advisor, Kalyan Veeramachaneni. His mentorship reshaped how I view research – encouraging me to look beyond results displayed in a table and pursue work that delivers meaningful impact across industries. Kalyan's guidance taught me to lead with bold ideas while never losing sight of the broader vision. His unwavering confidence in me, especially when my own faltered, was invaluable. His genuine care for his students' well-being sustained my motivation throughout this journey. The person I am today is a direct result of his continuous academic and personal support. I feel truly fortunate to have completed my Ph.D. under his supervision and look forward to continuing to work together in the years ahead.

To my thesis committee members, Saman Amarasinghe and Martin Rinard, thank you for being part of my committee. Your insightful comments and constructive feedback greatly strengthened this work, and I will remain deeply grateful for your continued support beyond the thesis itself. I also wish to thank my academic advisor, Aleksander Mądry, for his guidance and helpful advice throughout my graduate journey.

I would like to extend my gratitude to Laure Berti-Équille and Dongyu Liu for our collaborations, especially during my early years as a researcher. Their guidance taught me how to navigate the research community, and I appreciate all the comments they have provided on our papers.

To my industry collaborators, thank you for the countless meetings that deepened my understanding of real-world challenges (and sponsoring our research). Thank you to SES, ScottishPower Renewables (Sofia Koukoura and Robert Jones), Hyundai Motor Company (Sungjae Kim), and GE. I also wish to acknowledge my collaborators at Amazon: Matthew Reimherr, Zelin He, Akash Chandrayan, and Abhinav Pradhan.

To the MEng students I have had the pleasure of mentoring – Grace Song and Linh Nguyen – you have done incredible work, and I have learned from your meticulous attention to detail and thought-provoking writing. You have also taught me how to offer guidance most effectively, thank you.

Many individuals working behind the scenes made this research possible. I am grateful to Brian Jones for promptly helping me whenever I encountered issues with the lab machines. I am also thankful to the editors, designers, and staff – especially Cara Giaimo, Arash Akhgari, and Katie O'Reilly – for their support of both me and my fellow researchers.

Members of the Data to AI Lab, your camaraderie and creativity have been a constant source of inspiration. Thank you for your thoughtful discussions and for sharing a workspace over the years. It also gave us the opportunity to collaborate, Ola Zytek, Lei Xu, Alicia Sun, Sara Pidò, Frances Hartwell, Salim Cherkaoui, and many others, thank you for all of it.

Coming to MIT has felt like winning the lottery, and it would not have been possible without the Center for Complex Engineering and Systems (CCES) at King Abdulaziz City for Science and Technology (KACST). Led by Anas Alfaris, CCES nurtures ambitious young talent, fosters collaborative research with renowned scholars, and supports students in pursuing graduate studies at leading institutions. I am immensely grateful to CCES and to all the researchers and mentors who guided me during this formative time.

To my friends who have filled the past several years with laughter and unforgettable memories, thank you for always being there, for making this place feel like a second home, for sharing in my joys, and for lifting my spirits during difficult times. I am especially thankful that MIT introduced me to Yasmeen Alfaraj, a fellow traveler pursuing her Ph.D. in Chemistry. Her discipline, tenacity, and boundless curiosity inspired me, and our shared journey through graduate school became one of my treasured experiences. I learned deeply from her connectedness to people and ideas, and her care, support, and presence sustained me when I needed it most. To my friends back home, thank you for always making time for me and for turning every visit into a memory I cherish.

I am incredibly fortunate to have actual family in Boston, let alone at MIT – Mohammed Alsobay, Ghada Alwabil, and Hadeel. I cannot imagine how difficult this journey would have been without you. Even while navigating your own graduate studies and responsibilities, you always found time to support me and one another. I will cherish every Mario Party game we played (even when I lost) as a much needed break from graduate life.

I would like to also thank my extended family, who never fail to check in on me, surround me with laughter, and fill our time together with warmth and joy. You have made every visit special, especially August 16$^{\text{th}}$, 2024. I cannot wait for the next era.

My parents, Abdulaziz and Iman, words cannot express the depth of my gratitude. Your love and support have been a constant source of strength. Thank you for raising us to be curious, considerate, and resilient. Your belief in the value of education has guided my siblings and me throughout our lives. Where I am today is a testament to your sacrifices and hard work. To my siblings: Hadeel, Norah, Mohammed, Yasmeen, and Ibrahim, I am endlessly grateful for the ways you have supported me. Thank you for always grounding me, whether from thousands of kilometers away or just a few steps down the hall. I look up to each of you, and your encouragement (be it in person or through a well-timed tiktok video) has meant the world to me. Thank you for sharing in my passions, from football to tennis, and for making me feel understood in every way. I could not imagine a more supportive family. To Hadeel, especially, thank you for being a very supportive sister, for accompanying me to conferences around the world, and for reading my papers (including this thesis!).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Time series are everywhere. Over the past few decades, sensor-based monitoring of both large and small assets has become more popular, driven by improved sensor technology, advances in data collection, bandwidth availability for transient data, and increasingly accessible cloud storage. This has greatly increased the overall prevalence of time series data across fields and industries. This data can be monitored for what we call "anomalies" – indications of abnormal behaviors. Once flagged, these anomalies can be investigated, both to prevent bad outcomes and to improve systems overall. For example, in computer networking, unusual patterns in traffic data can indicate cybersecurity threats [Ahmed et al., 2016]. In industrial settings, abnormal sensor readings in heavy machinery may point to current or future failures; anomaly detection can help engineers predict these failures and/or optimize operations by changing control settings [Cook et al., 2020]. In healthcare, vital sign monitoring and wearable devices can reduce patients' health complications [Salem et al., 2013, Sunny et al., 2022]. In finance, anomaly detection can help identify fraudulent credit card transactions based on purchase history [Hilal et al., 2022]. Increasing reliance on time series monitoring has underscored the need for reliable anomaly detection systems [1] – namely, ones that judiciously flag anomalies and are usable by end users [2].

[1] The focus of this thesis is unsupervised time series anomaly detection. Throughout, we refer to *unsupervised time series anomaly detection*, *time series anomaly detection*, and *anomaly detection* interchangeably.

[2] End users are defined as people who are interested in using a model on their own data in order to find anomalies. We differentiate these users from ML researchers and ML developers, who are interested in inventing or developing new ML methods. Throughout this thesis, we use the terms *end users* and *users* interchangeably.

Chapter Outline

In this chapter:

- We describe the current state of time series anomaly detection in Section 1.1.
- We introduce unsupervised time series anomaly detection in Section 1.2.
- We identify challenges in recent research in Section 1.3.
- We propose alleviations for these challenges in Section 1.4.
- We present our contributions in Section 1.5.
- We lay out the organization of the rest of the thesis in Section 1.6.

1.1 Time Series Anomaly Detection

Assets deployed in the field, such as heavy machinery, are typically equipped with multiple sensors and operate under varying control settings. These systems continuously generate multivariate time series that reflect their operational states over time. As the assets interact with their environment and respond to external conditions, they may occasionally exhibit "unusual" behaviors, indicated within the time series by data anomalies. Detecting anomalous behaviors is essential for discovering valuable events, optimizing operations, reducing downtime, and preempting failures. When anomalies are identified, experts can investigate and intervene as necessary. For example, in retail, detecting a sudden surge in sales that goes beyond normal seasonal trends can help a company quickly restock inventory. It can also indicate positive returns for the marketing campaign that caused the spike.

Current State. To ground ourselves in an example, we examine how a leading satellite operations company approaches anomaly detection for telemetry time series:

1. **Visual monitoring:** Sensor streams are plotted live on dashboards, where experts and engineers can scan trends and manually flag anything suspicious.

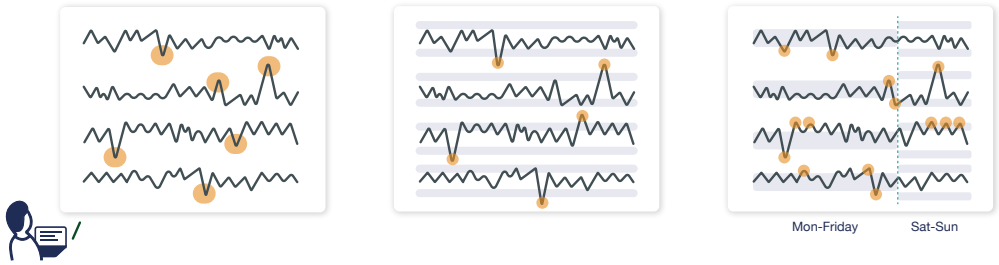


Figure 1-1: Current state of time series monitoring rely on thresholding mechanisms.

2. **Fixed thresholds:** Alarms are raised whenever a measurement exceeds a predetermined upper or lower limit. Many such events are actually harmless, generating a high volume of false alarms that still require expert review.

3. **Heuristic-driven thresholds:** To reduce false alarms, engineers replace static limits with rule-based, adaptive thresholds; for example, different thresholds for weekdays versus weekends. While this approach improves precision, the growing rulebook soon becomes complex and hard to maintain.

Machine Learning Model Requirements. This poses the question of whether we can use machine learning to learn the normal patterns of the data, and stop relying on these thresholds. For machine learning to be successful here, we define a set of requirements:

R1: Judicious Alarms. Properly developed anomaly detection methods must find abnormal sequences in time series while triggering few to no false alarms, as false alarms spur costly investigations. This precision also builds trust among operation teams.

R2: Feedback. Flagged events must be easy for domain experts to confirm as anomalous or dismiss as unconcerning. Interactive visualizations that overlay detected anomalies on raw signals are essential for this validation process.

R3: Explanations. When there is an alarm, users need to understand *why*. Rule-based systems provide this level of transparency by design, so machine learning approaches must offer comparable interpretability.

This thesis focuses on developing these models (R1) and presenting them to the end user (R2). We lightly touch on the interpretability of models, and suggest future work using large language models as post-hoc explainers (R3).

1.2 Unsupervised Time Series Anomaly Detection

Unsupervised time series anomaly detection refers to the process of automatically analyzing time series data to identify deviations without relying on labeled anomalies for training. These methods aim to model "normal" behavior from historical observations, and to flag patterns that deviate significantly from learned expectations. Researchers have been developing these methods for decades [Hodge and Austin, 2004, Chandola et al., 2009, Gupta et al., 2013]. Given the scarcity and high cost of labeled anomaly data, unsupervised approaches are often preferred.

Why not supervised? With supervised learning, models learn patterns from human-labeled data and then use those patterns to detect other anomalies. Training these models requires previously labeled events, which are difficult for humans to find. Moreover, anomalous patterns are constantly changing, making it more challenging for models trained on a limited set of labeled anomalies to make useful predictions. These models struggle to find "new" events that are interesting to the user. In contrast, with unsupervised learning, no ground truth is given to the model, revealing anomalies that may have otherwise gone unseen. This property is highly valuable, since human experts are often unable to determine *what* they are looking for or *when* it will occur (what we refer to as the "unknown-unknowns" issue). Since unsupervised models flag any intervals that deviate from what is expected, these models can help finding "unknown-unknowns."

1.2.1 Anomaly Detection with Traditional Methods

Single-Table Methods. We distinguish between algorithms that work on data presented in tabular formats, called *single-table*, and algorithms that directly work with raw time series. Single-table methods focus on a single asset and one time series pertaining to that asset. This time series is then formatted into a tabular format such that we can classify whether each row has an anomaly or not. We summarize the major restrictions of single-table-based methods compared to time-series methods in Table 1.1.

Segmenting a time series into a "single-table" representation removes the natural ordering of observations, so the model no longer directly "sees" the data's sequential structure.

Attribute	Single-Table Models	Time Series Models
Sequential Data	converted to tabular	kept as sequential
Multiple Time Series	one at a time	can be multivariate
Temporal Dependencies	neglected	captured

Table 1.1: Distinction between single-table and time series models.

Because this tabular view fixes every row to a preset window length, it also limits how well the model can capture temporal dependencies. Finally, even if the asset has several related signals, each must be either concatenated into an even wider table, or modeled separately, making multiple time series impractical to handle. In contrast, time-series models preserve order, learn dependencies over arbitrary horizons, and accommodate many correlated series without such contortions.

Rule-based systems have been a popular approach in this category. They're built on domain heuristics and have been widely used since their inception [Chen et al., 1990, Mukherjee et al., 1994]. These systems are simple to understand and easy to implement, making them popular for deployment. Other methods focus on classification algorithms to learn the normal segments of the data, such as one-class SVMs [Manevitz and Yousef, 2001, Zhang et al., 2007]

Time Series Methods. More recently, more sophisticated algorithms have been proposed, including distance, density, and isolation-based approaches [Chaovalitwongse et al., 2007, Liu et al., 2008, Berndt and Clifford, 1994]. Moreover, statistical methods such as ARIMA, Markov-Chains, and HMMs have been used to learn the temporal features of the data [Box and Pierce, 1970, Ye et al., 2000, Zucchini and MacDonald, 2009]. As data has become increasingly large, more complex, and multidimensional, traditional methods have begun to perform less competitively.

The aforementioned traditional approaches either neglect the temporal aspect of time series, cannot generalize for multivariate time series, or require labeled data to properly set the algorithm's parameters (which, as previously mentioned, tends to be limited or unavailable). This has created room for unsupervised learning algorithms to flourish.

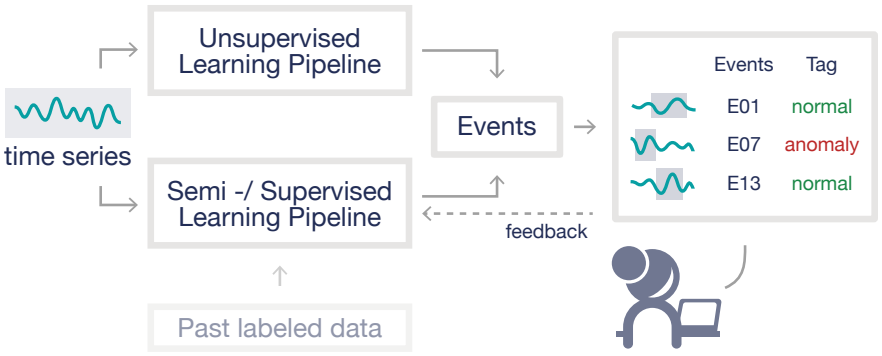


Figure 1-2: Workflow of time series anomaly detection with a closed loop.

1.2.2 Anomaly Detection with Deep Learning

Deep learning-based methods have garnered increased attention in the past several years [Zhang et al., 2019, Su et al., 2019, Yuan et al., 2018, Jacob et al., 2021]. They have gained popularity partly thanks to their ability to learn their own representations by extracting temporal and spatial features, with little to no manual engineering. Moreover, these methods can handle long and multivariate dependencies and scale with large data volumes, making them the prevailing choice in state-of-the-art time series research and applications. These models are capable of learning time series patterns in order to identify abnormal behavior.

1.2.3 Significance of Unsupervised Models

With unsupervised models, events are flagged and presented to a human expert for further investigation. After carefully inspecting an event, experts can verify whether it is a true anomaly – a problem they are interested in finding – or actually a normal occurrence that should not be flagged.

Over time, anomalies are accrued that have been carefully investigated and annotated by experts. This creates an opportunity to train semi– or fully-supervised models on past labeled data. Further annotations can be given to a supervised model as feedback, allowing it to continue to learn from new events. Given the ambiguity of what is considered anomalous and the constant change in data patterns, we still require an unsupervised model to work in parallel and suggest "new" anomalies that have not been seen before. We depict this workflow in Figure 1-2.

Given the criticality of unsupervised methods in judiciously flagging anomalies, in this thesis, we focus purely on unsupervised models and enclosing them in *usable systems*.

1.3 Challenges Encountered with SOTA Models

Anomaly detection in time series is a long-standing problem, with more models being published everyday. In 2024 alone, over 109 papers proposing a new unsupervised anomaly detection method for time series data were uploaded to arXiv, an average of more than 9 papers a month (see Appendix A.1 for search criteria). Challenges that exist in time series anomaly detection are now exacerbated by:

- the growing **number of assets** that must be monitored;
- the sheer **number of time series** along with high dimensionality;
- and the **amount of papers** that claim to have developed the state-of-the-art model for anomaly detection.

Such challenges may arise early on, during the modeling stage, or may be faced just prior to deployment. We begin by clearly delineating challenges that appear in industrial settings and form the gap between what is available and what is needed.

1.3.1 Barriers in Modeling

Limited context. Time series data can be categorized into multiple types. The most prevalent is *observation* time series: recorded measurements that represent the behavior of a single asset. For example, temperature is a widely recorded observation signal in heavy machinery.

Meanwhile, *control* time series record intentional changes in operating conditions that might affect the behavior of the asset. In heavy machinery, for instance, a control time series may record the operation of a switch.

Lastly, there are *status* time series, which give more information about the operating environment but do not necessarily come from a physical sensor. An example of a status

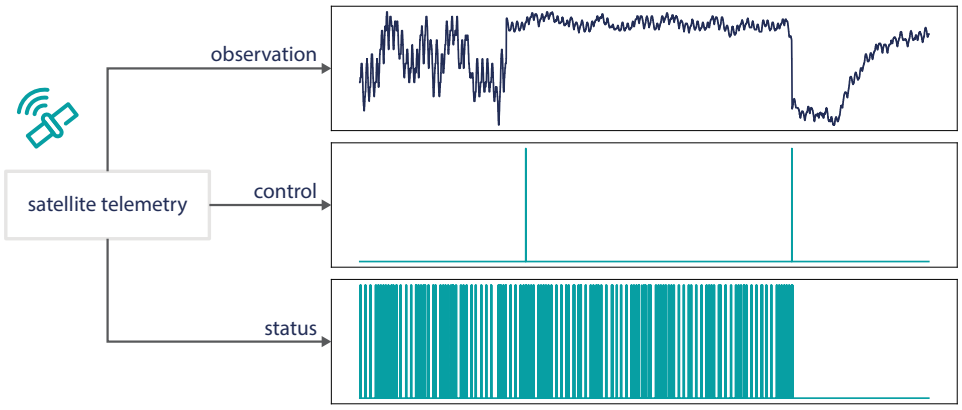


Figure 1-3: Example of `E-3` time series in satellite telemetry from SMAP. Satellites use a variety of sensors to monitor their own health, performance, and environmental conditions. Thermal sensors are used to monitor battery temperature (observation), satellite orientation is adjusted with attitude sensors (control), and the satellite's communication workload is tracked (status). In this example, the control signal clearly impacts the behavior of the observation signal.

time series is one that reflects the number of items the machine processes at a given time. This information can help identify whether the asset is under heavy operation or facing idle time. We highlight another example, regarding telemetry signals, in Figure 1-3.

Assets work differently under different control settings, and much context is buried in auxiliary signals. Many of today's time series anomaly detection models fail to distinguish well between signal types. Their task formulation assumes only observation time series will be relevant – when in reality, the incorporation of control and status signals allows for a more accurate understanding of the operation regime. Moreover, most existing models process only one signal at a time, ignoring the multivariate aspect of time series.

Unscalable learning. Time series data are sets of data points indexed by time. Each value within the data is dependent on its previous state. This temporal dependency is accompanied by unique properties such as *trend* – long-term movement in the data, *seasonality* – repeating patterns at regular intervals (e.g. weekly), and *cyclicality* – fluctuation at unspecified intervals [Granger, 1981]. Even though time series data can come from a single asset, the behavior exhibited can be quite diverse. Figure 1-4 depicts an example from a server machine dataset showing metrics such as CPU load, memory usage, and network usage [Su et al., 2019]. Therefore, to properly learn a time series pattern, it's necessary to train a

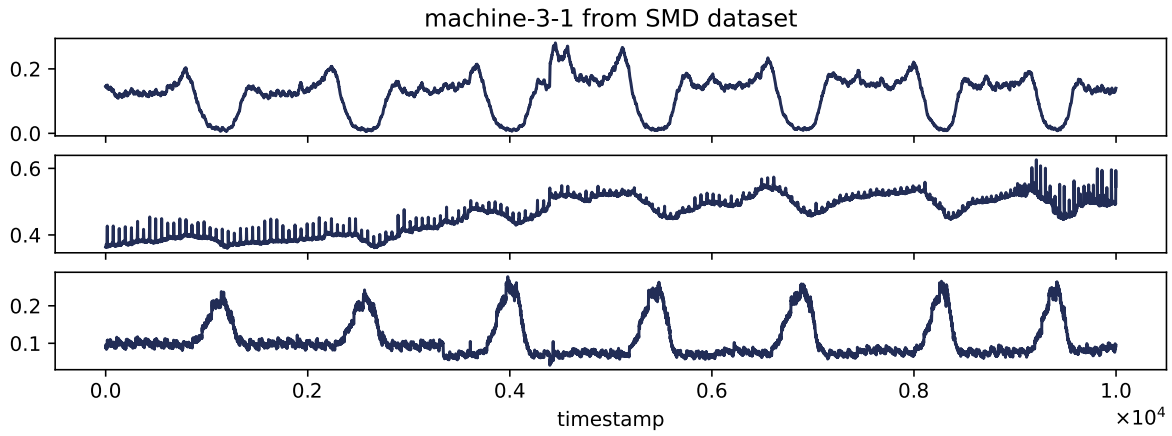


Figure 1-4: Three signals that come from the same machine, but show different patterns and trends.

model from scratch – one that learns the properties for that specific time series.

Training and deploying signal-specific and asset-specific models is not a scalable strategy. To put this challenge in perspective, if there are 10 assets that need to be monitored, each with 25 signals, this would require 250 models, all of which must be trained, deployed, and maintained. This challenge worsens as the number of time series grows and the dimensionality increases.

Ill-suited evaluations. With the wide diversity of methods available, choosing one can be difficult. This is especially hard because conventional, sample-based metrics, which are defined based on independent points within the dataset, are usually reported to be ineffective for time-based evaluations, particularly in time series anomaly detection. In time series, anomalies typically occur over temporal intervals rather than at isolated points. As a result, sample-based evaluations can lead to misleading conclusions. For example, in one study, models that correctly identified anomalous regions were penalized for having a slight temporal shift [Tatbul et al., 2018]. Moreover, model selection depends on the data, the type of anomalies, and the computational resources available. All these variables mean that no one model can outperform all others when it comes to public benchmarks. Therefore, a comprehensive evaluation is needed. Prior knowledge of the data is also required in order to find the best suited model – certain models, such as reconstruction-based models, are better at finding contextual anomalies, while prediction-based models are better at finding

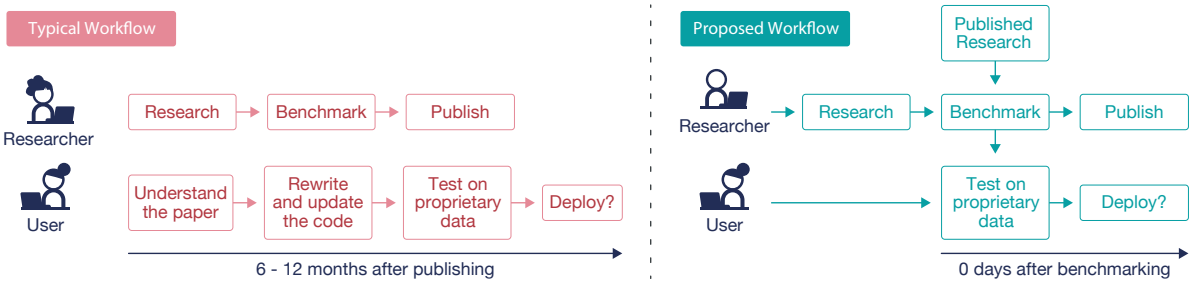


Figure 1-5: Typically, researchers and end users have independent processes. Researchers develop their method, and benchmark it in order to publish their papers. Once these methods are publicized, end users first work to understand the model, and then adapt the code to work on their own data. After a model is tested, end users decide whether it performs sufficiently well to be deployed. We propose a synchronized workflow where researchers can benchmark their pipelines and make them instantly available to end users.

point anomalies [Wong et al., 2022]. In Section 2.3 we highlight the difference between reconstruction- and prediction-based methods, and provide a thorough review of these models in Section 3.2.

1.3.2 Barriers to Adoption

Overwhelming number of published models. Rapid innovation in the machine learning space means users are inundated with new papers, all claiming state-of-the-art (SOTA) models. These models span various methodologies, from distance-based algorithms to generative models, filling the space with heterogeneity. Moreover, if users do decide to use the latest model, they often find themselves unsure of how to get started, as research papers are full of new terminology published alongside obfuscated code. When published models are not readily available, the end user needs to invest time into understanding the model.

The process of investigating, then implementing a SOTA model can be time-consuming. It often takes end users 6-12 months to decide whether or not to use a newly published method. Or worse, after taking the time to understand and implement a new model, end users may find that it does not improve on their current method. Figure 1-5(left) depicts this asynchrony between the research process and model usage.

Beyond benchmark performance. As new models for time series anomaly detection are proposed, users in industry become eager to adopt them. However, replacing existing

anomaly detection systems with these models is non-trivial, as it requires additional affordances. Beyond a model's performance reported in the paper's benchmark, adoption requires evaluating models against proprietary data, considering operational constraints, and integration with existing monitoring infrastructure. As such, organizations face a key challenge: *how can newly published models be rigorously and efficiently evaluated for their specific use case before deployment?* This question is difficult to answer, as ground truth is typically not available in proprietary data. Consequentially, users heavily rely on public datasets to gauge model performance, which may not be representative of their own use case. Moreover, additional constraints are often imposed on the model during deployment, such as a limit of alarms per day, in order to reduce the cost of investigations (especially given that fewer and more precise alarms help in gaining the trust of field operators). Lastly, industry users must be sure that the model is maintainable and will not become stale.

Separation between development and deployment processes. In many industrial organizations, the teams responsible for developing models are separate from those who deploy and operate them. This disconnect often leads to insufficient transfer of knowledge between the two teams, which can result in misaligned usage of the model. Consequently, field operators struggle to trust model outputs in practice, which prevents these models from being deployed and used. For unsupervised models in particular, the lack of intuitive outputs and explanations is a hindrance. Addressing this challenge requires not only models that perform well, but the exposure of these models in a system that can be directly used by field operators, such that they can investigate different models and test multiple examples. Moreover, it requires mechanisms for investigating model output and providing insights into how it works.

1.4 Towards Improving the Usability of Models and Systems

In the previous section, we looked at a multitude of challenges encountered during model usage and deployment. In this section, we propose attributes to help alleviate these challenges,

and our approaches for addressing them.

1.4.1 Attributes of *Usable* Models and Systems

We summarize how we address the aforementioned challenges below:

- Incorporating **contextual signals** into existing model architectures.
- Opening a new avenue of **scalable models** using pretrained models.
- Developing **range-based evaluations** that focus on time intervals rather than number of samples.
- **Standardizing** heterogeneous **models** into uniform abstractions.
- Releasing **readily-available models** and **periodic benchmarks** with top leaderboards.
- Developing **systems** with **cohesive APIs** which any user – expert or layman – can interact with seamlessly.

1.4.2 Our Approach

A1. Building context-dependent models. Deep learning-based anomaly detection models automatically learn feature representations from raw time series data, eliminating the need for manual feature engineering. However, to improve these learned representations, it is useful to condition the model on available control and status signals. Existing model architectures can often be adapted to ingest and process auxiliary signals through multi-input encoders or conditioning mechanisms. Incorporating contextual asset information can significantly improve detection performance by helping the model distinguish between benign fluctuations and true anomalies.

Approach. We propose `MixedLSTM`: an auto-encoder model with explicit incorporation of control and status signals into the model's objective function. Our design can be implemented in other models, as context signals have an independent term as part of the loss

function. We show that after adding contextual signals to `MixedLSTM` it significantly improves the detection on the MSL dataset.

A2. Inference-only modeling. The emergence of *foundation models* has introduced a new paradigm for machine learning, characterized by large-scale, pretrained models that can generalize across a wide range of tasks without additional training or fine-tuning [Bommasani et al., 2022]. Furthermore, large language models (LLMs) have showed a remarkable ability to forecast time series [Gruver et al., 2023]. This has also opened up an avenue for creating time series foundation models (TSFMs) such as TimesFM [Das et al., 2024] and TimeGPT [Garza et al., 2023]. This paradigm shift opens new possibilities for *scalable* unsupervised time series anomaly detection, as an alternative to training signal–and asset-specific models from scratch.

Approach. We investigate the utilization of LLMs and TSFMs for *zero-shot* time series anomaly detection. Specifically, for LLMs, we propose a direct prompting method to find anomalous values. For LLMs and TSFMs, we propose a forecast-dependent method for locating anomalies in time series data. We show that, while LLMs and TSFMs are capable of finding anomalies, state-of-the-art deep learning models are still superior in performance.

A3. Range-based evaluations. In traditional classification tasks, metrics such as precision, recall, and F_1 score are widely used for evaluation. However, as previously discussed, sample-based metrics are often inadequate for time-indexed data. In the context of time series anomaly detection, evaluation must account for the temporal aspect of anomalies – capturing not only whether an anomalous interval was detected, but whether the detection was timely, precise, or delayed. To ensure a meaningful assessment, it is essential to adopt a more nuanced range-based evaluation strategy. Standardizing such strategies across benchmarks will be critical for holding a fair model comparison.

Approach. To properly assess time series models for anomaly detection, we propose two evaluation strategies. The first, weighted segment, is an alternative to sample-based methods where time intervals are used as weighing factors. The second, overlapping segment, is a lenient approach that counts whether each alarm matches with an existing ground truth,

and how many ground truths are missed.

A4. Standardizing heterogeneous models. New unsupervised time series anomaly detection models are constantly being developed. We require modular and carefully designed abstractions, such that new models can apply these abstractions and existing models can be retrofitted into them. These abstractions should be able to represent any model regardless of its internal components – whether it utilizes transformers, recurrent neural networks, convolutional neural networks, or simple moving average methods. Finding universal abstractions will enable us to build a hub of readily available models.

Approach. We propose custom abstractions made up of *primitives* and *pipelines*. With these concepts, any model can be broken down into abstractions to construct an executable directed acyclic graph (DAG) with defined inputs and outputs. Furthermore, we propose Orion, a model-agnostic system with clear functionalities, such as `fit` and `detect`, that can invoke any model seamlessly. In addition, we propose SigLLM as a separate system for LLM-based detections.

A5. Periodic public benchmarks. Public benchmarks should not only display the top performing models. When they are executed periodically, moving away from *point-in-time* evaluations, benchmarks become a test of the stability of models. Moreover, as benchmarks start to include a wide variety of models and data and coverage increases, users can compare and contrast model performances across time, allowing them to select reliable models that produce stable results over time.

Approach. We propose an extension to our system Orion called OrionBench: a benchmarking framework that enables the integration of new models and datasets. OrionBench can be triggered by any user with a single command. Moreover, we frequently benchmark newly released models on pypi [3], publishing the results online for everyone to analyze.

A6. Readily available models. Benchmarks can incentivize researchers to develop the next state-of-the-art model. We propose a workflow that harmonizes between researchers and

[3] https://pypi.org/project/orion-ml/

users, where researchers can directly contribute their own models. Figure 1-5(left) illustrates how currently, researchers and end users generally operate independently, creating hurdles for end users when adopting a new published model.

Approach. With our system OrionBench, Figure 1-5(right), all researchers directly contribute their model to the benchmark. This creates a single source of readily available models for end users. This solution works together with standardizing models and creating universal abstractions.

A7. Usable and deployable systems. Field operators are integral to the success of model deployment. To move towards a deployable system, several design considerations are essential. First, the system must be *intuitive* and *user-friendly*, enabling field operators to engage with it directly without requiring extensive technical expertise. Designing simple application programming interfaces (APIs) for the system is one usability approach, evident when we look at systems such as `scikit-learn` [Buitinck et al., 2013]. Usability plays a pivotal role in facilitating long-term adoption; as operators gain more exposure, they develop familiarity with the model's behavior and can more effectively integrate its outputs into their diagnostic and maintenance workflows. Second, the system should support seamless investigation of detected events. This includes mechanisms for interpreting the model and providing explanations for why certain time segments were flagged as anomalous. Interpretable models and/or post-hoc explanation tools can help operators validate model outputs and build confidence in the system. Third, to earn and sustain field operators' trust, the system must raise alarms judiciously, minimizing false positives that lead to unnecessary investigations. Excessive false alarms not only impose operational costs, but may also cause users to disregard model outputs altogether, a phenomenon known in healthcare as "alarm fatigue" [Cvach, 2012].

Approach. For usable systems, we propose Orion and SigLLM, which provide direct API functionalities to use any model. These systems expose intermediate results to help understand the model's behavior. The detected events can be investigated and validated using MTV, which is a visual analysis system for time series and anomaly inspection. To understand the need for deployable systems, we conduct two use cases. Our first use case is with SES, a leading satellite operations company, to monitor the health conditions of

satellites. Here we worked closely with a team that monitors thousands of signals from telemetry data. Our second use case study was with a team at Hyundai Motor Company who aim to test the performance of electrical vehicles by asserting validation tests with anomaly detection methods.

1.5 Our Contributions

We summarize our contributions as follows:

- *Methods for unsupervised time series anomaly detection:* We propose new methods for unsupervised time series anomaly detection. The first is *TadGAN*, a generative adversarial model for learning normal time series patterns. The second is *AER*, which mixes prediction- and reconstruction-based models together for more accurate detection. We incorporate control and status signals in *MixedLSTM*, a reconstruction-based method with a custom loss function.

- *Range-based metrics and evaluation:* We implement two evaluation mechanisms that focus on range-based calculations and move away from sample-based evaluations. Specifically, we propose *weighted segment* and *overlapping segment* approaches for rigorous and lenient detection evaluations, respectively.

- *Systems for unsupervised time series anomaly detection:* We designed and implemented *Orion* and *SigLLM* libraries, which provide a suite of anomaly detection pipelines executable through a user-friendly interface. We provide universal abstractions of *primitives* and *pipelines* that allow us to represent any model. Pipelines include a variety of methods, including statistical, deep learning, and foundation models. The systems are supported with a benchmark continuously run on public datasets and frequent releases.

- *Foundation models for unsupervised time series anomaly detection:* We utilize pretrained models, including large language models and time series foundation models, for time series anomaly detection. We propose two paradigms for detection using

LLMs: prompter and detector. *Prompter* is a prompt engineering approach that elicits large language models to identify anomalous sections of an input. *Detector* is a forecasting approach that uses large language models as forecasting methods to find discrepancies between original and forecasted signals. Time series foundation models typically follow the forecasting approach.

- *Use cases in industry:* We apply our systems to two use cases: satellite telemetry and electric vehicles. We discuss the successes and shortcomings of our system and approaches.

Thesis publications:

The work of this thesis (and my collaborators) was published as papers summarized in Table 1.2.

1.6 Thesis Organization

The remainder of this thesis is structured as follows. Chapter 2 covers some useful background information on time series and anomaly detection in general, followed by related work in Chapter 3. We define the problem and propose our methods in Chapter 4. Chapter 5 describes the system and its components. Next, we explore the potential of foundation models in Chapter 6. We summarize our findings from working with industry teams in Chapter 7. Lastly, we conclude in Chapter 8.

Category	Paper	Citation
Models	·*Can Large Language Models be Anomaly Detectors for Time Series?*	[Alnegheimish et al., 2024b]
	·*AER: Auto-Encoder with Regression for Time Series Anomaly Detection.*	[Wong et al., 2022]
	·*TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks.*	[Geiger et al., 2020]
	·*Harnessing Vision-Language Models for Time Series Anomaly Detection.*	*pre-print* [He et al., 2025]
Systems	·*OrionBench: Benchmarking Time Series Generative Models in the Service of the End-User.*	[Alnegheimish et al., 2024a]
	·*Sintel: A Machine Learning Framework to Extract Insights from Signals.*	[Alnegheimish et al., 2022]
	·*MTV: Visual Analytics for Detecting, Investigating, and Annotating Anomalies in Multivariate Time Series.*	[Liu et al., 2022]
	·*Large Language Models to Identify and Explain Anomalies in Time Series.*	*pre-print*
Application	·*M^2AD: Detecting Anomalies in Heterogeneous Multivariate Time Series from Multiple Systems*	[Alnegheimish et al., 2025]
	·*Weakly-Supervised Multi-Sensor Anomaly Detection with Time-Series Foundation Models*	[He et al., 2024]
	·*Anomaly Detection for Electric Vehicle Data: A Case for the i-Pedal Function*	[Kim et al., 2024]

Table 1.2: Papers published as a result of this thesis.

Chapter 2

Background

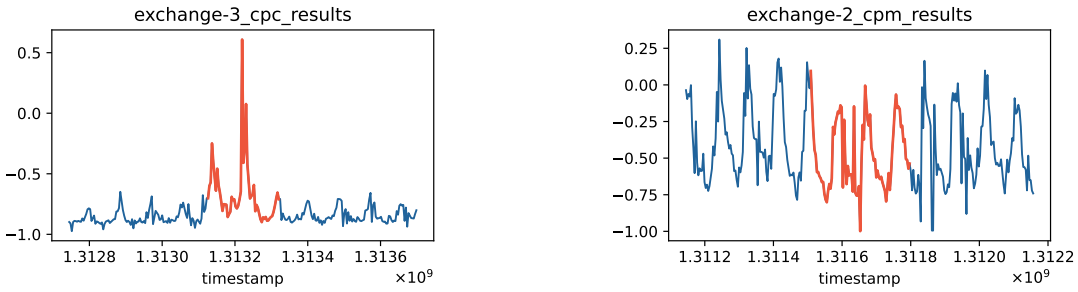
Chapter Outline:

- Section 2.1 describes the time series considered as part of the scope in this thesis.
- Section 2.2 defines what are anomalies and their type.
- Section 2.3 provides an overview of how machine learning models tackle anomaly detection.
- Section 2.4 lays out common operations for time series processing used throughout this thesis.

2.1 Time Series

A time series is defined as a sequence of observations each associated with a specific point in time. Time series data considered in this thesis focus on regularly and frequently sampled time series. For instance, measurements captured at regular, frequent intervals – such as temperature readings from thermostats or vehicle speeds obtained via wheel sensors – are within the scope of the thesis.

Conversely, event-driven time series data [Xu et al., 2019], are infrequent and irregularly sampled. An observation is recorded each time an event occurs; making the time series sparse and not consistently spaced in time. For example, entries in electronic health records



(a) out-of-range anomaly

(b) contextual anomaly

Figure 2-1: Example time series with out-of-range anomaly and contextual anomaly from `exchange_3_cpc_results` and `exchange_2_cpm_results` signals in NAB dataset.

that reflect a patient's health status at irregular visit times. Event-driven time series are beyond the scope of this thesis.

2.2 Anomalies

An anomaly in time series data is defined as a single or a sequence of data points that deviate from the expected normal behavior of the time series.

Not all anomalous events indicate faults or are problematic. The interpretation of an anomaly often requires domain-specific expertise to determine its significance. Following detection, an anomaly is investigated to establish whether it represents a genuine problem of interest, or that it is deemed as normal and can be explained under certain operating conditions. For instance, a temperature change in satellite telemetry data may initially appear abnormal, but further analysis might reveal that it corresponds to an expected external factor, such as the occurrence of a solar eclipse, which alters thermal exposure.

2.2.1 Anomaly Types

Chandola et al. [2009] classified anomalies into three categories: *point anomalies*, which are individual data points that reside outside a normal region; *contextual anomalies*, which are single instances that are considered abnormal in their local contexts; and *collective anomalies*, where an instance is not anomalous on its own, but multiple instances are collectively anomalous.

We generalize the aforementioned definitions, focusing on how the behavior of the time series changes:

- **out-of-range anomalies**, which contain extreme values with respect to the data distribution. These anomalies are considered "outliers" and can be captured through threshold-based approaches. We also use the term "point anomaly" to refer to anomalies in this category.

- **contextual anomalies**, where a sequence of values is abnormal in the immediate context. This usually manifests as unexpected changes in patterns.

Figure 2-1 shows examples of these anomaly types. Note that these definitions focus on the magnitude and pattern of the abnormal sequence, rather than whether it is comprised of individual or multiple data points.

Concretely, anomalies manifest as deviations from the expected temporal patterns, which may occur in various aspects of the signal [Chatfield and Xing, 2019, Hyndman and Athanasopoulos, 2018, Bloomfield, 2004]. Below, we list some examples of how anomalies can alter the expected pattern of the signal and illustrate them in Figure 2-2:

- **trend**: abrupt or gradual changes in the long-term directional component of the signal, including unexpected shifts in slope or intercept.

- **amplitude**: variations in the range or magnitude of oscillations, not necessarily violating the global distribution.

- **frequency**: deviations from expected periodicity, leading to changes in cycle length.

- **seasonality/phase**: temporal shifts in the expected seasonal behavior or misalignment of cycles.

- **noise**: sudden change in stochastic fluctuations that distort the shape of subsequences, potentially obscuring underlying patterns.

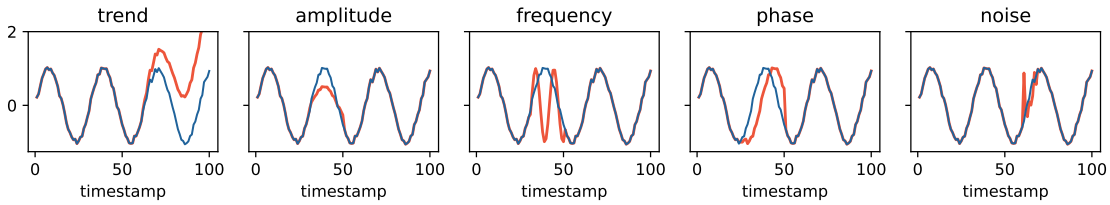


Figure 2-2: Examples showing anomalies with respect to trend, amplitude, frequency, phase, and noise properties of the signal.

2.3 Anomaly Detection using Machine Learning

Time series anomaly detection techniques have advanced significantly over the past decades, especially with the emergence of deep learning models for time series. Recent models are able to flag abnormal sequences in time series with no access to labeled data, solely by learning the temporal patterns of time signals from historical data.

In this section, we detail the general workflow of how ML is employed for anomaly detection. Moreover, we set the scope of how this thesis addresses multivariate time series.

2.3.1 General ML Usage for Anomaly Detection

Unsupervised machine learning-based anomaly detection models generally follow the sequence of steps presented in Figure 2-3. Like most machine learning methods, the process comprises two phases: training and inference, with each phase consisting of pre-processing, modeling, and post-processing steps. In the **training phase**, the focus is on learning the temporal features of the time series. This can be done within the time series *forecasting* or *reconstruction* paradigms. In the **inference phase**, we shift to finding discrepancies. We use the trained ML model to generate the expected "normal" behavior of the time series. We then compare this to the observed time series, and find the differences between what was expected and what was observed. When this discrepancy is high, this may indicate an anomaly.

Pre-processing operations include scaling the time series into a specific range, imputing missing values, and resampling the time series, while **post-processing** includes computing discrepancies between two sequences and calibrating the threshold. **Modeling** involves a wide variety of machine learning models that are trained to learn the features of the input

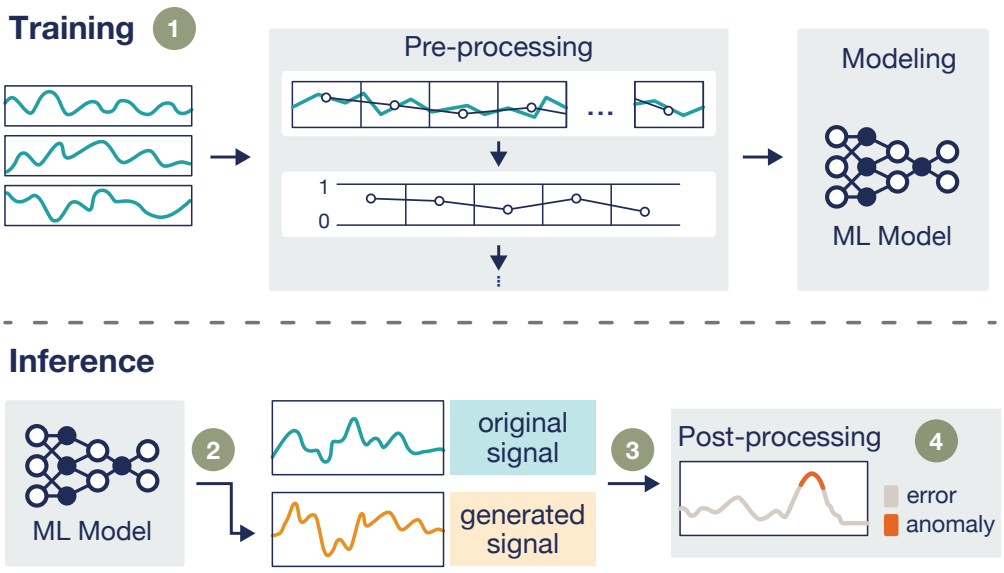


Figure 2-3: General principle of how machine learning models find anomalies in an unsupervised setting. Step 1: Apply a sequence of preprocessing operations, and train a machine learning model to learn the data pattern. This is the most time-consuming step. Step 2: Use the trained model to generate another 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.

data. In this thesis, we focus on two prominent paradigms for modeling time series using ML: forecasting and reconstruction. Forecasting models learn to predict the next value in a time series. Examples include recurrent architectures such as Long-Short-Term-Memory models (LSTMs) [Hochreiter and Schmidhuber, 1997, Hundman et al., 2018] and feature extractors using Convolutional Neural Networks (CNNs) [van den Oord et al., 2016, Borovykh et al., 2018]. On the other hand, reconstruction models are optimized to rebuild the observed segment. Examples include AutoEncoders (AE) [Sutskever et al., 2014, Malhotra et al., 2016], Variational AutoEncoders (VAE) [Park et al., 2018], Generative Adversarial Networks (GANs) [Goodfellow et al., 2014], and Transformers [Vaswani et al., 2017, Xu et al., 2022b, Tuli et al., 2022].

2.3.2 Possible ML Configurations

There are numerous possible mappings for ML models. Figure 2-4 illustrates possible configurations for a forecasting-based model. Reconstruction-based models are shown in Figure B-1

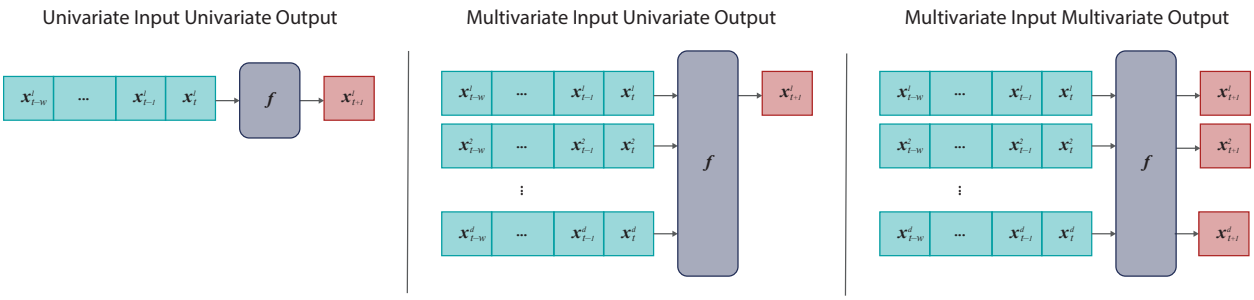


Figure 2-4: Model configurations for time series forecasting. (1) Univariate input and univariate output; (2) Multivariate input and univariate output; (3) Multivariate input and multivariate output.

in the Appendix. Possible configurations include:

1. **Univariate Input Univariate Output.** This mapping is the simplest setting of a model, where only one time series is learned.

2. **Multivariate Input Univariate Output.** Univariate models ignore the relationships between time series. In this configuration, the model learns these relationships to optimize the prediction of one time series.

3. **Multivariate Input Multivariate Output.** This configuration is the most complex. It typically cannot be scaled with high dimensional time series.

Focus of this thesis. This thesis focuses on the first two configurations. We use ML models to detect anomalies in a single time series.

We leave the extension of models to multivariate output through channel independence strategy, which is a recursive strategy that applies the model to each output independently [Han et al., 2023].

2.4 Preliminaries

Common pre-processing and post-processing operations that we will use throughout the thesis are introduced in this section.

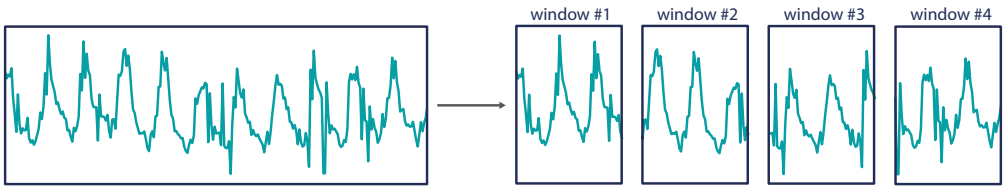


Figure 2-5: Creating rolling window sequences from a univariate time series.

2.4.1 Pre-processing

Time series pre-processing is an essential step of preparing data for any deep learning model. Let $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}$, denote a multivariate signal with d channels where $\boldsymbol{x}_i \in \mathbb{R}^d$. Below are some common operations we will apply before any time series modeling.

Equi-spaced Time Series. Given time series data, which are typically collected at an irregular frequency, we seek to resample the data such that each sample is equally spaced in time. Formally, we can take the average value at each interval:

$$\tilde{\boldsymbol{x}}_t = \frac{1}{\Delta t} \sum_{t=1}^{t+\Delta t} \boldsymbol{x}_t \quad \forall t \in \{1, \ldots, T\}$$

where Δt is the interval over which aggregation occurs, and this is done for all timestamps.

Rolling Windows. Given that time series are often too long to be absorbed by the model, we create rolling windows from the time series, as illustrated in Figure 2-5. Let w be the window size, and s the step size. Then we can create $N = \lceil (T - w)/s \rceil$ windows:

$$\mathcal{X}_{\text{windows}} = \{(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_{i+w})^i\}_{i=1}^N$$

To denote a specific window $\mathcal{X}^{(i)}_{\text{windows}}$ indexed at i, we sometimes refer to it as $\boldsymbol{x}_{i \ldots i+w}$, or $\boldsymbol{x}_{w_i}$ for simplicity. Note that the example illustrated in Figure 2-5 produces non-overlapping segments, since window size = step size. Segmenting the full time series into multiple small windows is a popular approach to creating the training dataset for deep learning models [Lütkepohl and Krätzig, 2004, Lv et al., 2014, Shi et al., 2015].

(a) point (b) area (c) dynamic time warping

Figure 2-6: High-level depiction of (a) point error (b) area error (c) dynamic time warping.

2.4.2 Post-processing

After training a machine learning model to learn the patterns of a given time series (as shown in Figure 2-3), we need to quantify the error between the model's expectation and the actual time series, and apply a threshold to the calculated error.

Error Functions. Our goal is to find the error vector $\boldsymbol{e} = \{e_1, e_2, \ldots, e_T\}$ that shows where the deviations are. To calculate the error between two given signals, we use a similarity function. The model will produce a one-dimensional vector (which we will refer to as $\hat{y} \in \mathbb{R}$) that aims to mimic the normal behavior of a channel in the time series $y \subseteq \mathcal{X}$. The objective is now to capture the discrepancies between y and $\hat{y}$ to help us locate anomalies. There are several approaches to this, including: point-wise difference, area difference, and dynamic time warping, which we highlight in Figure 2-6.

Point Error

This method applies a point-to-point comparison between the original and the generated signal. It is considered a sensitive approach that does not allow for many mistakes. For each step t, the prediction error is calculated:

$$f(y_t, \hat{y}_t) = |\, y_t - \hat{y}_t \,| \tag{2.1}$$

Area Error

This method captures the general area under the curve of both signals and then compares them. It is lenient, in the sense that the y and $\hat{y}$ do not necessarily need to have the same

shape in order to be similar:

$$f(y_t, \hat{y}_t) = \frac{1}{2l} \left| \int_{t-l}^{t+l} y_t - \hat{y}_t \, dy \right| \tag{2.2}$$

Dynamic Time Warping Error

A middle ground between the previous two methods is Dynamic Time Warping (DTW). DTW is a ubiquitous similarity measure between two temporal sequences that may vary (i.e. warp) in time [Berndt and Clifford, 1994]. It compares two signals using any pair-wise distance measure, but allows for one signal to lag behind another.

Given y and $\hat{y}$, we need to construct a warp path $W = w_1, w_2, \ldots, w_K$, where K is the length of the warp path and $w_k = (i, j)$ is the k^{th} element of that warp path mapping y_i to $\hat{y}_j$. The optimal path is given by

$$f(y, \hat{y}) = W^* = \min_W \left[\sqrt{\sum_{k=1}^{K} w_k} \right] \tag{2.3}$$

Given two time series sequences, we want to find an m-to-n mapping that illustrates which value in series y corresponds to which value in series $\hat{y}$ [Berndt and Clifford, 1994]. More formally, we need to find the optimal warping path W. This path can be found using a recurrence formulation of the cumulative cost γ. We define $\gamma(i, j)$ as the distance $d(i, j)$ of the current cell using any point-wise distance measure (e.g. Euclidean distance) and the minimum of the cumulative distances of the adjacent elements

$$\gamma(i, j) = d(y_i, \hat{y}_j) + min\{\gamma(i-1, j), \gamma(i, j-1), \gamma(i-1, j-1)\}$$

There are three main constraints imposed on DTW:

1. Boundary condition: The first and last index from y must be matched with the first and last index from $\hat{y}$ respectively. In other words, the warping path W needs to have $w_1 = (1, 1)$ and $w_K = (n, m)$. Since we assume $|y| = |\hat{y}|$ then $w_K = (n, n)$.

2. Continuity condition: Every index from y must be matched with one or more indices from $\hat{y}$, and vice versa. In other words, for a point (i, j) from the matrix, the previous point

must be $(i-1, j-1)$, $(i-1, j)$, or $(i, j-1)$.

3. Monotonic condition: The mapping between y and $\hat{y}$ must be monotonically increasing. Given $w_k = (a, b)$ then $w_{k-1} = (a', b')$ where $a - a' \geq 0$ and $b - b' \geq 0$.

Thresholding Once an error signal is obtained $\boldsymbol{e} = \{e_1, e_2, \ldots, e_T\}$, we can apply a threshold to find the anomalies. A common approach is to apply a *global threshold* to mark high error values as anomalous. However, such an approach misses local anomalies, which have high error scores compared to their neighbors. To capture both global and local anomalies, we apply a *sliding window* approach to thresholding:

$$a_t = \begin{cases} 1 & e_t > \delta_k \\ 0 & \text{otherwise} \end{cases} \qquad \text{where} \quad \delta_k = \mu_{w_t} + \kappa\, \sigma_{w_t}$$

where w is the window size, μ_{w_t} is the mean of the values captured in the window $\{e_t, e_{t+1}, \ldots, e_{t+w}\}$, and similarly σ_{w_t} is the standard deviation.

2.5 Conclusion

In this chapter, we outlined the types of time series data addressed in this thesis and proposed a taxonomy for categorizing anomalies of interest. We further emphasized that the presence of an anomaly does not inherently imply a problem, as certain deviations may be explainable under specific operational conditions. Finally, we introduced the role of machine learning in anomaly detection and illustrated key pre– and post-processing operations that will be used in subsequent chapters.

Chapter 3

Datasets & Related Work

Chapter Outline:

- Section 3.1 introduces some of the datasets used throughout this thesis.
- Section 3.2 describes related work about time series anomaly detection methods, systems, and benchmarks.

3.1 Datasets

Throughout this thesis, we use public time series datasets with known ground truths to test our models and systems. In this section, we introduce these datasets.

3.1.1 Dataset Details

We utilize 14 publicly accessible datasets from different sources. Table 3.1 illustrates some of their properties. Below, we provide a more detailed description for each dataset.

NASA. This is spacecraft telemetry data provided by NASA. It was originally released in 2018 as part of the `LSTM-DT` paper [Hundman et al., 2018], and can be accessed directly from their github. [1] It features two datasets: Mars Science Laboratory (MSL) and Soil Moisture Active Passive (SMAP).

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

	Dataset	# Signals	# Anomalies	Length=1	Avg. Signal	Real
NASA	MSL	27	36	0	4890.59	✓
	SMAP	53	67	0	10618.86	✓
NAB	Art	6	6	0	4032.00	✗
	AWS	17	30	0	3980.35	✓
	AdEx	5	11	0	1593.40	✓
	Traf	7	14	0	2237.71	✓
	Tweets	10	33	0	15863.1	✓
Yahoo S5	A1	67	178	68	1415.9	✓
	A2	100	200	33	1421.0	✗
	A3	100	939	935	1680.0	✗
	A4	100	835	833	1680.0	✗
Total		492	2349			

Table 3.1: Datasets Summary. There are 14 datasets with varying number of signals and anomalies. The table presents the average signal length and anomaly length for each dataset. All these datasets are publicly accessible.

- `MSL` contains 27 signals with 36 anomalies.
- `SMAP` contains 53 signals with 69 anomalies.

In total, the NASA datasets contains 80 signals with 105 anomalies. This dataset was pre-split into training and testing partitions.

NAB. The NAB dataset is part of the Numenta benchmark [Lavin and Ahmad, 2015]. [2] It includes multiple types of time series data from various applications and domains. In our benchmark, we selected five sub-datasets (name: # signals, # anomalies):

- artWithAnomaly (`Art`: 6, 6): this dataset was artificially generated;
- realAWSCloudwatch (`AWS`: 17, 20): this dataset contains AWS server metrics collected by Amazon Cloudwatch services, such as CPU Utilization;
- realAdExchange (`AdEx`: 5, 11), this dataset contains online advertisement clickrate metrics such as cost-per-click;

[2] https://github.com/numenta/NAB

- realTraffic (`Traf`: 7, 14): this dataset contains real-time traffic metrics from the Twin Cities Metro area in Minnesota, including vehicle occupancy, speed, etc.;
- realTweets (`Tweets`: 10, 33): this dataset contains metrics for a collection of Twitter mentions of companies (e.g. Google), such as number of mentions every 5 minutes.

Yahoo S5. This dataset contains four different sub-datasets. The A1 dataset is based on the real production traffic of Yahoo computing systems, with 67 signals and 179 anomalies. A2, A3 and A4 are all synthetic datasets, with 100 signals each and 200, 939, and 835 anomalies respectively. There are many anomalies in this dataset, with over 2,153 in 367 signals, averaging 5.8 anomalies in each signal. Most of the anomalies in A3 and A4 are short and last for only a few points in time. This data can be requested from Yahoo's website. [3]

UCR. This dataset was released in a SIGKDD competition in 2021. [4] It contains 250 signals, with only one anomaly in each signal. The anomalies themselves were artificially introduced into the signal. More specifically, many are synthetic anomalies, made by flipping, smoothing, interpolating, reversing, or prolonging normal segments. The dataset was created to provide access to more challenging anomalies.

3.2 Related Work

The scope of this thesis lies at the intersection of four domains: time series anomaly detection algorithms, time series anomaly detection systems, anomaly detection benchmark systems, and foundation models for time series.

3.2.1 Time Series Anomaly Detection Algorithms

Many algorithms have been proposed to address time series anomaly detection [Chandola et al., 2009, Goldstein and Uchida, 2016]. The most basic approaches simply flag regions where values exceed a certain threshold [Martínez-Heras and Donati, 2014, DeCoste, 1997].

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

[4] https://www.cs.ucr.edu/~eamonn/time_series_data_2018/UCR_TimeSeriesAnomalyDatasets2021.zip

While these methods are intuitive, they struggle to detect contextual anomalies – values that are anomalous within local contexts. More advanced methods are based on statistical hypothesis testing [Zheng et al., 2016], clustering [Iverson, 2004, 2008], and/or machine learning [Yairi et al., 2006]. We categorize anomaly detection methods as *proximity-*, *prediction-*, and *reconstruction-based* methods.

Proximity Methods. Proximity-based methods use a distance measure to quantify similarities between objects. Objects that are isolated and distant from others are considered anomalies. Proximity-based methods can be further divided into distance-based methods, such as K-Nearest Neighbor (KNN) [Angiulli and Pizzuti, 2002] – which use a given radius to define neighbors of an object and use the number of neighbors to determine an anomaly score – and density-based methods, such as Local Outlier Factor (LOF) [Breunig et al., 2000] and Clustering-Based Local Outlier Factor [He et al., 2003], which base measures of similarity on the density of objects and their neighbors. One drawback of applying proximity-based methods to time series data is that it requires prior knowledge of the expected number and duration of anomalies. Moreover, most of these methods do not capture temporal correlations.

Prediction Methods. Prediction-based methods involve a predictive model learning the patterns of the given time series data, and then predicting future values. This approach is similar to forecasting future values and then using the forecasting signal as a representation of what the original time series should look like. We then find the discrepancies between the predicted signal and the original signal, which allows us to pinpoint anomalous regions. Some of the most classic time series anomaly detection techniques are prediction-based, including ARIMA [Box and Pierce, 1970, Pena et al., 2013], Hidden Markov Models (HMM) [Zucchini and MacDonald, 2009], and Functional Data Analysis (FDA) [Torres et al., 2011], which all work by forecasting a signal and comparing it to the original. However, these methods are sensitive to parameter selection, and often require strong assumptions and extensive domain knowledge. Recent advancements in deep neural networks have led to the emergence of deep learning-based anomaly detection approaches to overcome these limitations. Hundman et al. [2018] propose a forecasting model assembled from Long Short-Term Memory networks to predict future values. In addition, they complement their model with non-parametric

dynamic thresholds, which aim to prune detected anomalies that are very close in error scores to normal intervals. A similar strategy has been proposed with Temporal Convolution Networks (TCN) [He and Zhao, 2019].

Reconstruction Methods. Reconstruction-based methods learn a model in order to capture the latent structure (low-dimensional representations) of the given time series data and then create a reconstruction of the signal. Reconstruction-based methods assume that anomalous information is lost when they are mapped to a lower dimensional space and thereby cannot be effectively reconstructed. Similar to prediction-based methods, we calculate the error as the deviation between the reconstructed signal and the original signal. We use this error signal to find anomalous regions. Principal Component Analysis (PCA) [Ringberg et al., 2007], a dimensionality reduction technique, can be used to reconstruct data, but is limited to linear reconstruction and requires the data to be highly correlated and to follow a Gaussian distribution [Dai and Gao, 2013]. With respect to deep learning, further reconstruction-based techniques have been investigated. These include the use of Auto-Encoder (AE) [Sutskever et al., 2014, Malhotra et al., 2016], Variational Auto-Encoder (VAE) [An and Cho, 2015], Generative Adversarial Networks (GAN) [Goodfellow et al., 2014, Geiger et al., 2020], and Transformer models [Vaswani et al., 2017, Tuli et al., 2022, Xu et al., 2022b].

While methods and algorithms provide innovative approaches for detecting anomalies, they alone do not support the necessary end-to-end workflow — from the input signal processing, model training, post-processing and evaluation, to the output signal and anomaly visualization and annotation — that would adequately assist users in making decisions.

3.2.2 Time Series Anomaly Detection Systems

With the increasing prevalence of time series data, a wide range of systems made specifically for time series have emerged. These address a variety of tasks, such as classification [Cao et al., 2019], feature extraction [Christ et al., 2018], and anomaly detection [Veeramachaneni et al., 2016, Laptev et al., 2015, Ren et al., 2019, Gao et al., 2020]. Table 3.2 summarizes the features present in some existing open source frameworks for anomaly detection. While these systems handle time series data, most of them only support a single anomaly detection

algorithm. Moreover, they fail to support APIs for usability. In contrast, Orion aims to provide an end-to-end development workflow to aid all types of users. New primitives and pipelines can be constructed and integrated using a `fit-predict` interface with minimal overhead [Smith et al., 2020]. Python and RESTful APIs make the system usable.

3.2.3 Anomaly Detection Benchmarks

Training and optimizing deep learning models can be computationally expensive, making selecting and comparing pipelines difficult. Benchmarking frameworks has become necessary for evaluating and comparing model performance in a standardized end-to-end fashion [Coleman et al., 2017]. With respect to anomaly detection, Lavin and Ahmad [2015] introduced one of the first open-source benchmark repositories for anomaly detection. They provide a collection of datasets (58 signals) from real-word and artificial sources. Recently Jacob et al. [2021] introduced Exathlon – a benchmark framework for anomaly detection and explanation discovery. The framework features a dataset generated systematically from real-world data traces. In addition, it elicits some of the intricacies involving time series benchmarks, including evaluation metrics and performance monitoring.

Table 3.3 compares multiple benchmarking systems with respect to model and data availability, as well as system features such as extensibility. Currently, benchmarking frameworks have limited pipelines, and are not easily extendable.

3.2.4 Foundation Models for Time Series

Large Language Models for Time Series. The past several months have seen considerable efforts toward Large Language Model (LLM) utilization for time series data. Given the parallels between predicting the next word in a sentence and predicting the next value in a time series, most of these efforts have focused on time series forecasting. One notable effort is LLMTime, where Gruver et al. [2023] employ GPT [Brown et al., 2020], and Llama [Touvron et al., 2023] models to forecast time series data. PromptCast [Xue and Salim, 2023] is a related work that translates a forecasting problem into a prompt, transforming forecasting into a question-answering task.

Other frameworks rely on fine-tuning to adapt LLMs for time series tasks. LLM4TS [Chang et al., 2023a] fine-tunes a GPT-2 [Radford et al., 2019] model to align the model to time series data and improve the forecasting task on public datasets. Similarly, TEMPO [Cao et al., 2023] utilizes GPT-2 with STL decomposition [Cleveland et al., 1990] to extract valuable embeddings for next-value prediction.

Pretrained Transformers on Time Series Data. Recently, more work has emerged adopting transformer-based models for forecasting purposes by pretraining transformer models with many parameters on a large corpus of time series data. TimeGPT is a private transformer model with an encoder-decoder structure that is pretrained on a large collection of publicly available time series datasets. ForecastPFN [Dooley et al., 2023] pre-trains a basic encoder-only transformer on a synthetically generated time series dataset. While GTT is another encoder-only architecture that trains on 200M time series samples from various domains [Feng et al., 2024]. Lag-Llama is a decoder-only Llama model pretrained on a large corpus of real time series data from diverse domains [Rasul et al., 2023]. Similarly, TimesFM is decoder-only architecture trained on real and synthetic datasets [Das et al., 2024]. The aforementioned models are trained to predict the next value or segment in a time series dataset, i.e. to forecast. On the other hand, UniTS [Gao et al., 2024] is a foundation model trained on public datasets and is not limited to forecasting. Their unified framework allows the model to be used for forecasting, imputation, anomaly detection, and classification. Moreover, Chronos [Ansari et al., 2024] converts transformers built for language into time series models. Specifically, they adopt a T5 architecture and parse 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.

Overall, there has been a tremendous shift towards building time series foundation models, mostly for forecasting.

		MS Azure 2019	ADTK 2020	Luminaire 2020	TODS 2020	Telemanom 2018	NAB 2017	EGADS 2015	Stumpy 2019	GluonTS 2020	Orion
Persona	End User	✓	✓	✓	✗	✗	✗	✗	✓	✗	✓
	System Builder	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
	ML Researcher	✗	✗	✗	✓	✓	✓	✓	✗	✓	✓
Engine	Preprocessing	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓
	Modeling	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
	Postprocessing	✗	✓	✓	✓	✗	✗	✗	✓	✗	✓
Modular		✗	✓	✓	✓	✗	✗	✗	✓	✓	✓
Comp.	Evaluation	✗	✓	✗	✗	✓	✗	✗	✗	✗	✓
	Benchmark	✗	✗	✗	✓	✗	✓	✗	✗	✓	✓
	Database	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
API	`lang. specific`	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓
	`RESTful`	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
HIL		✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Table 3.2: Comparison of anomaly detection softwares. A (✓) indicates the package includes an attribute, while an (✗) indicates the attribute is absent. Attribute categories from top to bottom: *Users* shows which user types can benefit from each software: *end users* are interested in detecting anomalies, *system builders* are interested in adding their own workflows, and *ML researchers* are interested in creating new pipelines that outperform existing methods; *Engine* denotes the operations handled by each software; *Modular* indicates whether or not pipelines can reuse primitives; *Comp.* shows whether systems include certain components including custom *evaluation* mechanisms, *benchmarking* frameworks, and an integrated *database* of results; *API* refers to the inclusion of APIs for user interaction, whether language-specific or `REST`; and *HIL* denotes the presence or absence of a human-in-the-loop component that can integrate experts' knowledge back into the system.

	Available		Pipeline Type			Properties			Published Leaderboard	
Framework	# Datasets	# Pipelines	Classic	DL	BBox	Evaluation	Extensible	Periodic	Last Update	Source
`Numenta` 2015	7	4	✓	✗	✗	✓	✓	✗	Jun 2018	Github
`TSB-UAD` 2022	18[1]	12	✓	✓	✗	✓	✗	✗	Nov 2022	Github
`TODS` 2021	4[1]	9	✓	✓	✗	✗	✗	✗	Dec 2021	Paper
`TimeEval` 2022	23	71	✓	✓	✗	✗	✓	✗	Aug 2022	Paper
`Exathlon` 2021	10[2]	3	✗	✓	✗	✓	✗	✗	Sep 2021	Paper
`Merlion` 2021	12	12	✓	✓	✗	✓	✗	✗	Sep 2021	Paper
OrionBench	14	12	✓	✓	✓	✓	✓	✓	Mar 2025	Github

Table 3.3: Comparison of anomaly detection benchmarks. A (✓) indicates the framework includes an attribute, while an (✗) indicates the attribute is absent. #Datasets and #Pipelines columns represent the number of currently available datasets and pipelines respectively. Columns under "Pipeline Type" represent whether certain pipeline types are supported, including classic pipelines such as ARIMA, Deep Learning (DL) pipelines such as LSTM, and BlackBox (BBox) pipelines that are called externally through an API such as Azure's AD service. Columns under "Properties" represent whether a benchmark has certain properties, including custom evaluation methods for time series anomaly detection; whether the benchmark is extensible and can integrate new datasets and pipelines, and of the benchmark is being released in periodic fashion with an updated leaderboard. The last two columns illustrate the last time a leaderboard was published and whether it has been shared in a paper or directly on GitHub.

[1]Additional synthetic and artificial data is generated in the benchmark.
[2]Data are traces from 10 distributed streaming jobs on a Spark cluster.

Chapter 4

Models for Unsupervised Time Series Anomaly Detection

This chapter looks closely at the task of anomaly detection. We formalize the task and its objectives, propose our approaches to using deep learning for anomaly detection, and introduce evaluation mechanisms specific to this task.

Chapter outline:

- Section 4.1 formally defines the unsupervised time series anomaly detection task and its objective.

- We propose our modeling methods in Section 4.2, where we build a generative adversarial network, an autoencoder with regression, and another autoencoder with explicit modeling of control and status signals.

- Section 4.3 illustrates two time series-based evaluation strategies for downstream metric calculations, including precision, recall, and F_1 score.

- Lastly, Section 4.4 showcases the preliminary performance of our proposed models under the proposed evaluation metrics.

4.1 Anomaly Detection Task Definition

In an anomaly detection task, we aim to detect whether any intervals in the time series data are anomalous; i.e. deviate from "normal." Given $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}$, a multivariate signal of length T with d channels where $\boldsymbol{x}_i \in \mathbb{R}^d$, and assuming there exists a set of *variable-length* anomalies $\mathcal{A} = \{(t_s, t_e)^k \mid 1 \leq t_s < t_e \leq T,\ k \geq 0\}$, we aim to train a model M that detects $\mathcal{A}$; $M(\mathcal{X}) \rightarrow \mathcal{A}$ where:

- $\mathcal{A}$ is unknown a priori.
- Because anomalies are scarce, often times $k = 0$.

Note that we address *variable-length* anomalies. For two anomalous intervals $a_1 = (t_s^1, t_e^1)$ and $a_2 = (t_s^2, t_e^2)$, the length of a_1 does not necessarily equal the length of a_2; for example, $(t_e^1 - t_s^1) \neq (t_e^2 - t_s^2)$ is typical for this problem.

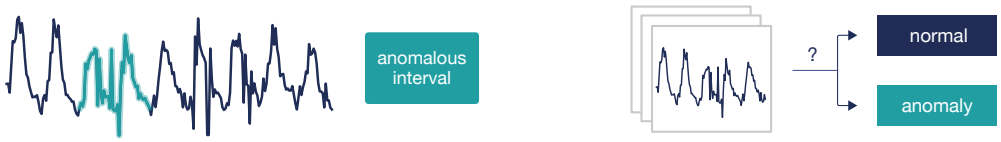
4.1.1 Common Confusion with Supervised Anomaly Detection

Several studies conflate this method with supervised anomaly detection. In supervised anomaly detection:

- The set of $\mathcal{A}$ is known a priori.
- The goal of $M(\mathcal{X})$ is to detect $\mathcal{A}$.

How is this different from classification? A time series classification task takes as input $\mathcal{X}$, and maps it to a probability distribution over the possible class variable values (*labels*) [Ismail Fawaz et al., 2019]. In an anomaly detection task, these class labels pertain to whether something is an anomaly or not. For example, Cao et al. [2019] use EEG data as input, divides the raw EEG data into segments of fixed duration, and then produces a single output that assigns a label to that particular segment. Hu et al. [2013] use time series classification to label ECG data with associated types of physical activity: normal-walking, walking-very-slow, descending-stairs, cycling, inactivity, etc.

We distinguish this classification problem from our anomaly detection task (Figure 4-1). In a detection task, the model works with the entire time series in its raw format and must



(a) Detection task (b) Classification task

Figure 4-1: Illustration of (a) a detection task, where the objective is to find anomalies in time series data, and (b) a classification task, where the objective is to assign a label to a time series or segment.

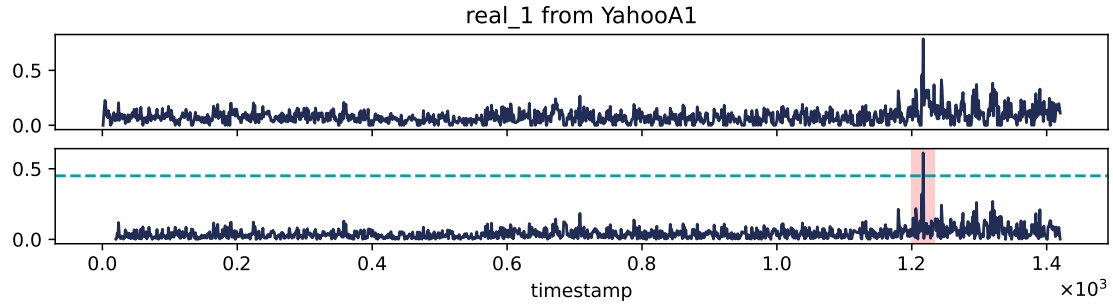


Figure 4-2: Detecting anomalies in `real_1` signal from YahooA1 using a moving average and a threshold. The top plot shows the raw signals, while the bottom plot shows the residuals from the moving average.

predict whether the signal contains anomalies. It also must *localize* them by producing start and end time stamp intervals.

4.1.2 Implications of Supervised versus Unsupervised

In a supervised setting, the set of anomalies $\mathcal{A}$ is known and available. Let θ denote the parameters of model M, meaning that θ can be tuned to maximize the detector accuracy based on $\mathcal{A}$. Let ϕ be an accuracy function, such as finding the overlap between two sets of intervals. We can then optimize our model to increase the overlap:

$$\text{argmax}_{\theta} \left[\phi(\mathcal{A}, M(\mathcal{X}; \theta))\right] \tag{4.1}$$

Misconception Example. Let's consider the simple example of a signal shown in Figure 4-2, specifically `real_1` from the YahooA1 dataset. It's possible to apply a threshold on the residuals between the moving average function and the raw signal. This is conveyed simply

as a one-liner $\text{MAvg}(\mathcal{X}) > 0.45$, which allows us to find $\mathcal{A}$, shown in the plot as the red region. However, how did the user come up with:

- MAvg as a function, and
- 0.45 as a threshold?

Both either assume a priori knowledge of $\mathcal{A}$ in order to calibrate the model correctly, or must be done via visual inspection (not a scalable strategy).

Argument for Simpler Models. Some anomalies, particularly out-of-range anomalies, are simpler to find and detect. When this is the case, simple models should be used, given their interpretability and effectiveness. However, these simple models are not able to locate more difficult-to-find anomalies, such as contextual anomalies that are buried in the signal. We justify the use of complex deep learning models because they can detect all anomalies, regardless of their complexity or type.

4.2 Our Methods for Unsupervised Anomaly Detection

In this section, we go through the technical details of two deep learning models: Generative Adversarial Networks and Auto-Encoder with Regression. The objective of these models is to generate an "expected" signal, as illustrated in Section 2.3, and compare it to the real signal in order to detect anomalies. We continue by describing our approach for modifying an LSTM Auto-Encoder to incorporate control and status signals.

4.2.1 TadGAN: Time Series Anomaly Detection using Generative Adversarial Networks[1]

One of the foundational challenges of deep learning-based approaches is that their remarkable ability to fit data carries the risk that they could fit anomalous data as well. For example, autoencoders, using an $L2$ objective function, can fit and reconstruct data extremely accurately – thus sometimes fitting anomalies as well. On the other hand, generative adversarial

[1]First version of this work was led by Alexander Geiger.

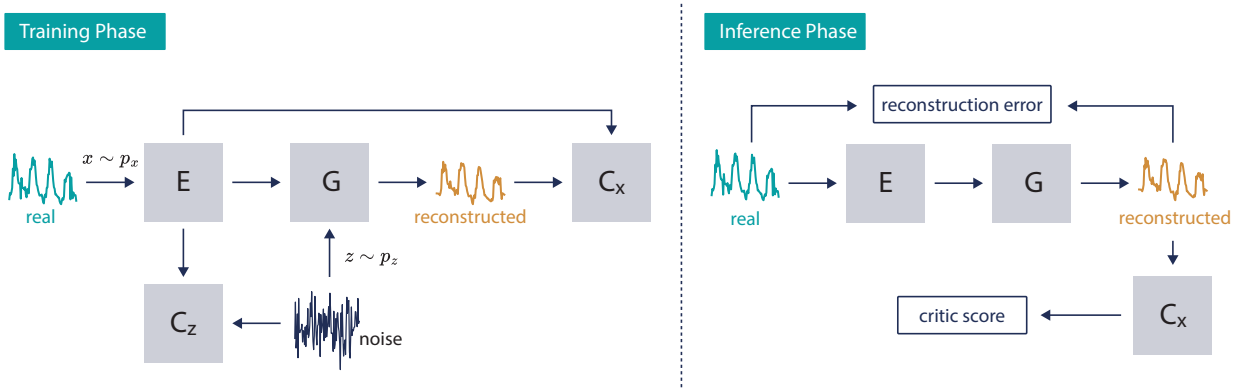


Figure 4-3: TadGAN's proposed workflow, illustrating the data flow at the training (left) and detection stages (right). During training, we learn two mapping functions: an encoder ($\mathcal{E}$) that maps the signal to the latent representation, "z", and a generator ($\mathcal{G}$) that recovers the signal from the latent variable. Both networks are adversarially trained. In detection, we use the trained encoder ($\mathcal{E}$) and generator ($\mathcal{G}$) to reconstruct the signal.

networks [Goodfellow et al., 2014] may be ineffective at learning the generator to fully capture the data's hidden distributions, thus causing false alarms. Here, we mix the two methods, creating a more nuanced approach.

Overview. TadGAN is a reconstruction-based method that uses GANS to learn the expected behavior of the signal [Geiger et al., 2020]. The generator $\mathcal{G} : \mathcal{Z} \rightarrow \mathcal{X}$ creates a time series based on some random latent variable in order to fool the critic $\mathcal{C}_x$, which tries to differentiate between "generated" examples and "real" ones. The generator and critic are then trained via min-max game:

$$\min_{\mathcal{G}} \max_{\mathcal{C}_x} V(\mathcal{G}, \mathcal{C}_x) = \mathbb{E}_{x \sim p_x}[\log \mathcal{C}_x(\boldsymbol{x})] + \mathbb{E}_{z \sim p_z}[\log (1 - \mathcal{C}_x(\mathcal{G}(z)))] \tag{4.2}$$

However, given the temporal complexity of time series data, randomly sampling from the latent space can lead to drastically inconsistent samples. To account for this variance, we introduce an encoder $\mathcal{E} : \mathcal{X} \rightarrow \mathcal{Z}$ to map time series to the latent space. We train the encoder adversarially through another critic $\mathcal{C}_z$ to distinguish between random latent samples and encoded samples.

$$\min_{\mathcal{E}} \max_{\mathcal{C}_z} V(\mathcal{E}, \mathcal{C}_z) = \mathbb{E}_{z \sim p_z}[\log \mathcal{C}_z(z)] + \mathbb{E}_{x \sim p_x}[\log (1 - \mathcal{C}_z(\mathcal{E}(\boldsymbol{x})))] \tag{4.3}$$

To reconstruct a time series $\boldsymbol{x} \in \mathcal{X}$, we use the trained encoder and generator: $\boldsymbol{x} \rightarrow \mathcal{E}(\boldsymbol{x}) \rightarrow \mathcal{G}(\mathcal{E}(\boldsymbol{x})) \rightarrow \hat{\boldsymbol{x}}$. To encourage mapping a sample from $\boldsymbol{x}$ to $\hat{\boldsymbol{x}}$, we add a cycle consistency loss by minimizing the $L2$ norm of the difference between the original and the reconstructed samples.

$$V_{L2}(\mathcal{E}, \mathcal{G}) = \mathbb{E}_{x \sim p_x}[\|\boldsymbol{x} - \mathcal{G}(\mathcal{E}(\boldsymbol{x}))\|_2] \tag{4.4}$$

The global optimization objective becomes a combination of all the aforementioned objectives:

$$\min_{\mathcal{G},\mathcal{E}} \max_{\mathcal{C}_x,\mathcal{C}_z} V(\mathcal{G}, \mathcal{C}_x) + V(\mathcal{E}, \mathcal{C}_z) + V_{L2}(\mathcal{E}, \mathcal{G}) \tag{4.5}$$

This process is shown in Figure 4-3.

Anomaly Scores. After reconstructing the signal, we calculate the discrepancies between the original and reconstructed time series to find anomalies, as illustrated in Section 2.3. Since the critic $\mathcal{C}_x$ is trained to distinguish original samples from generated ones, we utilize the critic scores as part of the reconstruction error. This can be done as a weighted average between the critic score and any other distance-based error f described in Section 2.4, such as point-wise error. We can formulate the general notation of an anomaly for some weight $\lambda \in [0, 1]$ as:

$$\boldsymbol{e} = \lambda\, \mathcal{C}_x(\boldsymbol{x}) + (1 - \lambda)\, f(\boldsymbol{x}, \hat{\boldsymbol{x}}) \tag{4.6}$$

Summary. TadGAN is a reconstruction-based model that leverages generative adversarial networks for anomaly detection in time series. The anomaly scores are composed of a weighted combination of the reconstruction error as well as the critic scores. We assume that the training data is free from anomalies during training, ensuring that the reconstruction error is indicative of where the anomalies are and that the critic model learns only the normal patterns of the signal. However, in practical settings, we cannot guarantee this assumption. In our evaluation, we show how TadGAN is performative in identifying contextual anomalies; however, it fails to perform as well at detecting out-of-range anomalies.

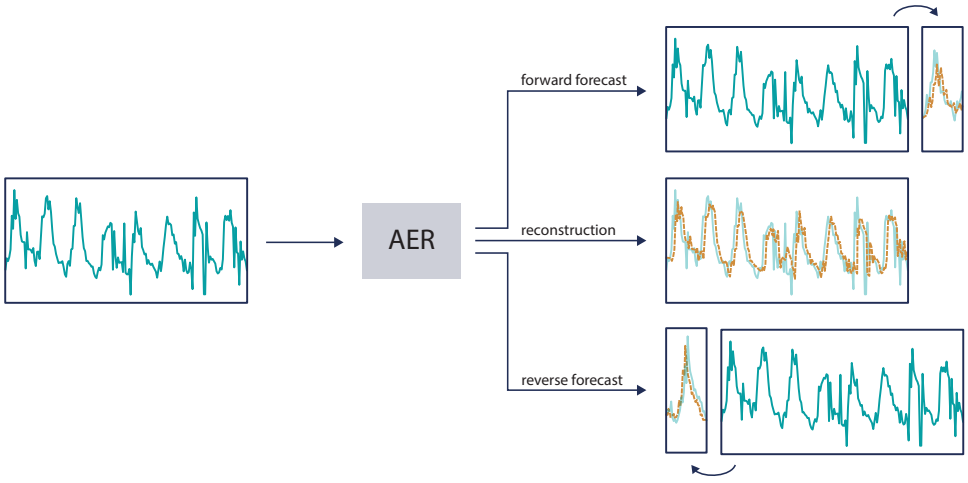


Figure 4-4: AER is a joint model consisting of an LSTM auto-encoder and regressor capable of (1) producing forward forecast (2) producing reverse forecast and (3) reconstructing the input sequence.

4.2.2 AER: Auto-Encoder with Regressor for Time Series Anomaly Detection[2]

Prediction-based and reconstruction-based anomaly scores have advantages and limitations that complement one another. For example, we observe from our experiments that prediction-based anomaly scores are better at identifying point anomalies, but produce relatively more false positives. On the other hand, reconstruction-based anomaly scores are better at identifying contextual anomalies, but produce relatively more false negatives. Therefore, our method strives to address these limitations and to leverage strengths from both types of models as an alternative solution for anomaly detection in time series.

Overview. AER is a hybrid model with an internal autoencoder that reconstructs the signal and also predicts the next and previous value of the input sequence, as illustrated in Figure 4-4. The idea is to leverage the advantages of both reconstruction– and prediction-based methods combined. Concretely, the model learns three representations for $\boldsymbol{x} \in \mathcal{X}$. Let w be the window size in which we segment our time series; then for each sample $\boldsymbol{x}_{t:t+w-1}$ starting at index t, we find:

- a reconstructed segment of the same sample $\hat{\boldsymbol{x}}_{t:t+w-1}$.

[2]Led by Lawrence Wong.

- a forward prediction of the next value in the segment $\boldsymbol{y}_{t+w}$.

- a reverse prediction of the previous value in the segment $\boldsymbol{r}_{t-1}$.

The objective function is a weighted combination of the reconstruction and prediction losses given $\gamma \in [0, 1]$:

$$\frac{\gamma}{2} V_{pred}\left(\boldsymbol{x}_{t+w}, \boldsymbol{y}_{t+w}\right) + \frac{\gamma}{2} V_{pred}\left(\boldsymbol{x}_{t-1}, \boldsymbol{r}_{t-1}\right) + (1-\gamma) V_{rec}\left(\boldsymbol{x}_{t:t+w-1}, \hat{\boldsymbol{x}}_{t:t+w-1}\right) \tag{4.7}$$

where V_{pred}, and V_{rec} are prediction and reconstruction loss functions, respectively. In practice, mean squared error is used for both V_{pred} and V_{rec}. After obtaining the three distinct variations of the time series, we first align the bi-directional scores from the forward and reverse predictions:

$$f_{bi-directional}(\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{y}) = \begin{cases} f_{pred}(x_t, r_t) & 1 \leq t \leq w \\ \frac{1}{2} f_{pred}(x_t, r_t) + \frac{1}{2} f_{pred}(x_t, y_t) & w < t < T - w \\ f_{pred}(x_t, y_t) & T - w \leq t \leq T \end{cases} \tag{4.8}$$

where f_{pred} is a prediction error function, such as point-wise error.

Anomaly Scores. The final anomaly score is the weighted average $\lambda \in [0, 1]$ between bi-directional and reconstruction errors:

$$\boldsymbol{e} = \lambda f_{bi-directional}(\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{y}) + (1-\lambda) f_{rec}(\boldsymbol{x}, \hat{\boldsymbol{x}}) \tag{4.9}$$

Summary. AER is a model that combines that successes of both reconstruction– and prediction-based approaches, where the anomaly score is calculated as a weighted combination of each approach's error measure. There are several ways to improve the AER. For example, the model architecture could be improved, since our study uses a vanilla auto-encoder architecture with one bidirectional LSTM layer for both the encoder and decoder. Our framework is designed to easily extend to any reconstruction- based method with minimum changes to the objective function.

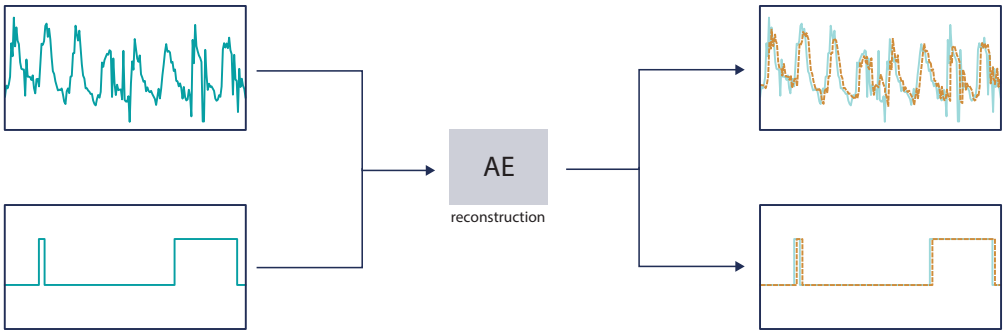


Figure 4-5: MixedLSTM is an LSTM auto-encoder at its core that incorporates the modeling of interdependencies in multivariate data as an auxiliary term.

4.2.3 MixedLSTM: Mixed Auto-Encoder for Multivariate Time Series Anomaly Detection with Context[3]

With automated monitoring, various control settings are recorded alongside the observation series to describe the operation conditions of the asset. In TadGAN and AER, $\mathcal{X}$ is treated as a pure observation set. However, in many multivariate cases, the set is composed of observation signals $\mathcal{X}_{obs}$ and context $\mathcal{X}_{ctxt}$ made up of control and status signals that exhibit the interdependencies between different channels in the data. We propose an auto-encoder of LSTM layers that factors in interdependencies as part of the modeling component. More specifically, we introduce a new term to the objective function that learns the behavior of contextual signals. Other models can also extend based on this work.

Overview. We focus mainly on incorporating the control/status signals into the loss objective during training to force the model to learn the different conditions of the signal. Let $\mathcal{X} = \mathcal{X}_{obs} \cup \mathcal{X}_{ctxt}$ be a multivariate time series dataset with d dimensions, where $\mathcal{X}_{obs} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}$ are observation signals consisting of continuous values of dimension d_{obs}, and $\mathcal{X}_{ctxt} = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_T\}$ denote a set of contextual signals of d_{ctxt} dimensions where each element c is binary; $c \in \{0, 1\}$. To properly model $\mathcal{X}$, we set an appropriate loss

[3]This work was done collaboratively with Grace Song.

function for each subset:

$$\mathcal{L}_{mse} = \frac{1}{b}\sum_{i=1}^{b}(\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i)^2, \tag{4.10}$$

$$\mathcal{L}_{ce} = -\frac{1}{b}\sum_{i=1}^{b}(\boldsymbol{c}_i \cdot \log \hat{\boldsymbol{c}}_i + (1 - \boldsymbol{c}_i) \cdot \log (1 - \hat{\boldsymbol{c}}_i)), \tag{4.11}$$

$$\mathcal{L}_{joint} = \lambda \cdot \mathcal{L}_{mse} + (1 - \lambda) \cdot \mathcal{L}_{ce} \tag{4.12}$$

where b is the batch size, and $\lambda \in [0, 1]$ is the weight for observation vs. context trade-off. More explicitly, we employ a mean squared error loss for observation signals and binary cross entropy for control/status signals. However, these signals are sparse by design, where "1" only appears to indicate a change in behavior. Given the imbalanced problem, we use focal loss (FL) to alleviate extreme imbalanced learning [Lin et al., 2017]. Let c be the ground truth of the control setting, and $p \in [0, 1]$ the model's estimated probability of $c = 1$. Then FL is defined as:

$$FL(p_c) = -\alpha\,(1 - p_c)^{\gamma} \log(p_c), \qquad p_c = \begin{cases} p & c = 1 \\ (1 - p) & \text{otherwise} \end{cases} \tag{4.13}$$

where $\alpha \in [0, 1]$ is the weighing factor to balance classes, and $\gamma \geq 0$ is a focusing parameter to learn harder examples. More specifically, when $\gamma > 1$, the loss downweights the contribution of "easy" examples (ones with high probabilities p). Putting these together, we replace the classic binary cross-entropy with focal loss:

$$\mathcal{L}_{fl} = -\frac{1}{b}\sum_{i=1}^{b}\alpha\,(1 - p_{c_i})^{\gamma} \log(p_{c_i}) \tag{4.14}$$

$$\mathcal{L}_{joint} = \lambda \cdot \mathcal{L}_{mse} + (1 - \lambda) \cdot \mathcal{L}_{fl} \tag{4.15}$$

With explicit modeling of the control signals by the model, we aim to train a better model that learns the underlying patterns of the signal.

Anomaly Scores. The modeling stage outputs a reconstruction of each window across all channels in the data. There are two main strategies for identifying anomalies:

- *Observation Focused*: Control and status signals generally represent deliberate states (e.g., input commands) and typically do not exhibit anomalies. In contrast, anomalies are more likely to occur in continuous observation signals. Therefore, the objective should be to reconstruct the control and status signals as accurately as possible in order to understand how they influence the observation signals. However, anomaly detection should ultimately be applied only to the observation columns, as they are the primary source of abnormal behavior.

- *Column-wise Detection*: Apply anomaly detection independently to all columns. Then, filter and aggregate the detected anomalies by identifying time intervals that are frequently flagged the most channels and/or associated with the highest anomaly scores.

Summary. MixedLSTM highlights the potential of leveraging contextual signals for anomaly detection in multivariate, heterogeneous time series. It builds on a standard autoencoder architecture, augmented with an additional term to incorporate contextual information from control and status signals.

4.3 Evaluation Metrics

Metrics are essential for understanding the difference in performance between anomaly detection models. When evaluating the efficacy of a model, we rely on signals for which we have annotations — *known anomalies* — and treat them as ground truth anomalies. In classification, the most widely-used sampled-based metrics include *precision*, *recall* and F_1 scores. However, as noted by Tatbul et al. [2018], these scores are not useful in the context of time series, where data is not regularly sampled. For a given set of ground truth anomalies $\mathcal{A}_{truth}$ and detected anomalies $\mathcal{A}_{detect}$, each is a set consisting of time intervals (t_s, t_e) where t_s and t_e represent the start and end timestamps of an anomaly respectively. Here we define specific methods to enable the fair computation of metrics without restrictions on the data: weighted segment and overlapping segment.

Input: ground truth anomalies $\mathcal{A}_{truth}$, detected anomalies $\mathcal{A}_{detect}$
Output: confusion matrix $\mathbf{M}$
begin
  $E \leftarrow \mathcal{A}_{truth} \cup \mathcal{A}_{detect}$ // all t_s and t_e timestamps
  $T \leftarrow \varnothing, D \leftarrow \varnothing, W \leftarrow \varnothing$
  $E \leftarrow \text{sort}(E)$ // sort timestamps from small to large
  $start \leftarrow \text{pop}(E)$ // the first timestamp
  while $E \neq \varnothing$ **do**
    $end \leftarrow \text{pop}(E)$
    $a \leftarrow (start, end)$ // create a time interval (t_s, t_e)
    $W \leftarrow W \cup \{t_e - t_s\}$
    $T \leftarrow T \cup \{\text{overlap}(a, \mathcal{A}_{truth})\}$ // check if a in ground truth
    $D \leftarrow D \cup \{\text{overlap}(a, \mathcal{A}_{detect})\}$ // check if a in detected
    $start \leftarrow end$
  end
  $\mathbf{M} \leftarrow$ `confusion_matrix` (T, D, W)
return M

Algorithm 1: Weighted Segment Evaluation. We create sequences partitioned based on the ground truth and predicted anomalies. For each sequence, we obtain a time range and whether it is part of the ground truth or predicted set. We compute the confusion matrix weighted by its respective duration.

4.3.1 Weighted Segment (WS)

Weighted segment-based evaluation is a strict approach that weights each segment according to its actual time duration. As illustrated in Algorithm 1, the time series is segmented into multiple sequences by the edges of the anomalous intervals. For each edge, we record whether it was observed in the $\mathcal{A}_{truth}$ set or $\mathcal{A}_{detect}$ set and record it in T and D, respectively. We then compute the confusion matrix, which makes a segment-to-segment comparison and records true positive, false positive, false negative, and true negative accordingly. We then weight each segment by its time range. This approach is valuable when precise detection is needed. Note that in cases where the time series is inherently regularly sampled, this approach is equivalent to sample-based evaluation.

4.3.2 Overlapping Segment (OS)

Overlapping segment is a more lenient evaluation approach. It is inspired by the evaluation method of Hundman et al. [2018], which rewards the model if it alerts the user to even

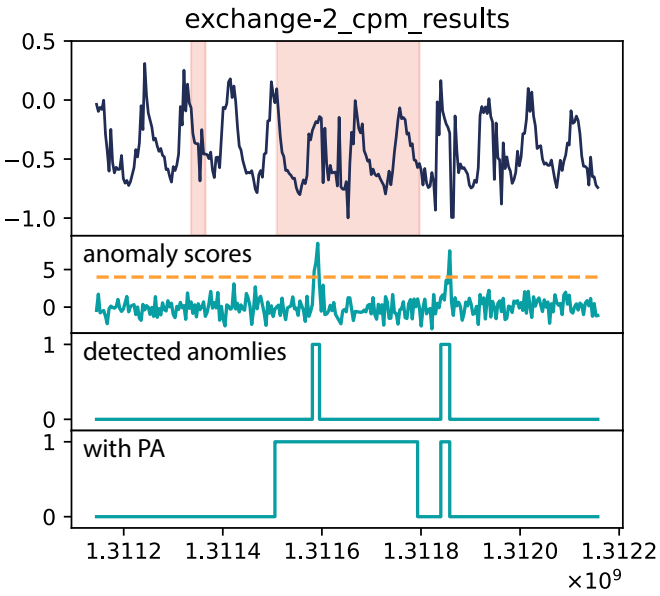
Input: ground truth anomalies $\mathcal{A}_{truth}$, detected anomalies $\mathcal{A}_{detect}$
Output: confusion matrix $\mathbf{M} = \langle \text{tp}, \text{fp}, \text{fn} \rangle$
begin
  $U \leftarrow \varnothing$ // bookkeeping unmatched events
  $\text{tp} \leftarrow 0$
  while $\mathcal{A}_{truth} \neq \varnothing$ **do**
    $gt \leftarrow \text{pop}(\mathcal{A}_{truth})$
    for $d \in \mathcal{A}_{detect}$ **do**
      if $\text{overlap}(gt, d)$ **then**
        $\text{tp} \leftarrow \text{tp} + 1$ // matched
      end
    end
    if $\text{unmatched}(gt)$ **then**
      $U \leftarrow U \cup \{gt\}$ // add to unmatched
    end
  end
  $\text{fn} \leftarrow |U|$
  $\text{fp} \leftarrow |\mathcal{A}_{detect}| - \text{tp}$
  return $\langle \text{tp}, \text{fp}, \text{fn} \rangle$

Algorithm 2: Overlapping Segment Evaluation. For each ground truth anomaly, we search whether it overlaps with any event in the predicted set. If so, it counts towards a true positive; if not, it is considered a true negative. We then compute the total false positives to be the complement of true positives.

a subset of an anomaly. This is considered sufficient because domain experts monitor the signal and will investigate even an imprecise alarm, likely discovering the full anomaly in the process. Algorithm 2 illustrates our approach to counting:

1. **true positive** if a ground truth segment overlaps with the detected segment.
2. **false negative** if the ground truth segment does not overlap with any detected segments.
3. **false positive** if a detected segment does not overlap with any labeled anomalous region in the ground truth set.

The overlapping segment approach does not account for true negatives and is less sensitive to exact detection times. Moreover, in cases where an entire time series is determined to be anomalous, this method will return high metric scores. Therefore, it is important to regulate the length of the anomalies in this evaluation approach.



Strategy	**OS**	**PA**
True Positive	1	81
False Positive	1	5
False Negative	1	8
Precision	0.5	0.94
Recall	0.5	0.91
F_1 Score	0.5	0.92

Figure 4-6: Example showing ground truth anomalies and detected anomalies. Applying point-adjustment (PA) results in overestimated scores.

How does overlapping segment differ from point adjustment? Point adjustment (PA) is an evaluation mechanism proposed by Audibert et al. [2020], where any detected region that has a corresponding ground truth is expanded to cover the entire duration. The evaluation is then conducted using traditional sample-based metrics. One limitation of this strategy is that it produces inflated scores [Kim et al., 2022]. Figure 4-6 shows an example of how point adjustment alters the detected anomalies. With point adjustment, the strategy gives the impression that the set of detected anomalies is highly accurate ($F_1 = 0.925$). However, these scores can be considered exaggerated, since only one anomaly has been correctly identified. We show alternative, less biased metrics scores when using the overlapping segment (OS) approach.

4.4 Evaluation

In this section, we visually observe the outputs of `TadGAN`, `AER`, and `MixedLSTM` and compare them to each other. Moreover, we report their performance using the proposed range-based evaluation metrics.

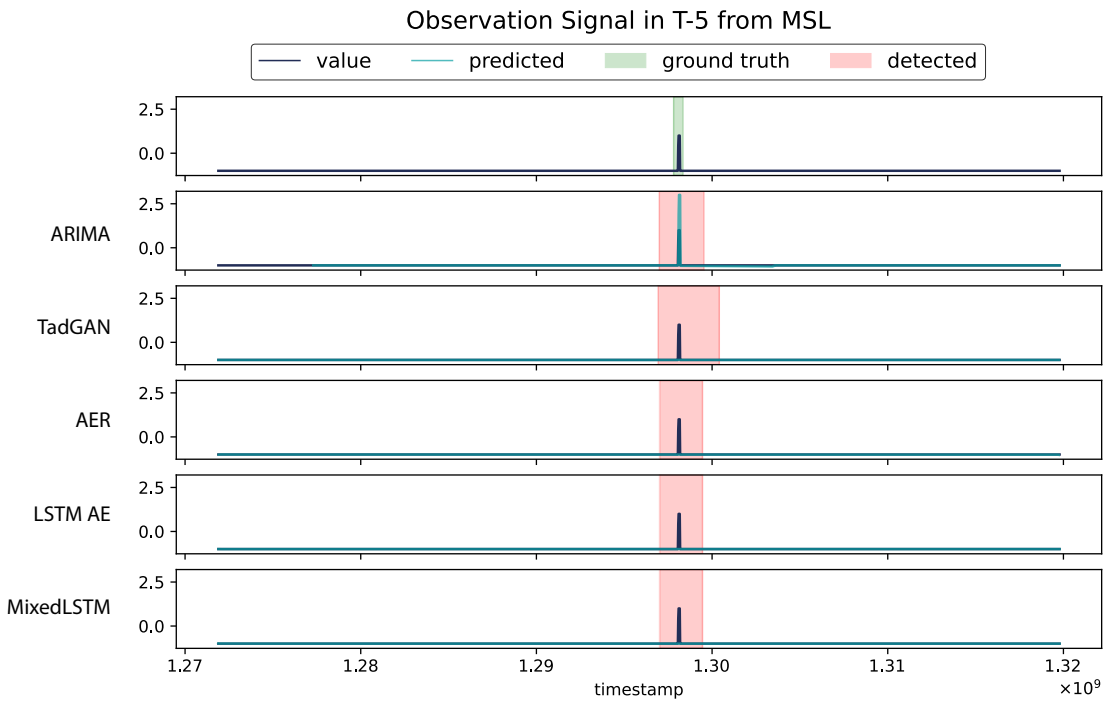


Figure 4-7: Visualizing `T-5` from MSL detections by `ARIMA`, `TadGAN`, `AER`, `LSTM AE`, and `MixedLSTM` where the signal has point anomalies.

Datasets. We evaluate the performance on 11 datasets from NASA, Yahoo, and NAB with known ground truths. Full details of these datasets are available in Section 3.1. Note that only NASA's data (MSL & SMAP) is multivariate with contextual signals. Therefore, the results of `MixedLSTM` on Yahoo and NAB are comparable to a vanilla LSTM auto-encoder.

Models. For our baseline, we look at `ARIMA` [Box and Pierce, 1970], a classical statistical method that forecasts the next step and can be used as a prediction-based model for time series anomaly detection [Pena et al., 2013]. In addition, we include `LSTM AE` in our visual inspections to clarify how things change after we include contextual signals in the model's architecture.

4.4.1 Qualitative Evaluation

We qualitatively assess the performance of our proposed models in comparison to `ARIMA`. We present visual results in Figure 4-7 for out-of-range anomalies and Figure 4-8 for contextual ones.

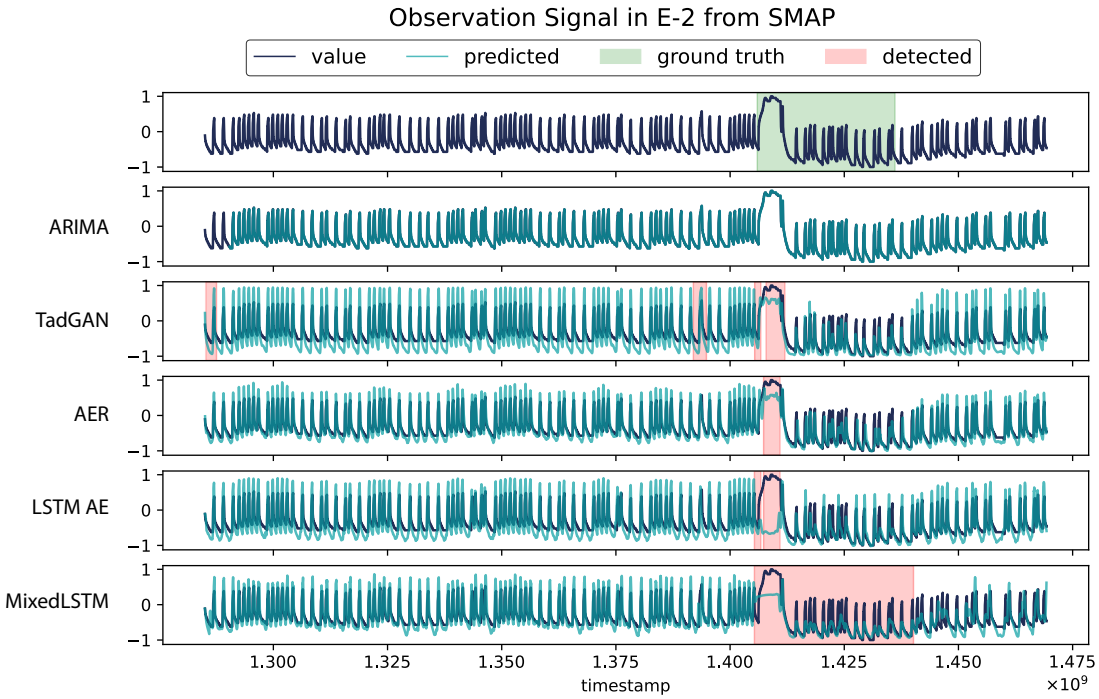


Figure 4-8: Visualizing `E-2` from SMAP detections by `ARIMA`, `TadGAN`, `AER`, `LSTM AE`, and `MixedLSTM` where the signal has contextual anomalies.

Modeling Observation Only. Figure 4-7 shows a simple example, where all three models were able to detect the extreme point change in the signal. In Figure 4-8, we visualize a more challenging example of a contextual anomaly. We see that `ARIMA` overfits to the signal it is trying to predict, making it unable to find the real anomaly. On the other hand, while `TadGAN` partially found the correct anomaly, it also raised two false alarms. `AER` is the best out of these three models at detection; however, it still missed a portion of the ground truth anomaly.

All the above models focus on modeling the observation signal independently, which limits the model's understanding of the expected behavior. Next, we investigate modeling control signals with `MixedLSTM`.

Modeling Contextual Variables. After adding contextual information for `E-2`, we can see how `MixedLSTM` was able to find the correct anomaly through more accurate detection, visualized in Figure 4-8. For a clearer depiction, Figure 4-9 shows an example of multivariate time series `E-3` from SMAP. We train `MixedLSTM` on the time series using only the observation signal first. Then, we include control and status signals in addition to observation. After

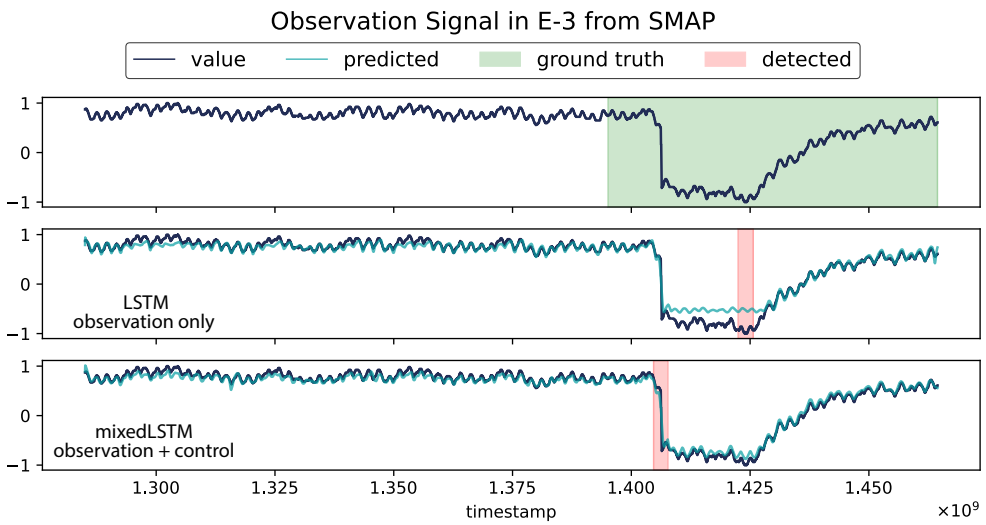


Figure 4-9: Visualizing `E-3` on `MixedLSTM` under two conditions. After including more contextual information, the model detected the anomaly at an earlier time.

including contextual information, we notice two clear improvements: the model learned a better representation of the observation time series (with a mean squared error of 0.004 compared to 0.013), and the anomaly was detected earlier .

4.4.2 Metric Evaluation

Comparing Range-based Metrics. For the previously shown examples (`T-5`, and `E-2`), we compare the scores of each model using weighted segment and overlapping segment approaches, in order to clarify their differences. Note that the NASA data is regularly sampled; thus, sample-based metrics and weighted segment will result in the same scores. In `T-5`, all models were able to detect the ground truth anomaly with overlapping segment, achieving an $F_1 = 1.0$. Weighted segment requires a more nuanced evaluation, as precision is low because additional time intervals were considered anomalous. On `E-2`, where `ARIMA` failed to find any anomalies due to overfitting, all scores are zero. Additionally, we can see how the remaining models (which did find the correct anomaly) score differently. First, `TadGAN` includes false alarms, reducing its precision score. While `AER` and `LSTM AE` do not include false positives, they did not detect the entire duration of the anomalous interval, making `MixedLSTM` the highest scoring model. The weighted segment approach is more rigorous than the overlapping segment approach and evaluates precise detections.

		T-5			E-2		
		Precision	Recall	F_1	Precision	Recall	F_1
Weighted Segment	ARIMA	0.212	1.000	0.350	0.000	0.000	0.000
	TadGAN	0.155	1.000	0.269	0.462	0.164	0.242
	AER	0.221	1.000	0.362	0.876	0.146	0.250
	LSTM AE	0.200	1.000	0.333	0.993	0.199	0.332
	MixedLSTM	0.200	1.000	0.333	0.909	0.997	0.951
Overlapping Segment	ARIMA	1.000	1.000	1.000	0.000	0.000	0.000
	TadGAN	1.000	1.000	1.000	0.333	1.000	0.500
	AER	1.000	1.000	1.000	1.000	1.000	1.000
	LSTM AE	1.000	1.000	1.000	1.000	1.000	1.000
	MixedLSTM	1.000	1.000	1.000	1.000	1.000	1.000

Table 4.1: Precision, Recall, and F_1 scores on T-5 and E-2 results using weighted– and overlapping-segment approaches.

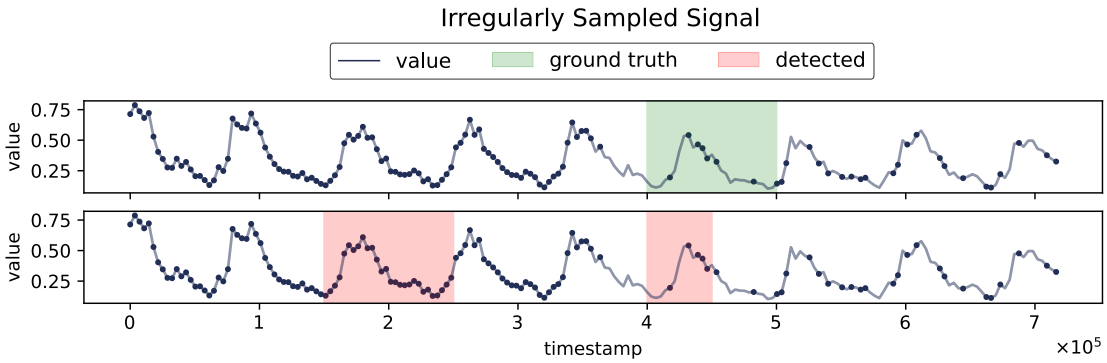


Figure 4-10: Example of an irregularly sampled signal.

Comparing Range-based and Sample-based Metrics. To illustrate the importance of range-based metrics, we show an example of an irregularly sampled signal in Figure 4-10. Initially, this signal is frequently sampled (shown as points in the figure), but it becomes more sparse with time. The ground truth anomaly occurs at an infrequently sampled duration, between timestamps 4×10^5 and 5×10^5. Moreover, the algorithm detects two anomalies in this signal – one that is not a true anomaly, and one that partially overlaps with the ground truth. Table 4.2 shows the precision, recall, and F_1 scores for each strategy. For sample-based evaluation, we directly use `scikit-learn`'s implementation. We observe the following: (1) because the false positive interval has many samples, the precision score is low for sample-based evaluation; (2) the recall score is high in comparison, even though half

		Precision	Recall	F_1
	Sample-based	0.152	0.714	0.250
Range-based	Weighted Segment	0.333	0.500	0.400
	Overlapping Segment	0.500	1.000	0.667

Table 4.2: Comparison between sample– and range-based metrics.

of the duration of the true anomaly is not detected – this is because only two samples are marked as false negatives. On the other hand, the weighted segment approach provides a more robust evaluation, even if the number of samples present in the data is irregularly sampled. Similarly, the overlapping segment-based approach evaluates the performances as one true detection and one false detection, making it easy to interpret detection performance.

4.4.3 Performance Evaluation

Overall Performance. We compare the performance of our proposed models against `ARIMA`, a classical approach to time series anomaly detection. Moreover, we investigate the anomaly scoring techniques for `TadGAN`, `AER`, and `MixedLSTM` and whether they produce more accurate results. Table 4.3 shows the overall F_1 scores of each model in comparison to `ARIMA`.

- *All deep learning models outperform `ARIMA` on NASA's data.* `MixedLSTM` is the best-performing modeling on MSL and `AER` is the best-performing on SMAP, surpassing `ARIMA` by 25.7% and 44.9% respectively – a drastic improvement in detection scores.

- *`ARIMA` achieves competitive scores on Yahoo S5.* `ARIMA` reaches F_1 scores that are near the best scores on A1 and A4. Moreover, it surpasses both `MixedLSTM` and `TadGAN` on A3. However, `ARIMA` has a lowest average F_1 score overall.

- *`AER` is overall the best performing model.* `AER` achieves an F_1 score of 0.739 on average, surpassing the next-best model, `TadGAN`, by 12.9%.

- *Reconstruction-based models struggle to find out-of-range anomalies.* We can see this clearly in A3 and A4, where `TadGAN` and `MixedLSTM` have significantly lower scores

	NASA		Yahoo S5				NAB					
	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	$\mu \pm \sigma$
`AER`	0.587	**0.775**	**0.780**	**0.988**	**0.869**	**0.686**	0.769	**0.750**	0.733	**0.611**	0.581	0.739 ± 0.123
`TadGAN`	0.581	0.652	0.612	0.859	0.408	0.321	0.667	0.687	**0.783**	0.529	**0.606**	0.610 ± 0.153
`MixedLSTM`	**0.692**	0.703	0.620	0.874	0.460	0.227	0.667	0.750	0.615	0.471	0.533	0.601 ± 0.174
`ARIMA`	0.435	0.326	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513	0.578 ± 0.174

Table 4.3: F_1 Scores of `TadGAN`, `AER`, and `MixedLSTM` on 11 datasets.

	NASA		Yahoo S5				NAB					
	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	$\mu \pm \sigma$
						`TadGAN`						
$\mathcal{C}_x$	0.393	0.472	0.285	0.118	0.008	0.024	0.625	0.000	0.350	0.167	0.548	0.272 ± 0.224
f_{point}	0.394	0.556	0.568	0.224	0.165	0.156	0.545	0.593	0.583	0.387	0.415	0.417 ± 0.170
f_{area}	0.299	0.532	0.554	0.228	0.145	0.136	0.545	0.593	0.545	0.345	0.415	0.394 ± 0.173
f_{dtw}	0.344	0.553	**0.612**	0.459	0.164	0.149	0.250	0.577	0.640	0.519	0.545	0.437 ± 0.180
$\mathcal{C}_x \times f_{point}$	0.521	0.620	0.591	0.637	**0.408**	**0.321**	0.588	0.629	0.621	**0.529**	0.541	0.546 ± 0.100
$\mathcal{C}_x + f_{point}$	0.575	**0.652**	0.557	0.637	0.139	0.168	**0.667**	**0.687**	0.636	0.390	**0.606**	0.519 ± 0.198
$\mathcal{C}_x \times f_{area}$	**0.581**	0.588	0.588	0.621	0.284	0.266	0.625	0.648	0.625	0.412	0.485	0.520 ± 0.140
$\mathcal{C}_x + f_{area}$	0.480	0.606	0.561	0.579	0.118	0.097	**0.667**	0.667	0.667	0.457	0.588	0.499 ± 0.206
$\mathcal{C}_x \times f_{dtw}$	0.535	0.619	0.604	0.804	0.393	0.283	0.615	0.625	0.621	0.450	0.556	0.555 ± 0.139
$\mathcal{C}_x + f_{dtw}$	0.500	0.623	0.586	**0.859**	0.250	0.159	0.462	0.643	**0.783**	0.368	0.543	0.525 ± 0.211
						`AER`						
f_{bi}	0.494	0.685	0.705	0.923	**0.869**	**0.686**	0.500	0.541	0.688	**0.611**	0.556	0.666 ± 0.141
$f_{bi} + f_{rec}$	0.488	0.680	0.714	0.936	0.719	0.553	0.500	**0.750**	**0.733**	0.606	0.559	0.658 ± 0.133
f_{rec}	0.500	0.683	0.707	**0.988**	0.620	0.416	0.444	0.644	0.692	0.571	0.519	0.617 ± 0.159
$f_{bi} \times f_{rec}$	**0.587**	**0.775**	**0.780**	0.959	0.752	0.572	**0.769**	0.635	0.621	0.606	**0.581**	0.694 ± 0.122

Table 4.4: `TadGAN` and `AER` performance under various configurations.

than the prediction-based models. This occurs for two reasons (1) the model can suffer from overfitting to anomalies; (2) the smoothing function applied to the error vector can erase the contribution of a point anomaly.

TadGAN Ablation Study. We test multiple reconstruction error functions, including: point-wise (f_{point}) errors, area (f_{area}) errors, and dynamic time warping (f_{dtw}) errors, as presented in Section 2.4. Moreover, we investigate how Critic ($\mathcal{C}_x$) scores contribute to anomaly discovery. We present these results in Table 4.4.

- *Using Critic alone is unstable.* Critic scores have the lowest average F1 score (0.272) and the highest standard deviation (0.224). While Critic alone can achieve a good performance in some datasets, such as SMAP and Art, its performance may also be unexpectedly bad, such as in A2, A3, A4, AWS and Traf. No clear shared characteristics are identified among these five datasets. For example, some datasets contain only

contextual anomalies (AWS, Traf), while other datasets, like A3 and A4, have point anomalies as the majority types. One explanation could be that because the number of anomalies in the mentioned dataset is comparably high, the Critic mistakenly learns that anomalous segments are normal.

- *f_{dtw} slightly outperforms the other two reconstruction error types.* Of all the variations, $\mathcal{C}_x \times f_{dtw}$ achieves the best average score (0.555 ± 0.139). In addition, f_{dtw} on its own outperforms the other reconstruction scores: its score is 0.437, compared to 0.417 for f_{point} and 0.394 for f_{area}. $\mathcal{C}_x \times f_{point}$ achieves the second-best average, with a score of 0.546 ± 0.100. Further, its standard deviation is smaller, indicating that this combination is more stable. Given that f_{point} is faster to compute than f_{dtw}, $\mathcal{C}_x \times f_{point}$ combination is the safe choice when encountering new datasets without labels.

- *Combining Critic outputs and reconstruction errors improves performance in most cases.* For all datasets except A1, combinations achieve the best performance. Let's take the SMAP dataset as an example: We observe that combining f_{point} with the Critic score results in a score of 0.652, despite the fact that f_{point} alone results in an F_1 score of 0.556 (.096$\downarrow$), and Critic ($\mathcal{C}_x$) alone results in an F_1 score of 0.472 (0.18$\downarrow$). In addition, we find that after combining with Critic scores, the average F_1 score improves for each of the individual reconstruction error computation methods. However, it's also interesting to note that for dataset A1, using f_{dtw} errors alone achieves the best performance. Note that A1 is the only real dataset from Yahoo S5, while the others (A2, A3, A4) are synthetic.

- *Multiplication is a better option than convex combination.* Multiplication consistently leads to a higher average F_1 score than convex combination when using the same reconstruction error type (e.g., $\mathcal{C}_x \times f_{point}$ and $\mathcal{C}_x + f_{point}$). Multiplication also has consistently smaller standard deviations. Thus, multiplication is the recommended way to combine reconstruction scores and Critic scores. This is explained by the fact that multiplication can better amplify high anomaly scores.

AER Ablation Study. We look at bi-directional (f_{bi}) errors and reconstruction (f_{rec}) errors and their combination approaches, presented in Table 4.4.

- *Combining ($f_{bi} \times f_{rec}$) anomaly scores achieves the highest averaged F_1 score of all method combinations.* The product ($f_{bi} \times f_{rec}$) combination of prediction-based and reconstruction-based anomaly scores produced the highest F_1 scores for 5 of 11 datasets, as shown in Table 4.4. Most of these datasets were non-synthetic, including MSL, SMAP, A1, and Tweets. In terms of average F_1 scores, this combination method outperformed the convex ($f_{bi} + f_{rec}$) combination by 5.5%, the (f_{bi}) combination by 5.2%, and the reconstruction-based only (f_{rec}) combination by 12.6%. Additionally, by excluding the A3 and A4 synthetic datasets (which have many point anomalies), we achieved an average F_1 score of 0.701 for the product ($f_{bi} \times f_{rec}$) combination – a 10.5% increase over the (f_{rec}) combination, which achieves a score of 0.634. These results support the idea that mixed anomaly scores offer more information than reconstruction-based anomaly scores in general, as well as offering more information than prediction-based anomaly scores for cases outside identifying point anomalies.

- *Prediction-based (f_{bi}) anomaly scores alone perform better on datasets with mostly point anomalies.* Bi-directional scoring produced the highest F_1 scores for datasets like A3 and A4 that have mostly point anomalies.

- *The combination method for each dataset should be chosen according to the use case.* We recommend that users default to using product ($f_{bi} \times f_{rec}$) anomaly scores, and use prediction-based (f_{bi}) scores only when they primarily want to identify point anomalies. The AER model reports the F_1 scores of AER (f_{bi}) for the A3 and A4 datasets with mostly point anomalies, and AER ($f_{bi} \times f_{rec}$) for the other datasets, even though these might not be the best combination methods according to the ablation study. This is a similar conclusion to that of the `TadGAN` ablation study. Because anomaly detection is an unsupervised problem, real-world datasets come without labels. Hence, it is impossible to retroactively tune the best method to calculate anomaly scores for each dataset.

MixedLSTM Ablation Study. We investigate whether introducing contextual signals into the model improves detection. `LSTM AE` is similar to `MixedLSTM` without including the additional term in its loss function. Table 4.5 shows these results for NASA's data.

	MSL			SMAP		
	Precision	Recall	F_1	Precision	Recall	F_1
`LSTM AE`	0.486	0.472	0.479	0.639	**0.687**	0.662
`MixedLSTM`	**0.925**	**0.552**	**0.692**	**0.886**	0.582	**0.703**

Table 4.5: Ablation results on NASA's satellite data (MSL & SMAP) using `LSTM AE` and `MixedLSTM`, which views the performance with observation and observation + contextual signals, respectively.

- ***`MixedLSTM`** generally improves on **`LSTM AE`**, showing the importance of contextual signals.* We evaluate both models on NASA's satellite telemetry dataset, which is comprised of MSL & SMAP. These datasets are multivariate time series consisting of one observation signal along with control/status signals. MSL contains 55 dimensions, while SMAP has 25. We summarize the results in Table 4.5, with the highest scores shown in bold. We find that in terms of F_1 score, `MixedLSTM` outperforms `LSTM AE` by 21.3% on MSL and by 4.1% on SMAP.

4.5 Conclusion

This chapter introduced a formal definition of unsupervised anomaly detection in time series data and presented three models designed to advance different aspects of the detection task: `TadGAN`, which leverages adversarial training; `AER`, which jointly learns to predict and reconstruct time series; and `MixedLSTM`, which explicitly models interdependencies across dimensions. We further proposed two methodologies for evaluating detection quality in the presence of ground truth labels. Empirical results demonstrated that all three models outperform an `ARIMA` baseline across all datasets, with `AER` achieving the highest overall F_1 score and `MixedLSTM` exhibiting notable improvements on the MSL dataset.

Chapter 5

Systems for Unsupervised Time Series Anomaly Detection

This chapter details the systems we developed for the task of time series anomaly detection, and evaluates the performance of those systems and their internal components.

- Section 5.1 introduces *Orion*, a system that abstracts models, evaluates performance, and provides users with a simple application programming interface.
- Section 5.2 extends Orion to develop our benchmarking system, *OrionBench*.

5.1 Orion – A Machine Learning System for Unsupervised Time Series Anomaly Detection

Many algorithms have been developed to address the task of time series anomaly detection, ranging from statistical methods to machine learning techniques [Hodge and Austin, 2004, Chandola et al., 2009, Goldstein and Uchida, 2016, Habeeb et al., 2019]. Despite the existence of these algorithms, no system provides users with complete functionality. Existing systems often fail to encompass an end-to-end detection process, to facilitate comparative analysis of various anomaly detection methods, or to allow users to engage with the system via functional APIs. We previously provided a summary of current systems in Table 3.2, which

gives a through review of the features available in current systems. This precludes current methods from being used by end users who are not ML experts.

When focusing on *usability*, we often refer to a highly successful library such as scikit-learn [Pedregosa et al., 2011], which made machine learning algorithms available to all users, including models for classification, regression, and clustering. With over 62.6k stars on GitHub and averaging 105 million downloads on pypi on a monthly basis, this has become the default choice for rapid prototyping as well as for production pipelines. Its widespread adoption is a testament to how easy it is to use.

We design Orion to provide the end-to-end solution for end users, and base some of our design choices on lessons from scikit-learn's development. We note that scikit-learn reached its current structure through an iterative process, and that many necessary elements were initially missing from the library. Throughout this section, we continue to draw parallels between Orion and scikit-learn. Orion is comprised of a series of components. We first introduce the Orion input format (Section 5.1.1), then define the machine learning stack (Section 5.1.2), which contains primitives, templates, and pipelines. Then we introduce the core interaction (Section 5.1.3), which is the main entry point that allows users to select and train pipelines and save them. This is followed by a description of the hyperparameter tuning component (Section 5.1.4) and anomaly annotation using MTV, our visualization system (Section 5.1.5).

5.1.1 Orion Data Format

timestamp	value
1222819200	215
1222840800	124
⋮	⋮
1334905600	15

Table 5.1: Univariate time series.

timestamp	v_1	$\cdots$	v_k
1222819200	13		12
1222840800	7		34
⋮	⋮	$\cdots$	⋮
1334905600	31		52

Table 5.2: Multivariate time series.

Given the formulation of the problem (shown in Section 4.1), Orion has to follow this definition. Therefore, we standardize the system's input and output.

Input. The input is a `pandas` dataframe containing at least two columns:

- `timestamp`: an integer column, with the time of the observation in Unix Time format.
- `value(s)`: a float column, with the observed value at the indicated timestamp. Each value is represented as a different column.

Table 5.1 and 5.2 show an example of a univariate and multivariate dataframe, respectively.

start	end	score
1222905600	1223208000	0.0524
1402012800	1403870400	0.6225

Table 5.3: Anomalies format.

Output. A list of intervals. Each interval contains at least two entries:

- `start`: timestamp where the anomalous interval begins.
- `end`: timestamp where the anomalous interval ends

Optionally, there can be a third entry that contains the anomaly score (the likelihood that the interval is anomalous). Table 5.3 shows an example of two detected anomalies, presented in the Orion format. In this example, the second detected anomaly is more likely to be anomalous than the first.

5.1.2 Machine Learning Stack

The machine learning model is the center of any ML-based solution. However, the model alone cannot perform a task from beginning to end. This was initially neglected in many libraries, including scikit-learn, where originally only the classification or regression model was available, without key pre- and post- operations. Later on, data transformations, such as normalization, were introduced as pre-processing functionalities.

With Orion, we convert these machine learning algorithms into standardized end-to-end programs, which we call *pipelines*. Pipelines take a univariate or multivariate signal as an input $\mathcal{X}$ and use it to generate an array of intervals $\mathcal{A}$ representing the anomalies discovered.

A large number of models are available, and in most cases, the user is interested in finding anomalies but doesn't really care about the underlying method. Therefore, our pipelines are built end-to-end, such that Orion operates independently of what pipeline the end user selects. This is similar to how scikit-learn is model-impartial. To understand the composition of the pipelines, we describe their basic building blocks, or *primitives*, below.

Primitives

Primitives are reusable software components [Smith et al., 2020]. A primitive receives data in the form of a specified input, performs an operation, and returns a calculated output. Each primitive is responsible for a single task, ranging from data transformation to signal processing to machine learning modeling to error calculation. It is possible to build complex pipelines by stacking primitives on top of one another. Each primitive is represented through a `json` file. This file has associated metadata, including annotations such as the name of the primitive, the description and documentation link, and the engine category. Figure A-1 in the Appendix shows an example `json` primitive template. After mining numerous papers for unsupervised time series anomaly detection, we noticed a common theme. Most algorithms include processes that can be categorized into pre-processing, modeling, and post-processing operations. Orion covers these three engines:

Pre-processing. Time series are rarely handled in their raw form. Before using a signal, the data must be transformed through pre-processing. A pre-processing primitive can be used to scale the signal, impute missing values, or prepare training examples.

Modeling. Once the signal has been processed, we can start training a model. There are different techniques for modeling. In time series anomaly detection, we are interested in predicting or reconstructing the signal so that we can have a *generated* signal (see Figure 2-3). Models range from a multilayer perceptron, to auto-encoders, to transformers, to diffusion models.

Post-processing. After obtaining the *generated* signal, we use discrepancies between the generated and the real signal to find anomalies. We refer to this process as error calculation. Post-processing primitives output intervals containing potentially anomalous sub-sequences alongside the probability that they are anomalous.

We note that scikit-learn does not support any post-processing operations, even though they are crucial for predictive modeling. For example, in classification, mapping model outputs from logits to their respective binary classes is often defaulted to 0.5 or left for the user to optimize. We argue that this is a very important step that has gone missing. From the perspective of anomaly detection, without proper thresholding of error values, the pipeline will produce unintended anomalous events.

Modularly designed engines can re-use primitives between, within and across pipelines. This reduces the number of lines of code – and thus error potential – and increases transparency. Having a granular definition also encourages best practices such as proper documentation, unit tests, and validation. Contributors can integrate a new primitive into Orion without modifying an entire pipeline.

Primitive Hyperparameters

Hyperparameters of primitives can be categorized as *fixed* or *tunable*. Fixed hyperparameters are specified, with their type as a predefined default value when initializing the primitive. On the other hand, tunable hyperparameters have an additional section defining the search space of said hyperparameter. Both settings are configurable at the time of initialization.

Pipelines

Pipelines are end-to-end programs composed of primitives. Each pipeline is translated into a dedicated computational graph in which every step becomes a primitive, edges depict data flow between primitives, and the overall structure forms a directed acyclic graph (DAG), similar to the examples shown in Figure 5-1. In this paper, the term "pipeline" always refers to a program tasked with identifying anomalies in time series data. Primitive and pipeline structures have been successfully adopted in many other applications, including healthcare [Alnegheimish et al., 2020, Smith et al., 2020].

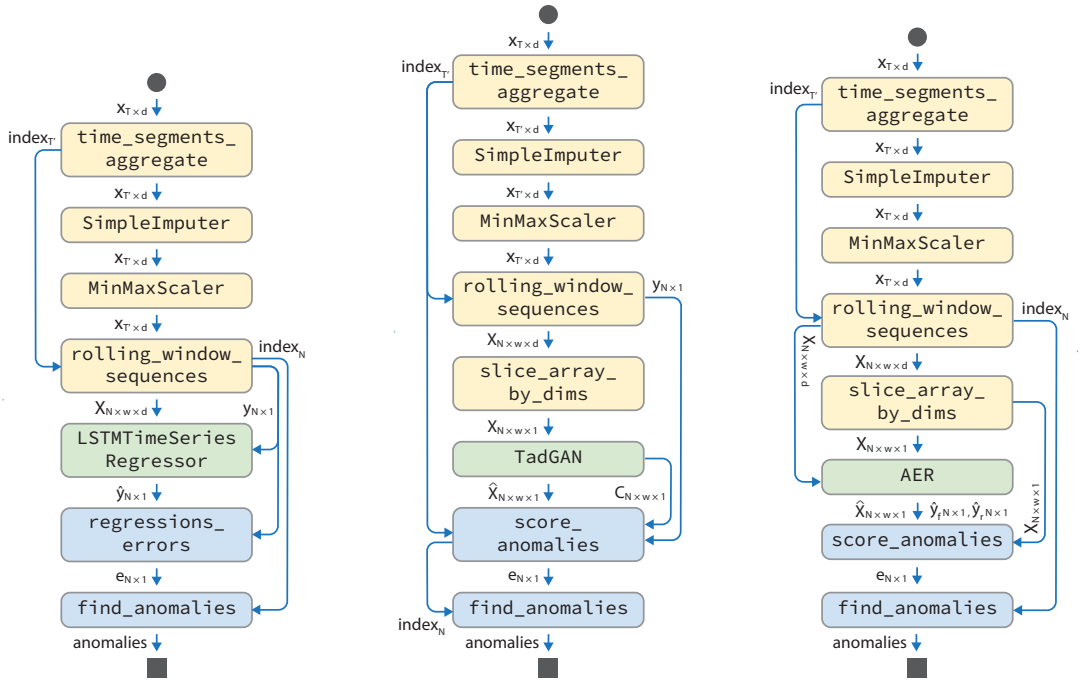
In most cases, pipelines require primitive hyperparameters to be set based on the dataset. To satisfy this requirement, we introduce a *template* concept where a template $\mathcal{T} = \langle V, E, \Lambda \rangle$, V is a set of pipeline steps, E is a set of edges between steps to represent data flow, and Λ is

the joint hyperparameter space for the underlying primitives [Smith et al., 2020]. Following this definition, a pipeline is a configured template with a fixed hyperparameter setting $P = \langle V, E, \lambda \rangle$ where $\lambda \in \Lambda$ is a specific set of hyperparameters. This definition allows us to easily create and manipulate pipelines, allowing them to be used with a wide range of signals. More importantly, it gives us visibility into which hyperparameters are altered when the pipeline is run on one dataset versus another. This transparency is crucial to making our results reproducible.

The importance of pipelines as a concept made scikit-learn introduce their own definition of a pipeline. Their type of pipeline allows users to sequentially apply a list of data transformers for pre-processing, and can additionally include an estimator at the end for predictive modeling. This pipeline design is limited in the sense that the data flow between many transformations, and later the model, is limited to a single data object, whereas our design renders as a directed acyclic graph where many data objects can flow between primitives. Moreover, our pipelines are *named pipelines*. end users select an Orion pipeline by specifying the name of the pipeline, e.g. "aer." This is particularly useful since end users are typically interested in utilizing off-the-shelf pipelines, rather than composing their own as is done in scikit-learn.

Dissecting Pipeline Examples

The pipeline in Figure 5-1a uses a Long Short-Term Memory (LSTM) network to predict data values at future timestamps. It is an equivalent representation to the LSTM non-parametric thresholding (`LSTM DT`) model proposed by Hundman et al. [2018]. We first take a raw signal $\mathcal{X}$ and feed it into the `time_segments_aggregate` to produce $\mathcal{X}' = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{T'}\}$ where the time intervals between $\boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_t$ are equal. Then we scale the data $\mathcal{X}' \in [-1, 1]$ and impute missing values using the mean value of the signal. After that, we create the training window sequences. This step produces a dataset of prediction pairs $\mathcal{D} = \{(\boldsymbol{x}^{(i)}_{1\ldots w}, y^{(i)}) \mid y \coloneqq x^d_{w+1})\}_{i=1}^N$, where d is the target dimension we want to predict. We aim to learn an LSTM model $y^{(i)} \approx f_\theta(\boldsymbol{x}^{(i)})$. Once the network is trained, we generate $\hat{\boldsymbol{y}} = \{\hat{y}_t \mid \forall\, t \in T'\}$ and compute the discrepancies using `regression_errors`, which is an absolute point-wise difference $|\hat{\boldsymbol{y}} - \boldsymbol{y}|$. Lastly, we use a dynamic threshold on error values to find anomalous



(a) `LSTM DT` (b) `TadGAN` (c) `AER`

Figure 5-1: Directed acyclic graphs (DAGs) of Orion pipelines. (a) LSTM forecaster with non-parametric dynamic threshold; (b) Generative Adversarial Networks (GAN) for time series anomaly detection; (c) AutoEncoder with regression.

regions [Hundman et al., 2018].

In the previous example, we implemented a pipeline for an existing algorithm. We can also implement novel algorithms as pipelines, such as TadGAN and AER, which are contributions made by Orion developers shown in Figure 5-1b and Figure 5-1c. Pre-processing is similar to `LSTM DT`, with an added primitive `slice_array_by_dims` to select a single dimension from $\mathcal{X}'$ to reconstruct. Moreover, both `TadGAN` and `AER` have their own modeling and scoring primitives as previously described in Section 4.2.

Customizing pipelines is fairly easy. Users can configure or even replace a primitive. For example, to use z-score normalization, users can swap the primitive `MinMaxScaler` with `StandardScaler`, where both primitives utilize scikit-learn's implementation.

Pipeline Hub

Orion stores a collection of end-to-end anomaly detection pipelines that work with state-of-the-art methods. We have incorporated `ARIMA` [Box and Pierce, 1970, Pena et al., 2013],

`LSTM DT` [Hundman et al., 2018], `LSTM AE` [Malhotra et al., 2016], `TadGAN` [Geiger et al., 2020], `AER` [Wong et al., 2022], `AT` [Xu et al., 2022b] and more. Figure B-2 in the Appendix depicts the graphs associated with these pipelines. Moreover, we provide "pipelines" that can connect to existing anomaly detection services, such as Microsoft Azure's anomaly detection service (`Azure AD`) [Ren et al., 2019]. This set of pipelines is easily extendable and can be expanded further.

Reusability of Primitives and Pipelines

Let's consider the primitive `find_anomalies` — which applies a windowed thresholding technique to find anomalies from error values. The primitive provides the option of using dynamic thresholding, as proposed by Hundman et al. [2018], or fixed thresholding. All pipelines produce an error vector, and almost all pipelines utilize this primitive, with the exception of `AT`, which has its own thresholding approach, and `Azure AD`, which is a black-box service. We note that `AT` did not propose a new thresholding technique in their paper, and use a naive thresholding approach in their implementation. To determine the threshold, they choose a value that marks r proportion of the data as anomalous. For example, if $r = 1.0$, then the top 99% of error values are flagged as anomalous.

With the `find_anomalies` primitive, a robust approach to finding out-of-range and contextual anomalies, we provide all pipelines with a consistent thresholding mechanism. Moreover, this allows researchers to improve other pipeline components. For example, `AER` uses the `find_anomalies` primitive, which is similar to `LSTM DT` and `LSTM AE` but outperforms both of them. The innovation we see in `AER` is solely focused on the modeling and anomaly scoring primitives.

Moreover, if we compare the `Dense AE` and `LSTM AE` pipelines (visualized in Figure B-2), we notice that they are identical copies of one another, with the exception of the `Seq2Seq` primitive. This shows that in order to build a completely new pipeline with similar architecture, we simply needed to change one primitive in the `json` file from `DenseSeq2Seq` to `LSTMSeq2Seq`, which effectively means that we have changed the auto-encoder layers from dense layers to LSTM layers.

```python
from orion import Orion
from orion.data import load_signal

train_data = load_signal('S-1-train')
orion = Orion(
    pipeline='aer'
)

# train the pipeline
orion.fit(train_data)

# incoming data
new_data = load_signal('S-1-new')

# detect anomalies
anomalies = orion.detect(new_data)
```

(a)

```python
from orion import Orion
from orion.data import load_signal, load_anomalies

# load pre-trained model
path = 'path/to/model'
orion = Orion.load(path)

# load data & ground truth anomalies
data = load_signal('S-1')
anom = load_anomalies('S-1')

# evaluate the performance
metrics = ['precision', 'recall']
score = orion.evaluate(data, anom, metrics=metrics)
```

(b)

Figure 5-2: Usage with python SDK. (a) End-to-end anomaly detection pipeline. The user first loads the data, either externally or with `load_signal`. Then the user select the desired pipeline for detection. In this example, we use `aer`. The user then trains the pipeline using `orion.fit`, and similarly, detects anomalies using `orion.detect`. (b) End-to-end evaluation of a pre-trained model. The user can pass the ground truth anomalies to `orion.evaluate` to measure the performance score.

5.1.3 Core Interaction

To address the *usability* of the system, we need to find the right level of abstraction to make interaction a pleasant experience. Orion's core provides a set of coherent application programming interfaces (APIs), allowing users to execute end-to-end processes. Given a signal $\mathcal{X}$, we want to obtain a set of detected anomalies $\mathcal{A}$. With Orion, this is straightforward. First, the user loads a signal that follows the input standard – (timestamp, values). We provide a helper function to load data from `csv` files. Next, the user selects the pipeline of interest from a suite of available options. To view the currently available pipelines, users can read the documentation or use `get_available_pipelines` to learn more about them. Once a pipeline is selected, it is trained on the signal using `orion.fit(train_data)`. To detect anomalies, the user then executes `orion.detect(new_data)` to produce a set of possible anomalies. Users can also use the evaluation mechanisms defined in Section 4.3 to view the performance of a pipeline if the ground truth anomalies are present. Figure 5-2 shows an example code for this process. Below, we detail the attributes and functionalities of the Orion class.

1. `orion`: To create an Orion instance, the user instantiates an object that specifies the pipeline and additional hyperparameters if needed. There are multiple ways to specify a pipeline:

    - providing an existing named pipeline, e.g. "aer."
    - providing a path to the `json` file of a pipeline.
    - providing a dictionary that specifies the contents of the pipeline.

    For Orion pipelines, we recommend the first option. An example instantiation is `orion = Orion(pipeline='aer')`. To support quick user testing of Orion, we select a default pipeline in cases where the user does not supply any specific pipeline, i.e. `orion = Orion()`. The default pipeline is `AER`, because it achieves the best performance overall (see Figure 5-8 in Section 5.3).

2. `orion.fit`: To train the pipeline – including all its trainable primitives – we use `orion.fit(train_data)`, where `train_data` is the training data that follows the Orion format (Section 5.1.1).

3. `orion.detect`: To use the pipeline for detecting anomalies, we run `orion.detect(data)`, which internally does the following:

    - If the pipeline is trained, it runs the inference part of the pipeline to produce anomalies.
    - If the pipeline is not trained, it proceeds to first execute `orion.fit(data)` on the same data object. This is similar to running `orion.fit_detect(data)`.

4. `orion.evaluate`: To allow users to evaluate the pipeline (if the ground truth is known), we provide a functionality that uses the weighted or overlapping segment evaluation strategies proposed in Section 4.3. Using `orion.evaluate(data, ground_truth)`, the pipeline will produce anomalies, then produce a report of the default metrics (f1, precision, and recall), calculated using

Input: template $\mathcal{T}$, dataset D, scorer function f, budget B.
Output: best hyperparameter λ^*
begin
  `init Tuner` // Bayesian Tuner from BTB
  $s^* \leftarrow +\infty, \lambda^* \leftarrow \varnothing$
  while $B > 0 \wedge s^* \neq f^*$ **do**
    $\lambda \leftarrow$ `Tuner.propose`$(\mathcal{T})$ // propose a set of hyperparameters
    $P \leftarrow (\mathcal{T}, \lambda)$ // construct pipeline
    $s \leftarrow$ `cross_validate`(f, P, D)
    `Tuner.record`(λ, s) // update tuner
    if $s < s^*$ **then**
      $s^* \leftarrow s$
      $\lambda^* \leftarrow \lambda$
    end
    reduce(B) // decrease budget
  end
  return λ^*

Algorithm 3: Automated hyperparameter optimization in Orion, searching for the best configuration of λ and evaluating pipelines using the scoring function f.

the weighted segment approach. Optionally, the user can also provide `train_data` to perform training, detection, and evaluation all in a single step.

5. `orion.save`: To utilize the pipeline for inference later on, the entire Orion object, including its trained pipeline, can be saved locally to a provided path `orion.save(path)`.

6. `orion.load`: To load an Orion instance from the path, the user runs `orion.load(path)`, which reloads the Orion object, with the pipeline maintaining the same weights it had at the save point.

Simple code execution, accomplished via `fit/detect/evaluate` functionalities and the pipeline, makes the framework unified, usable, and accessible – similar to the popular `fit/predict` interfaces for democratized libraries such as scikit-learn [Buitinck et al., 2013].

5.1.4 Hyperparameter Tuner

Hyperparameter tuning is instrumental to ML systems [Bergstra and Bengio, 2012, Bergstra et al., 2011]. To tune pipelines automatically (Figure 5-3), we integrate `BTB` [1], an open-

[1] https://github.com/MLBazaar/BTB

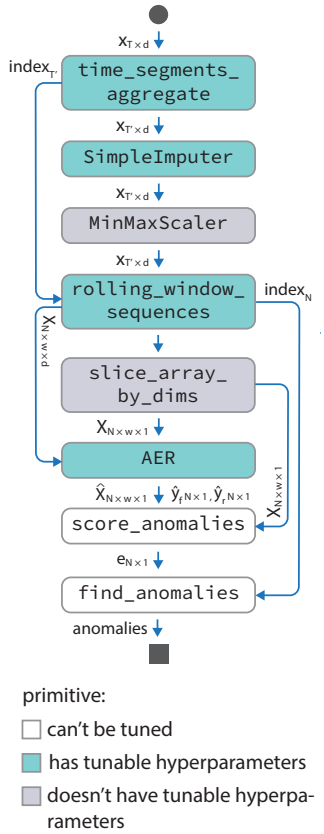
```python
from orion import Orion
from orion.data import load_signal

train_data = load_signal('S-1-train')
valid_data = load_signal('S-1-valid')

hyperparameters = {
    'AER': {
        'reg_ratio': {
            'default': 0.5,
            'range': [0, 1]
        }
    }
}

orion = Orion(
    pipeline='aer',
    hyperparameters
)

# tune the pipeline
orion.tune(train_data, valid_data, scorer='mse')
```

```python
hyperparameters = {
    'time_segments_aggregate': {
        'interval': 21600
    },
    'rolling_window_sequences': {
        'window_size': {
            'default': 250,
            'options': [100, 200, 250]
        }
    },
    'AER': {
        'epochs': 35,
        'reg_ratio': {
            'default': 0.5,
            'range': [0, 1]
        }
    },
    'find_anomalies': {
        'window_size_perc': 0.3,
        'fixed_threshold': false
    }
}
```

Figure 5-3: (left) Tuning usage with python SDK. `orion.tune` allows users to tune templates and select the best configuration for their instance using a scorer of their choice. (right) hyperparameter configuration in `json` format showing "fixed" hyperparameters and "tunable" hyperparameters with the search space of the tuner is defined using range/options parameters.

source and extensible framework with black-box Bayesian Optimization [Smith et al., 2020]. In short, the AutoML component of the framework aims to find the configuration of hyperparameters for a given pipeline template that best maximizes some set of objective functions. Given pipeline template $\mathcal{T}$ and an objective function f that assigns a performance score to pipeline P_λ with hyperparameters $\lambda \in \Lambda$, we recover $\lambda^* = argmax_{\lambda \in \Lambda} f(P_\lambda)$. We use `GPTuner`, which optimizes candidates using a Gaussian process meta-model, records evaluations, and proposes hyperparameters λ. We continue this search until our budget runs out or we have reached the optimal value. Algorithm 3 depicts the general flow of optimization in Orion.

Since Orion is an *unsupervised* framework, the tuner focuses on the sub-pipeline that attempts to generate the signal closest to the original one. To achieve this, users specify the pipeline template and evaluation metrics, such as `MSE`, `MAPE`, `MAE`, etc. for their objective function. Other objective functions, such as picking an estimator that produces the least number of anomalies, can also be specified. However, objective functions that are discrete are difficult to optimize; therefore, we do not facilitate this option.




```
tunable_hyperparameters = {
    'time_segments_aggregate': {
        'method': {
            'type': 'str',
            'default': 'mean',
            'options': ['min', 'max', 'sum',
            ↪  'mean', 'median']
        }
    },
    'SimpleImputer': {
        'strategy': {
            'type': 'str',
            'default': 'mean',
            'options': ['mean', 'median',
            ↪  'most_frequent']
        }
    },
    'rolling_window_sequences': {
        'window_size': {
            'type': 'int',
            'default': 250,
            'options': [100, 200, 250]
        }
    },
    'AER': {
        'reg_ratio': {
            'type': 'float',
            'default': 0.5,
            'range': [0, 1]
        },
        'learning_rate': {
            'type': 'float',
            'default': 0.001,
            'options': [0.1, 0.01, 0.001, 0.0001]
        },
    }
}
```

Figure 5-4: Orion inspects the pipeline and identifies pre-processing and modeling primitives (left). Then it returns a dictionary with each primitive, its set of tunable hyperparameters, and their corresponding search space (right).

On the back end, Orion inspects the pipeline and identifies pre-processing and modeling primitives based on the `json` metadata. Then, using the pipeline's helper function, `get_tunable_hyperparameters`, we gather all the hyperparameters that can be tuned and their corresponding search spaces. Figure 5-4 depicts which primitives in `AER` have tunable hyperparameters, and the corresponding set of these hyperparameters with their associated values. Executing `orion.tune` will return the best pipeline (the one with lowest `MSE`) based on these values.

Figure 5-5: Snapshot of MTV— the visualization component of Orion. Multiple signals are displayed at the top left as an overview, and the detailed view of one selected signal is shown at the bottom left. The right panel displays how users assign tags and comment on the signal of interest.

5.1.5 Visualization and Anomaly Annotation[2]

Another phase of our system's workflow is anomaly analysis. Our goal is to enable the user to annotate the set of flagged events, and to inspect detected anomalies. To achieve this, we introduce **M**ultivariate **T**ime Series **V**isualization (**MTV**), an interactive visual analytics system for anomaly investigation and annotation [Liu et al., 2022]. Figure 5-5 illustrates a snapshot of the system, which contains three major panels: the Signal Overview panel (top left), the Signal Focused View panel (bottom left), and the side panel (right). The side panel includes collapsible panels featuring a Periodic View panel, a Signal Annotation View panel, a Event Details panel, and a Similar Segments panel. We discuss the features of each panel below.

[2]This work was done collaboratively with Dongyu Liu.

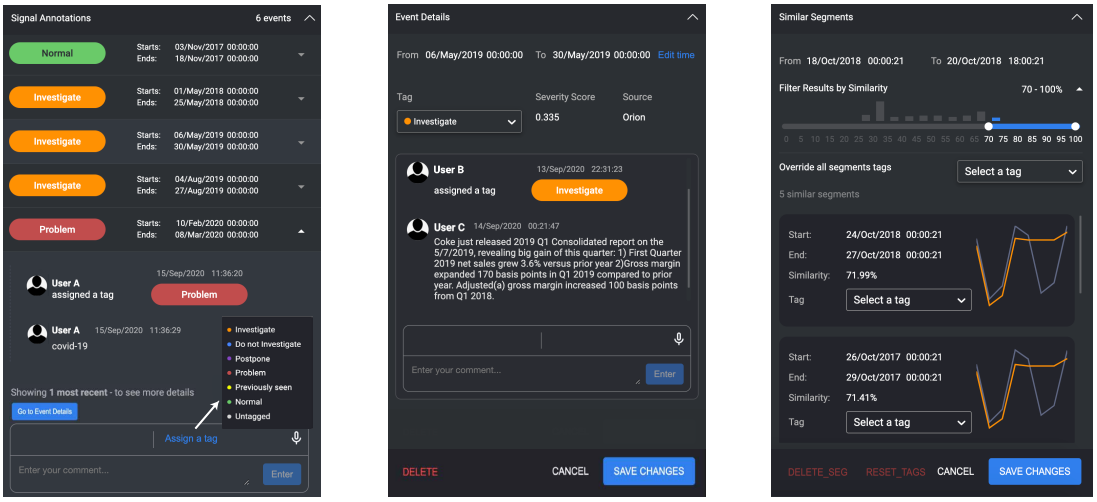
Signal Overview

The overview panel gives quick highlights of the multivariate time series and where anomalies were flagged. Given limited screen space, the design of this panel must be highly space-efficient. Therefore, we choose *small multiples*, a space-efficient technique that is widely used for visualizing multivariate time series [Javed et al., 2010]. Each variable in the multivariate time series is rendered as a separate line plot, where flagged anomalous events are shown by highlighting the curve with a warning color, e.g. yellow. This design allows experts to see the correlation between trends across signals, as well as co-occurring anomalous events. Events that happen across signals that are aligned closely together in time, known as co-occurring events, may indicate a larger and more important event.

Signal Focus View

This view expands a segment of the time series selected in the signal overview panel, displaying additional details that might be informative. In this panel, the curve of any signal flagged as an anomalous events is highlighted in the color of the tag associated with the event. To further enhance awareness, the color of the header bar double-encodes this tag information. A transparent gray background is added to make anomalies more visually apparent.

Interpretability. Intermediate results of the pipeline can provide users with the necessary information to understand why certain events were flagged as anomalous. The signal (whether generated though prediction or reconstruction) and the computed error vector are valuable details that can provide insights into pipeline behavior. We incorporate these results in this panel. The generated signal is visualized as an overlay line plot over the original signal, plotted as a thinner, bright yellow curve. Moreover, right above the line plot, we visualize the error vector as a chart highlighting the intensity of errors across the timeline. A thicker region indicates a larger discrepancy between the original values and the generated values. These visualization elements increase the transparency of the pipeline and enable experts to visually understand why a certain anomaly was identified.



(a) Signal Annotation (b) Event Details (c) Similar Segments

Figure 5-6: There are three sub-views available in the side panel: (a) the Signal Annotation View provides an overview of the annotations made for the selected signal (ordered by event time); (b) the Event Details View shows more details about one particular event, such as severity score and source; (c) the Similar Segments View displays similar segments to a selected event, allowing users to quickly perform annotations.

Periodic View

This view shown on the right hand side of Figure 5-5 is designed for analyzing periodic patterns of the selected signal. The table on the top summarizes the number of overall tags for a particular signal in focus. The bottom graph provides experts with a new perspective for exploring the periodical patterns of a signal as *glyphs*. The glyph design is inspired by the circular silhouette graph [Aigner et al., 2011], where each circle represents a different year, and the lines along the radius indicate the time series values in a clockwise manner. Three levels of periodical glyphs, corresponding with three different time granularities (i.e., year, month, and day), are proposed to support multi-scale analysis. If for one time period (year/month/day) a signal has an irregular shape or unusual spikes, this may indicate anomalies. In addition, we highlight the anomalies according to their tags by overlaying radial segments in the corresponding time periods. This enables experts to observe how anomalies are distributed across the year, month, or day. For example, in Figure 5-5, the red-tagged event occurred close to the start of the year, the green-tagged event was close to the end of the year, and the three orange-tagged events were close to the middle of the year.

Signal Annotations View

Figure 5-6a depicts this view, which provides an overview of all tags associated with the currently selected signal. From here, experts can quickly glance through the event information and what tags are assigned with these events. We design this tag similarly to GitHub issue labels, as a rounded rectangle where the background encodes the tag type and the text shows its associated name and meaning. Experts can click to open an event and explore the most recent annotations associated with it. To improve efficiency, experts are allowed to directly post their comments or assign a tag here. These events are chronologically ordered.

Event Details View

The Event Details View is shown in Figure 5-6b. Events can enter it by clicking it directly on the Side Panel (Figure 5-5) or by using the "Go to Event Details" button in the Signal Annotations View (Figure 5-6a). The Event Details View, from top to bottom, displays the starting and ending times, the tag information, the severity score, the source of the tag, and the comment box. The source can either be "Orion" (if it was assigned by an Orion pipeline), or "User's name" (if it was assigned manually by the user). In this view, the comment box shows all the historical annotations of the event, in contrast to the Signal Annotations View, where only the five most recent annotations are shown. Along with the other coordinated views, this view allows experts to perform in-situ annotation and communication, with all the necessary contextual information displayed on one screen. We have designed six general types of tags, plus the status "untagged," to assist collaborations between experts. Here are these tags and their meanings:

1. `Do not investigate` (action tag): We are not interested in this and have decided not to investigate.

2. `Postpone` (action tag): This event is interesting but low-priority; we will postpone its investigation.

3. `Investigate` (action tag): This event is interesting and we should investigate it now.

4. `Problem` (information tag): This is a new problem, and while we can describe it colloquially, we don't have a term for it yet.

5. `Previously seen` (information tag): This is a well-known problem that we have investigated before.

6. `Normal` (information tag): This event is normal, has an obvious explanation, and is not harmful.

This tag design comes from numerous design meetings with our domain experts. The first three action tags suggest the next step that should be taken pertaining to an event, while the last three explain what has already been decided. An event can be tagged differently over time. In practice, if an event is tricky, experts often initially use an action tag to spur team communication, and switch to a specific information tag later on, when a consensus has been reached.

Similar Segments View

This is the last available view in the side panel. The Similar Segments View provides experts with the ability to search the most similar segments for a selected event, as illustrated in Figure 5-6c. The idea is to try to find similar events to the currently selected events in order to facilitate tag propagation. We use a shape-matching algorithm based on euclidean distance or dynamic time warping [Liu et al., 2022]. The bar chart at the top of this view is used to filter the segments based on the similarity score, normalized into a percentage range from 0% to 100%. Below the chart is a list of segments showing more detailed information, such as start and end time. The graph on the right plots the returned segment line overlaid by the original selected event line, color-coded by its tag (in this case orange), allowing experts to visually compare how similar they are. This feature can be used for annotation sharing, decision support, and false alarm mitigation, in order to increase annotation efficiency.

Benefits of MTV

MTV supports standard operations such as multi-signal viewing and zoom functionalities. In addition, it allows for a multi-aggregation view, which allows a user to compare the signal at

different aggregation levels. These operations help experts understand why certain intervals have been flagged, and allow for modification and annotation. In addition, we provide a discussion panel so that team members can comment on or dispute the status of an event.

Expert annotations are extremely important for understanding whether certain events are truly anomalous. Moreover, these annotations are persistent, allowing future teams and users to understand why certain decisions were made. Although the sequence of discussions and actions that have led to the classification of an event are an important part of forming the canonical logic behind a decision, the steps themselves are often quickly forgotten. Within our system, this information is specifically collected and stored in a database, so that users can trace back the decision-making process.

5.2 OrionBench – Periodic Benchmarking System for Unsupervised Time Series Anomaly Detection

Benchmarks are essential for fairly, rigorously, and reproducibly comparing the performances of models as they emerge. The lack of a periodic standardized benchmark for unsupervised time series anomaly detection has made this field more scattered. When *end users* – defined here as people who are interested in training a model on their own data in order to find anomalies – attempt to use these models, they regularly run into certain challenges and pain points, which we highlight below.

- There are numerous generative modeling techniques, with each new model promising better performance than all the previous ones. An end user worries that the model they have been using is suboptimal and needs to updated.

- Published work has a lot of complex, machine learning-specific jargon, which makes it difficult for an end user to effectively compare methods. Important components (e.g. pre-processing functionalities) are hidden behind the complexity of the model, when in reality these components are what made the model successful.

- Given these challenges, end users may spend substantial time trying to adapt a new

model's code for their data, only to discover that the new model does not outperform the previous one in their specific situation.

With OrionBench, we aim to address these pain points. How can we support end users in confidently deciding whether or not to adopt a new model? How can we best represent models with proper abstractions, such that new models can be represented as a set of components, and differences between models can be easily identified? How can we provide end users with ready-to-use models, should they choose to adopt them?

5.2.1 Overview

OrionBench is a benchmark suite within the Orion system. A researcher creates a new model and integrates it with Orion through a pull request. A benchmark run is executed and produces a leaderboard, and the model is then stored in the sandbox. This part of the workflow satisfies the goals of the researcher, who aims to compare the performances of different models. To serve end users, pipelines in the sandbox are tested by an Orion developer. Pipelines that pass the tests are verified and become available to end users. This workflow is depicted in Figure 5-7. Five main properties enable our framework for benchmarking unsupervised time series anomaly detection models: model abstractions (Section 5.2.2); hyperparameter standardization (Section 5.2.3); extensions to add new pipelines and datasets (Section 5.2.4); verification of pipelines (Section 5.2.5); and continuous benchmark releases (Section 5.2.6).

5.2.2 Abstracting Models into Primitives and Pipelines

The universal representation of *primitives* and *pipelines* detailed in Section 5.1.2 allows us to include any model in OrionBench. As portrayed in Figure 5-2, we use the `fit` method to train the model and the `detect` method to run inference. With this standardization, we are able to treat all models equivalently. Moreover, this allows researchers to conduct ablation studies in order to attribute pipeline performance to the contribution of specific primitives.

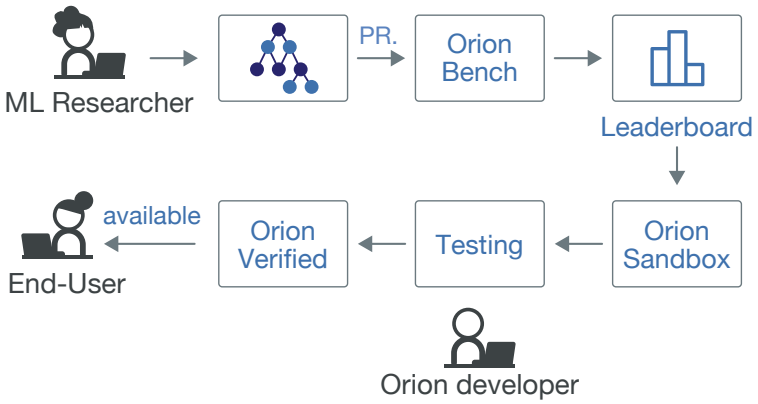


Figure 5-7: OrionBench integrates new models made by ML researchers and compares their performance to currently available models through the leaderboard. After the validity and reproduciblity of the new model is tested, it is transferred from "sandbox" to "verified" and becomes readily available to the end user.

5.2.3 Standardizing Hyperparameter Settings

Deep learning models require setting a multitude of hyperparameters, some of which are model-specific. This has made it more challenging to keep benchmarks fair and transparent. In OrionBench, hyperparameters are stored as `json` files to expose configurations in both machine- and human- readable representations. Figure 5-3 is an example of the hyperparameter settings for `AER`.

To increase benchmark fairness, we standardize both *global* and *local* hyperparameters. Global hyperparameters are shared between pipelines, and typically pertain to pre- and post-processing primitives. For example, in Figure 5-3, `interval` is a global hyperparameter that denotes the aggregation level for the signal – here it is set to 6 hours of aggregation (21,600 seconds). Such hyperparameters are selected based on the characteristics of the dataset, and in some cases are dynamic. For example, `window_size_perc` sets the window size to 30% of the signal length. Local hyperparameters, such as `epochs`, are pipeline-specific and are selected based on the authors' recommendation in the original paper. These hyperparameters are consistent across datasets per pipeline in order to alleviate any bias introduced by knowing the ground truth anomalies of the dataset.

5.2.4 Integrating New Pipelines and Datasets

A main pillar of open-source development is continuously maintaining and updating a library. Benchmark libraries are no different. For a library to grow, it is essential to keep introducing new pipelines and datasets to benefit the end user.

ML researchers build new primitives and compose new pipelines easily in OrionBench. The framework provides templates that guide researchers through this process. Moreover, ML researchers can utilize primitives in other packages given a corresponding `json` representation. Pre- and post- processing primitives are often reusable across pipelines. OrionBench started out with 2 pipelines, and now has 12. The same applies to benchmark datasets. To make the data more accessible, we host publicly available datasets on an Amazon S3 instance. Signals can be loaded via a `load_signal` command (as shown in Figure 5-2) that will directly connect to S3 if the data is hosted there. Otherwise, the system will search for the file locally. This also enables users to load their own private, custom data for benchmarks.

5.2.5 Verifying Pipelines

We organize pipelines into *verified* pipelines and *sandbox* pipelines. When a new pipeline is proposed, it is categorized under “sandbox" until several tests and validations have been done. The ML researcher opens a new pull request, and is asked to pass unit and integration tests before the pipeline is merged and stored in the sandbox. Next, Orion developers test the new pipeline and verify its performance and reproducibility. One of the most commonly encountered situations is a mismatch between the researchers’ comparison report and the results an Orion developer would get from running the same framework. A common reason for this was that researchers had failed to update a hyperparameter setting. Once these checks are made, pipelines are transferred from “sandbox” to “verified”. The increased reliability of verified pipelines enhances the end user’s confidence in adopting pipelines.

5.2.6 Releasing Regularly

The last requirement for an end user-friendly benchmarking framework like OrionBench is to keep track of how benchmark results change over time. Most pipelines are stochastic in

nature, meaning benchmark results can change from run to run. Moreover, when the underlying dependency packages (e.g. `TensorFlow`) introduce new versions, benchmark results can be affected or even compromised. Therefore, it is crucial to monitor pipeline performances over time, and prevent possible breakdowns due to backwards incompatibility.

This is a main driver behind the creation of *OrionBench*. Benchmarking was introduced as a measure of *stability* and *reproducibility* testing, analogous to how Continuous Integration Continuous Deployment (CI/CD) tests have greatly increased the reliability of open-source libraries. OrionBench now serves as a test of pipeline stability over time. As of now, 19 releases have been published, and the *leaderboard* changes with each release.

5.2.7 Benefiting the End User

OrionBench is available to the end user on pypi, where they can install OrionBench through "pip install orion-ml." They then have all verified pipelines at their fingertips, and can train a pipeline using the `fit` API and detect anomalies using the detect API. End users have access to a collection of models that they trust to perform as expected, that fit their computational needs, and that are continuously maintained and benchmarked.

5.3 Evaluation

Datasets. Currently, the benchmark is executed on 14 datasets with ground truth anomalies. These datasets were introduced in Section 3.1.

Pipelines. OrionBench includes 12 pipelines: `ARIMA` – Autoregressive Integrated Moving Average statistical model [Box and Pierce, 1970]; `MP` – Discord discovery through Matrix Profiling [Yeh et al., 2016]; `AER` – AutoEncoder with Regression deep learning model with reconstruction and prediction errors [Wong et al., 2022]; `LSTM-DT` – LSTM non-parametric Dynamic Threshold with two LSTM layers [Hundman et al., 2018]; `TadGAN` – Time series Anomaly Detection using Generative Adversarial Networks [Geiger et al., 2020]; `LSTM VAE` – Variational AutoEncoder with LSTM layers [Park et al., 2018]; `LSTM AE` – AutoEncoder with LSTM layers [Malhotra et al., 2016]; `Dense AE` – Similar to `LSTM AE`, with Dense

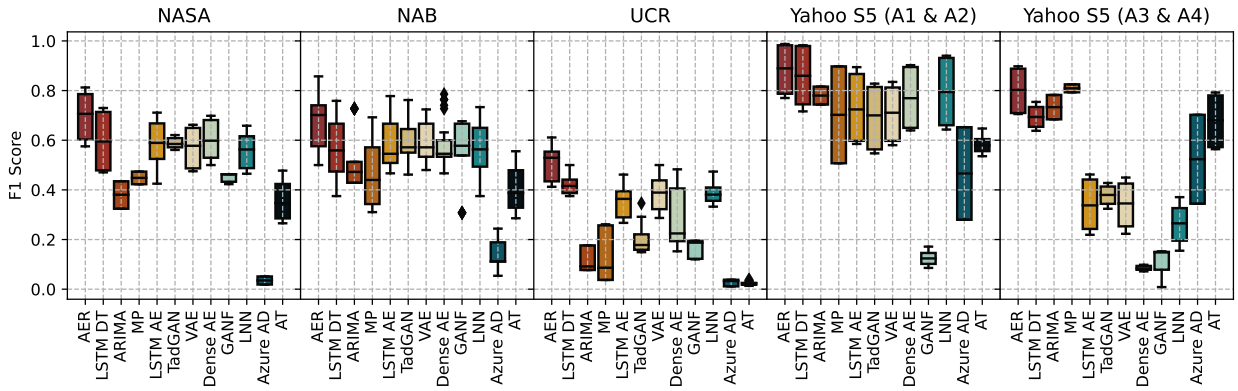


Figure 5-8: Distribution of F_1 Scores across NASA, NAB, UCR, and Yahoo S5. Yahoo S5 was split into two subsets, highlighting the difference in F_1 scores that pipelines experience when detecting point anomalies.

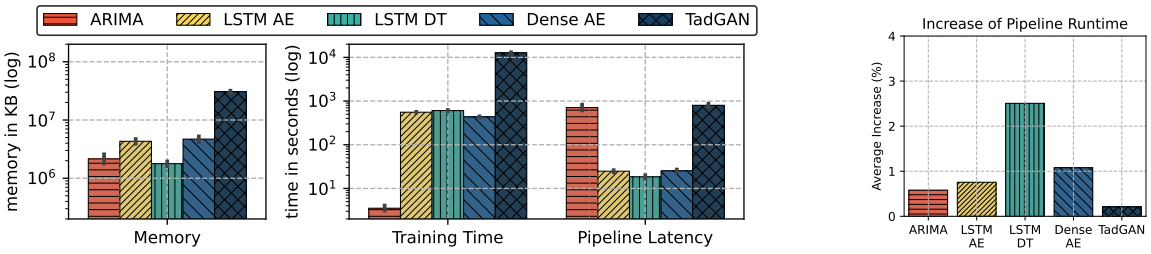
layers [Sakurada and Yairi, 2014]; `LNN` – Liquid Neural Network model, a variant of Liquid Time-Constant Networks [Hasani et al., 2021]; `GANF` – Graph Augmented Normalizing Flows density-based model [Dai and Chen, 2022]; `AT` – AnomalyTransformer model with association discrepancy [Xu et al., 2022b]; `Azure AD` – Microsoft Azure Anomaly Detection service [Ren et al., 2019].

Settings. We use Orion version `0.5.2`. The benchmark is executed to run for 5 iterations over all pipelines and datasets.

Compute. We set up an instance on the MIT SuperCloud [Reuther et al., 2018] with an Intel Xeon Gold 6249 processor of 10 CPU cores (9 GB RAM per core) and one NVIDIA Volta V100 GPU.

5.3.1 Experiment Results

To evaluate our systems, we introduce several experiments that demonstrate the use, performance, and effectiveness of Orion. Moreover, we demonstrate the use of OrionBench on 12 pipelines, ranging from classic to generative models, and 14 datasets. We also lay out how benchmarking works as a mechanism to test pipeline stability.



(a) (b)

Figure 5-9: (a) Pipeline computational performance. (b) Difference in runtime between standalone primitives and end-to-end pipelines.

Qualitative Performance. Figure 5-8 depicts the F_1 score obtained for each dataset, on average. The score achieved by each pipeline differs based on the dataset and its properties. We can see that `AER` is the highest-performing pipeline overall. Another interesting observation is that `LSTM AE`, `TadGAN`, `VAE`, and `Dense AE` are not effective at detecting point anomalies. These pipelines are all reconstruction-based and are susceptible to anomalous regions when computing the deviation between the original and reconstructed signal, producing anomaly scores with reduced peaks at these points. Anomalies thus pass by undetected [Wong et al., 2022]. This is clearly demonstrated in the Yahoo S5 datasets, where F_1 scores for A3 & A4 datasets are low compared to those for A1 & A2. Furthermore, the `Azure AD` pipeline frequently flags segments as anomalous. This strategy works for datasets with a lot of anomalies, such as Yahoo S5. We therefore notice an increased F_1 score there compared to other datasets. Full F_1 scores are shown in Table C.3 in the Appendix.

Computational Performance. We test a subset of pipelines to evaluate the computational performance of pipelines in Orion. We focus on five pipelines: `ARIMA`, `LSTM AE`, `LSTM DT`, `Dense AE`, and `TadGAN`. Figure 5-9a shows the *training time* (the time necessary to train the pipeline end-to-end); the *pipeline latency* (the time it takes the pipeline to produce an output while in `detect` mode); and the *memory* usage necessary for benchmarking NASA, Yahoo, and NAB signals for each of the pipelines presented. We note that the `TadGAN`, `LSTM AE`, and `Dense AE` pipelines require the most memory due to their reconstructive natures. `TadGAN` takes the longest amount of time to train and produce outputs, likely due to its architecture: It is a GAN structure with four interleaved neural networks being trained si-

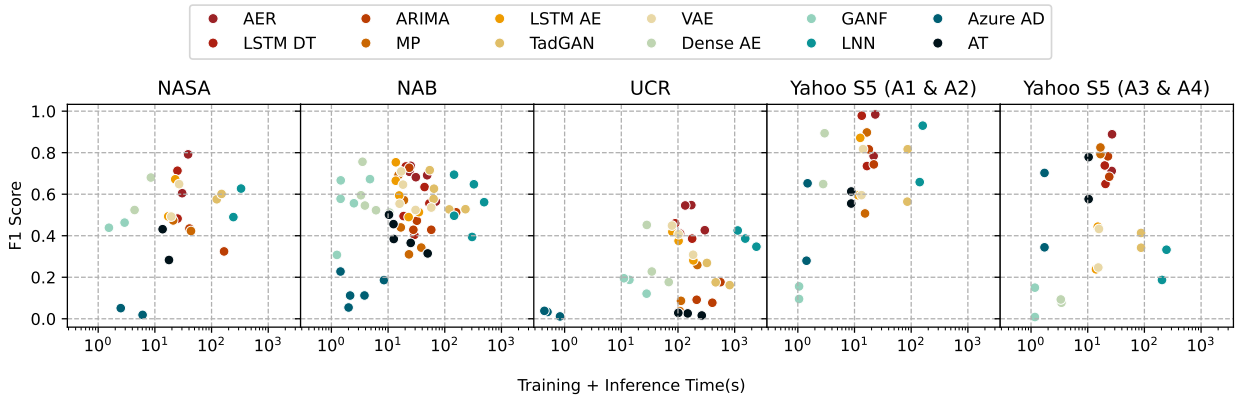


Figure 5-10: Average elapsed time of pipelines across dataset groups with their respective average F_1 scores.

multaneously. `ARIMA`— a popular statistical model — requires a similar amount of time as deep learning pipelines once both training time and pipeline latency have been factored in.

Primitive Profiling. We evaluate the extra computational cost of using pipelines in our framework. We first compute the total runtime required for each primitive to run in an external setting (outside of our framework). Next, we compare this to the time needed to run a pipeline from beginning to end. We determine the runtime of each model on the entire dataset. We compute the delta as the difference between using a pipeline and running the primitives independently. Although running primitives independently is faster than running the same primitives as part of a pipeline counterpart, the delta is generally minimal ($\mu \pm \sigma, \%$ avg. inc. time): `ARIMA` ($4.5 \pm 5.4s$, 0.58%), `LSTM AE` ($12.8 \pm 32.4s$, 0.75%), `LSTM DT` ($15.6 \pm 17.6s$, 2.5%), `Dense AE` ($17.8 \pm 44.4s$, 1.0%), and `TadGAN` ($28.7 \pm 46.4s$, 0.2%). Figure 5-9b illustrates the average percentage increase that comes from running primitives in our pipeline versus independently. Given their stochastic nature, deep learning models tend to be more volatile from one signal to another, leading to higher runtime variability.

Performance Trade-Off. In addition to quality performance, end users are interested in the tradeoff between pipelines' computational time and performance. Figure 5-10 illustrates how much time (in seconds) on average each pipeline needs, as well as its score. Elapsed time includes the time it takes to train a pipeline and the time it takes to run inference. Pipelines that are on the bottom right of the plot show ineffectiveness in performance while

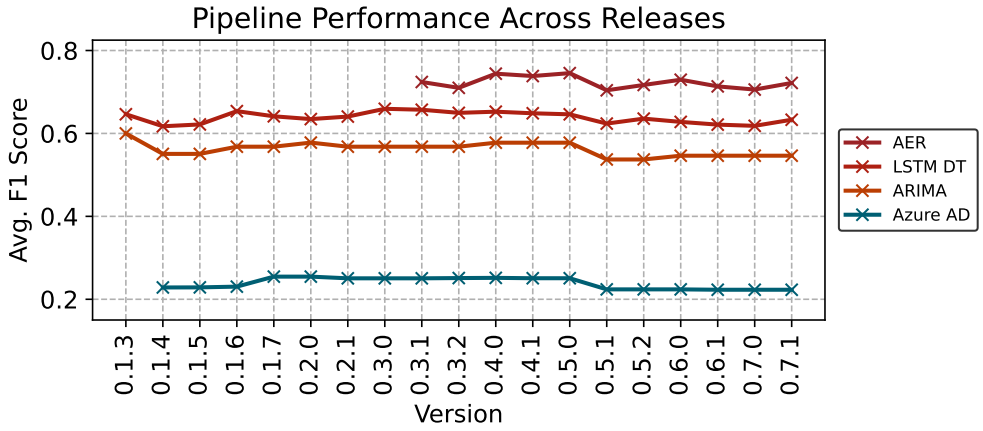


Figure 5-11: Monitoring pipelines' performance across releases.

also being computationally expensive. Moreover, each dataset has a varying performance evaluation. For example, in NAB, we can see that `GANF` is a well-performing model and computationally inexpensive, similar to `Dense AE`. However, their efficiency is fruitless when evaluating on A3 & A4, given their sub-par performance. In general, `LNN` and `TadGAN` are the most time-consuming pipelines given their internal model complexity. Overall, we can see that `AER` and `LSTM DT` are performance and computationally comparable across all datasets.

Moreover, a user might sacrifice quality performance for computational efficiency, or vice versa. Individual end users can make their own decisions when weighing these tradeoffs.

5.3.2 Stability.

As we see in Figure 5-8, `AER` is the highest-performing pipeline: *Was this always the case?* OrionBench publishes benchmark results with every package release.

Figure 5-11 depicts the average F_1 score of four pipelines. These pipelines are chosen to represent the best-performing pipeline (`AER`), the worst-performing pipeline (`Azure AD`), the first implemented pipeline (`LSTM DT`), and the most classic pipeline (`ARIMA`) currently available in our framework. The observed performance can change from one release to another for a number of reasons, including the stochastic nature of pipelines, internal changes in dependency packages, and dynamic thresholding. If we look closely at Figure 5-11, we notice three shifts (viewed as slopes) to `LSTM DT`, and only two to `AER`, `ARIMA`, and `Azure AD`.

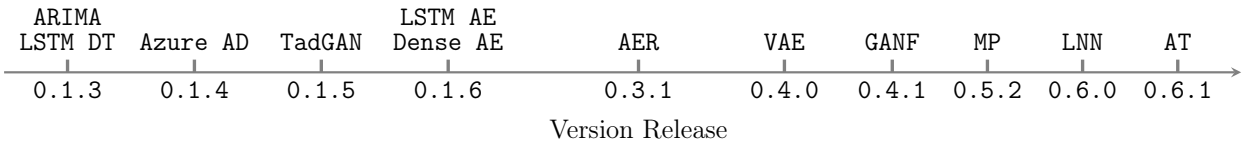


Figure 5-12: Timeline of pipeline introductions to OrionBench. In 2020, we started with 2 pipelines; over the course of four years, we introduced 10 more pipelines at different stages.

First, in version update `0.1.3` → `0.1.4`, we saw a drop in F_1 score due to an internal change in how we calculate the overall scores. The aggregation calculation became automated, and was conducted on the dataset level rather than the signal level. Second, in version update `0.1.5` → `0.1.6`, there was an increase in performance that can be traced back to our hyperparameter setting modifications. Third, going from version `0.3.2` → `0.4.0` shifted our implementation from `TensorFlow` version 1 to 2, which impacted the underlying implementation. Lastly, after introducing a new dataset, namely UCR, we noticed a drop in the overall performance by pipelines in `0.5.0` → `0.5.1` because this was a more difficult dataset. For developers, running the benchmark every release reassures that there has been no performance disruption. Moreover, it stabilizes pipelines and makes sure that they do not go out of date, especially with dependency package updates. Overall, the observed changes were minimal and could all be traced back to framework alterations.

5.3.3 Pipeline Integration.

Figure 5-12 showcases exactly when each pipeline was integrated into OrionBench. The first benchmark release, version `0.1.3`, in September 2020, featured only 2 pipelines. Over time, new models have been developed and integrated. As of today, OrionBench has 12 verified pipelines, ranging from classical models to deep learning models and made by 5 different contributors.

5.3.4 OrionBench in Action

As anomaly detection models continue to be developed, OrionBench allows researchers and end users to understand and compare these models. In this subsection, we walk through two real-world scenarios, where benchmarking was useful for: (1) guiding researchers to develop

a new model for unsupervised time series anomaly detection; (2) providing end users with an existing state-of-the-art model. We show that OrionBench is a commodity benchmarking framework.

We first describe the state of OrionBench, where `LSTM DT` [Hundman et al., 2018] and `TadGAN` [Geiger et al., 2020] (which was developed by the Orion team) performed competitively against each other until version `0.3.1`, when `AER` [Wong et al., 2022] was introduced. Below, we illustrate the story behind the `AER` model and how we, the OrionBench developers, helped benchmark this model.

Scenario 1 – OrionBench guided a researcher to focus in the right direction. Researchers are eager to adopt the latest innovations in deep learning. An independent researcher was keen on introducing the attention mechanism to anomaly detection [Vaswani et al., 2017]. While the model was promising in local experiments, to assure its performance, we decided to run it through OrionBench. Unfortunately, the model could not improve on either `LSTM DT` or `TadGAN`. This reoriented the project and led to an investigation of the successes and limitations of pipelines. Subsequently, it led to a deep understanding of where prediction models prevailed compared to reconstruction models and vice versa. OrionBench helped guide this process by cross-referencing model performance with dataset properties. The conclusion was that prediction-based anomaly scores are better at capturing point anomalies than reconstruction-based anomaly scores. Moreover, reconstruction-based anomaly scores are better at capturing longer anomalies. Wong et al. [2022] uncovered more associations related to anomaly scores and error methods. The outcome of this investigation ultimately resulted in the `AER` model, which is now the best-performing pipeline on OrionBench.

Scenario 2 – OrionBench helped an end user add a new model, and provided an them with confidence in other models. We had been working with an end user from a renowned satellite company for over four years when they approached us with interest in a new SOTA model. The model, `GANF`, [Dai and Chen, 2022] had caught their attention after being featured in a news article [3]. New models are frequently published that claim

[3] https://news.mit.edu/2022/artificial-intelligence-anomalies-data-0225

SOTA performance by beating existing models on their benchmarks. The end user wanted to know: *Should we adopt this new model?* Several issues can prevent such models from living up to their promised performance in industrial and operational settings. Real-world datasets are inherently more complex than pristine benchmark datasets. Furthermore, authors often fine-tune a model to the benchmark datasets, neglecting others and causing their model to underperform on unseen datasets. OrionBench, as an independent benchmark, can help determine whether it makes sense to adopt a new model. We integrated `GANF` into OrionBench. As presented earlier in Figure 5-8, it was only competitive on the NAB dataset. However, due to the seamless integration of the pipelines into OrionBench, the end user was still able to apply the pipeline to their own data and obtained valuable results. This emphasizes that the behavior of models differs from one dataset to another, and there is no one-pipeline-fits-all.

Similarly, `LNN` models [Hasani et al., 2021] have been utilized in a variety of applications, including robot control. A published news article[4] suggests that these models can perform any time series task. To test their ability to perform unsupervised anomaly detection, we implemented an `LTC` primitive and, shortly after, the `LNN` pipeline. Hasani et al. [2021] released an accompanying `pip` installable library, which has made creating the `LNN` pipeline straightforward. It took one week from its first commit to when it merged on the main branch and became sandbox-available. OrionBench has made it easier for us to incorporate new models and assess their anomaly detection capabilities.

5.4 Conclusion

In this chapter, we introduce two systems: Orion and OrionBench. Orion is an end-to-end framework for unsupervised time series anomaly detection. Through its abstractions of primitives and pipelines, Orion enables users to seamlessly access a diverse set of heterogeneous models via standardized APIs. Complementing this framework, OrionBench is a continuous benchmark that currently includes 45 primitives, 12 pipelines, and 14 publicly available datasets. We evaluate both the qualitative and computational performance of the pipelines

[4]https://news.mit.edu/2021/machine-learning-adapts-0128

across all datasets and further present the results accumulated by OrionBench since 2020. These results underscore the benchmark's role in enabling continuous evaluation, while also demonstrating the extensibility and stability of the pipelines over time.

Chapter 6

Unlocking a New Paradigm with Foundation Models

Recent work has shown the extraordinary ability of pretrained models to perform a wide variety of tasks in zero-shot conditions, i.e. without fine-tuning [Radford et al., 2019, Sanh et al., 2022]. In this chapter, we investigate the utilization of foundation models, primarily Large Language Models (LLMs) and Time Series Foundation Models (TSFM), in performing unsupervised time series anomaly detection.

Chapter outline:

- We look closely at LLMs and converting time series into textual input for time series anomaly detection in Section 6.1
- Section 6.2 investigates the new paradigm of TSFM in performing time series anomaly detection.

6.1 Large Language Models as Anomaly Detectors

Large Language Models (LLMs) have demonstrated an outstanding ability to learn natural language tasks implicitly, whether through performing reading comprehension, text summarization, translation, or related tasks [Radford et al., 2019, Brown et al., 2020, Sanh et al., 2022, Wei et al., 2022, Chowdhery et al., 2023]. Moreover, LLMs have shown tremendous

promise for formal language generation, including code generation and synthesis [Austin et al., 2021, Chen et al., 2021, Xu et al., 2022a], and in production beyond textual output, such as generating images and videos from natural language descriptions [Saharia et al., 2022, Koh et al., 2023]. Testing these models on new tasks and data modalities allows us to push the boundaries of LLMs and clarify their capabilities. In this thesis, we investigate the question "*can LLMs become anomaly detectors for time series data?*". Here, LLMs are exposed to a new data type – time series – and are tasked with a detection task, which is different from the classification tasks at which they are known to excel [Howard and Ruder, 2018].

In this section, we will present two methodologies:

1. **Prompter**, a simple and direct prompting method that asks LLMs to identify the parts of a sequence it thinks are anomalous.

2. **Detector**, which leverages LLMs' ability to forecast time series to find anomalies by using the residual between the original signal and the forecasted one.

Before we detail our methods, we describe our *timeseries-to-text* representation component, which converts time series data into LLM-ready input.

6.1.1 Time Series Representation

LLMs process sequential data in textual form that are later converted into tokens. For this section, we will focus on a simpler case of univariate time series $\mathcal{X} = \{x_1, x_2, \ldots, x_T\}$, where $x_t \in \mathbb{R}$ 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 $\mathcal{X}$ into a sequence of text values that can be tokenized. We follow a sequence of reversible steps – up to a certain degree – that we detail below.

For the remainder of this subsection, we will consider the following sequence as an example:

$$0.2437, 0.3087, 0.002, 0.004, 0.462$$

Scaling. Time series data includes values of varying numerical magnitudes, and may include both positive and negative values. To standardize 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 $\mathcal{X}_s = \{x_{s_1}, x_{s_2}, \ldots, x_{s_T}\}$, where $x_{s_t} \in \mathbb{R}_{\geq 0}$. In other words, we have introduced a mapping function: $\mathcal{S} : \mathbb{R} \to \mathbb{R}_{\geq 0}$.

$$0.2437, 0.3087, 0.002, 0.004, 0.462 \to 0.2435, 0.3085, 0, 0.002, 0.460$$

Other scaling methods, such as min-max or logarithmic scaling, can be used to achieve the same goal. However, with min-max scaling, reducing the set of possible values to a smaller range (e.g. [0, 1]), may cause information loss in the quantization step, while increasing the range will mean there are more digits to tokenize later. On the other hand, logarithmic scaling can stretch and compress the space between values. 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 a 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 [Ansari et al., 2024, Gruver et al., 2023]. We use the rounding method, as proposed in Gruver et al. [2023]. 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 $\mathcal{X}_q = \{x_{q_1}, x_{q_2}, \ldots, x_{q_T}\}$, where $x_{q_t} \in \mathbb{Z}_{\geq 0}$. Below is an example of this operation where we retain 3 decimal points:

$$0.2435, 0.3085, 0, 0.002, 0.460 \to 244, 309, 0, 2, 46$$

Overall, we use 2 mapping functions: the scaling function noted $\mathcal{S} : \mathbb{R} \to \mathbb{R}_{\geq 0}$ and the

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

Model Tokenizer	no space	with space
`gpt`	244 , 309 , 0 , 2 , 46	2 4 4 , 3 0 9 , 0 , 2 , 4 6
`mistral`	2 4 4 , 3 0 9 , 0 , 2 , 4 6	2 4 4 , 3 0 9 , 0 , 2 , 4 6

Table 6.1: How different model tokenizers tokenize a squence of numbers. Inserting space can help enforce per-digit tokenization.

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

Rolling windows. LLMs are bounded by an upper limit on the context length (e.g., `mistral` has a limit of 32k tokens and `gpt-3.5-turbo` has a limit of 16k tokens for both input and output). Moreover, computational constraints on GPU memory make it infeasible to process an entire time series at once. Therefore, we propose a rolling windows technique to manage input data that exceeds these boundaries. This method involves segmenting each time series into rolling windows characterized by predetermined window size w and step size s; i.e., a processed time series $\mathcal{X}_q$ is segmented and turned into a set $\mathcal{X}_{windows} = \{\left(x^{(i)}_{q1\ldots w}\right)\}_{i=1}^{N}$, where N is the number of windows. For a cleaner notation, we refer to the set as $\{\left(x^{(i)}_{1\ldots w}\right)\}_{i=1}^{N}$. We drop q in the notation from this point on, as all the input is now quantized.

Casting. We convert each number from an integer into a string representation with a predetermined separator value. By default, we choose the comma separator "," since it is a commonly used separator for numeric sequences.

$$244, 309, 0, 2, 46 \rightarrow \text{“244,309,0,2,46”}$$

Tokenization. Different tokenization schemes vary in how they treat numerical values. Several open-source LLMs, such as `llama` [Touvron et al., 2023] and `mistral` [Jiang et al., 2023], utilize the byte-level Byte-Pair-Encoding (BPE) tokenization algorithm [Sennrich et al., 2016, Touvron et al., 2023] using the implementation provided by SentencePiece [Kudo and Richardson, 2018], which segments numbers into individual digits. On the other hand, `gpt` uses a `cl100k_base` tokenizer, which tends to segment numbers into chunks that may not correspond directly with the individual digits [Liu and Low, 2023]. For instance, the number

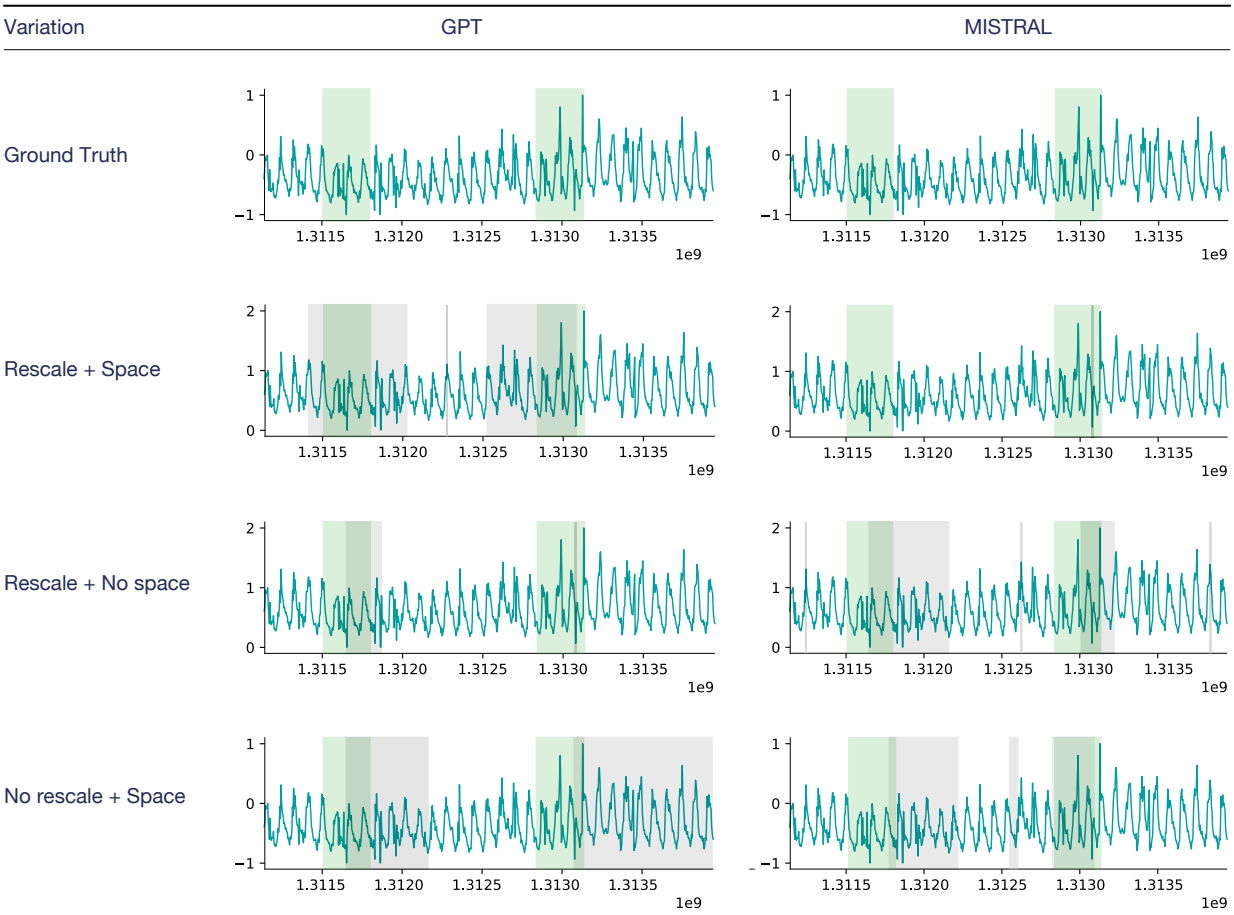


Figure 6-1: Visualizing the output of large language models (`gpt` and `mistral`) under different variations of the transformation process. Each row depicts the exchange-2_cpm_results signal from the AdEx dataset. The first row indicates the ground truth anomalies present in the time series (highlighted 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 using the prompter method. Overall we find that "scaling + space" is the configuration that yields a better output for `gpt`; and "scaling + no space" is better for `mistral`.

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 [Gruver et al., 2023]. To make sure `gpt` tokenizes each digit separately, we insert whitespace between the digits themselves and separators. Continuing with the running example:

"244,309,0,2,46" → "2 4 4 , 3 0 9 , 0 , 2 , 4 6"

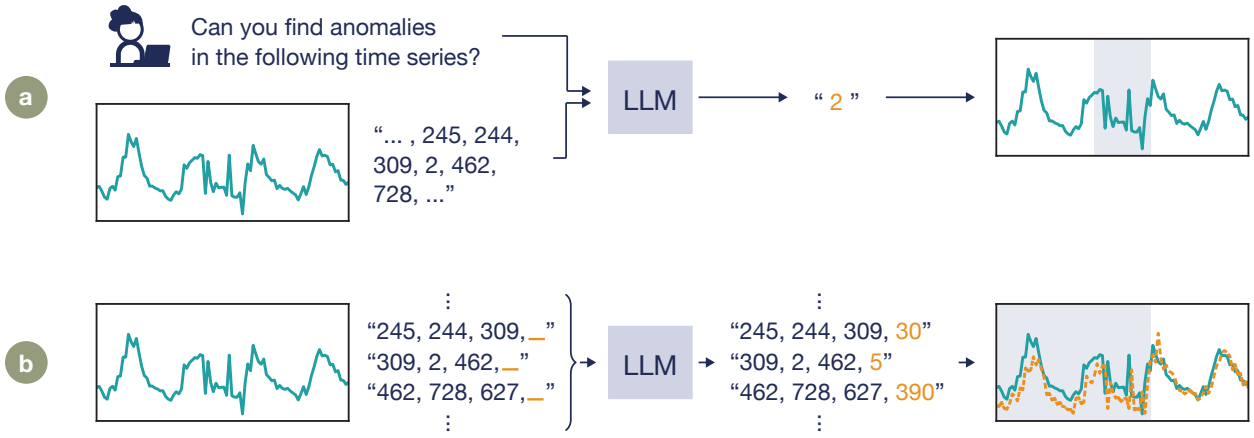


Figure 6-2: Anomaly detection methods using LLMs. (a) **Prompter**: a prompt engineering approach, directing large language models to identify the parts of the input that are anomalies. (b) **Detector**: a forecasting approach, using large language models as forecasting methods. **Detector** then finds discrepencies between the original and forecasted signal, which indicate the presence of anomalies.

where each digit is now encoded separately. We highlight the difference in tokenization achieved by inserting space into the sequence in Table 6.1.

Forcing spaces into the input sequence of `gpt` makes the sequence much longer, as now we have at least $2 \times T$ tokens representing a space, one before and one after each digit. However, because we do not have direct access to the model, we cannot post-process the spaces out of the input sequence after tokenization and before prompting the model.

Figure 6-1 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 et al., 2023].

After transforming a univariate time series into its text representation, we introduce two methods for anomaly detection using LLMs. We visualize our methodologies in Figure 6-2.

6.1.2 Prompter: Finding Anomalies through Prompting

As depicted in Figure 6-2, this pipeline involves querying an LLM directly, asking it to find time series anomalies through a text prompt concatenated with the processed time series

window $u^{(i)}_{1\ldots k} \coloneqq \text{prompt} \oplus x^{(i)}_{1\ldots w}$, 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 $p_\theta(u_{k+1}|u_{1\ldots k})$.

Following a series of experiments (shown in Table C.1 in the Appendix) we iterated over trial #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: {$x^{(i)}_{1\ldots w}$}."*

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 anomalous indices using the prompt presented in Table C.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 6.1.1, we adopt a rolling window approach to segment the time series before inputting it into the language model. For each window, we sample the model $M = 10$ times and record the positions (i.e., indices) identified as anomalous in each sample. Therefore, a single timestamp in the data can occur in multiple samples as well as multiple windows. We introduce voting strategies in order to determine if a timestamp should be counted as anomalous or not.

Sample-level voting. An index within a window is considered anomalous if it appears in at least α-percent of the samples. These anomalous indices are then mapped back to their corresponding positions (timestamps) in the original time series.

Overlapping window-level voting. To aggregate the results across overlapping windows, we count how many times each timestamp was flagged as anomalous across all the windows that it appears in. A timestamp is finally labeled as anomalous if it was flagged in at least β-percent of its windows.

The hyperparameters $\alpha \in [0, 1]$ and $\beta \in [0, 1]$ control the strictness of anomaly detection at the sample level and across overlapping windows, respectively. We can view α and

β as confidence hyperparameters, where if the LLM is certain about a timestamp being anomalous, then it should be present in most samples and windows.

6.1.3 Detector: Finding Anomalies through Forecasting

As depicted in Figure 2-3, the first step in a typical ML pipeline involves training an ML model on a collection of historical time series. However, recent work demonstrates that pretrained LLMs are capable of zero-shot forecasting, allowing us to skip the training phase and go directly to inference [Gruver et al., 2023].

Pre-processing. As detailed in Section 6.1.1, our first step involves transforming a raw input into a textual representation, and creating samples ready for the LLM from the rolling window sequences $\mathcal{X}_{windows} = \{\left(x^{(i)}_{1\ldots w}\right)\}_{i=1}^{N}$ where $N = \lceil (T-w)/s \rceil$.

Forecasting. For each given window $x^{(i)}_{1\ldots w}$, we aim to predict the next values $x^{(i)}_{w+1\ldots w+h}$ where h is the forecast horizon. For ease of notation, the predicted sequence for a window i is noted as $x^{(i)}_{h}$, and the lack of i indication means this is applied for all windows. This can be achieved through the next token conditional probability distribution noted $\hat{x}_h = p_\theta(x_h \,|\, x_{1\ldots w}$ and $x \in \mathbb{Z}_{\geq 0})$. With this approach, we give the model the input window $x_{1\ldots w}$, and sample multiple sequences from the distribution to estimate $x_h \approx \mathcal{E}^{-1}(\mathcal{G}^{-1}(\hat{x}_h))$. This yields multiple overlapping sequences $\{\left(\hat{x}^{(i)}_{h}\right)\}_{i=1}^{N}$ at each point in time, depending on h and step size s. Concretely, the overlap would occur $\lceil h/s \rceil$ times.

Post-processing. The post-processing phase contains multiple steps. The first step transforms the forecasted time series into a univariate time series. The second step finds the discrepancies between the original and forecasted series.

Time series aggregation. For each time point t, we now have multiple forecasted values in different windows, which we refer to as W_t. For example, when $s = 1$, we will have $W_t = \{\hat{x}^{(i)}_{j}, i + j = t\}$. Furthermore, to increase the reliability of the prediction, we take M samples from the distribution for each window. Therefore, each $\hat{x}_t$ has M samples.

$$S_t = \{\hat{x}^{(i,m)}_{t} \,|\, i \in W_t,\ m = 1, \ldots, M\}$$

To map this back to a univariate time series, we explore the results by taking the mean,

median, 5^{th}-percentile, and 95^{th}-percentile as values. For the purpose of anomaly detection, an extreme forecast value could indicate a precursor to an anomaly; therefore, acute values can be informative.

$$\tilde{x}_t = \text{agg}\left(S_t\right)$$

Now, we have reconstructed the time series as $\tilde{\mathcal{X}} = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_T\}$.

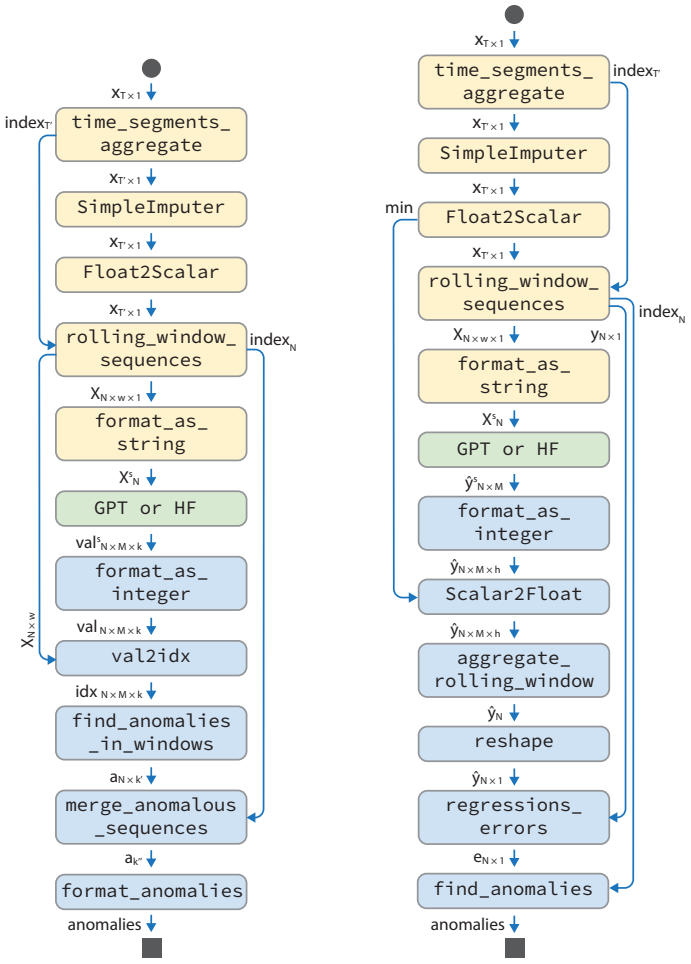
We next compute the discrepancy between $\mathcal{X}$ and $\tilde{\mathcal{X}}$. A large discrepancy indicates the presence of an anomaly. We denote this discrepancy as an error vector $\boldsymbol{e}$ by computing point-wise residuals, given their simplicity and ease of interpretation. We explore the use of absolute difference and squared difference as suggested by Hundman et al. [2018]. More complex functions that capture the difference between two signals, such as dynamic time warping [Müller, 2007] can be used. However, Geiger et al. [2020] showed that discrepancies found with point-wise errors are sufficient for this purpose. Moreover, we apply an exponentially weighted moving average to reduce the sensitivity of the detection algorithm [Hundman et al., 2018]. We use a sliding window approach to compute the threshold to help reveal contextual anomalies that are abnormal compared to their local neighborhoods.

6.1.4 SigLLM – System for Unsupervised Time Series Anomaly Detection using Large Language Models

Working with large language models requires additional considerations in order to handle the multi-modality of time series and textual data. Therefore, we build SigLLM on top of Orion.

Machine Learning Stack. During SigLLM's development, we questioned whether Orion's abstractions of primitives and pipelines would translate to this new setting. We have since confirmed that they do: the abstractions map cleanly and let us integrate both service-based LLMs, like OpenAI's, and libraries beyond Orion's initial scope, such as HuggingFace transformers library.

In addition to the Orion primitives, SigLLM requires specific primitives for representing time series as textual data (described in Section 6.1.1). Below, we summarize these primitives:



(a) Prompter (b) Detector

Figure 6-3: Directed acyclic graphs (DAGs) of SigLLM pipelines. (a) Prompter pipeline; (b) Detector pipeline.

- `Float2Scalar`: converting floating values into scalars. This primitive features the scaling and quantization operations.
- `format_as_string`: casting scalars as string.
- `format_as_integer`: verifying string output to include only scalar values, then casting it as integer.
- `Scalar2Float`: converting scalar values back into floating values. This step reverses the quantization operation and rescales the data.

Figure 6-3 refers to SigLLM pipelines, where sigllm primitives can be either pre-processing

or post-processing primitives. The `gpt` primitives connects to OpenAI's APIs, while `HF` primitive loads any model locally.

Supported Models. As of writing this thesis, we have included support for any language model accessible through the Hugging Face transformer library [2]. Moreover, we provide support for `gpt` through OpenAI APIs [3]. Users should export Hugging Face and OpenAI API keys to their environments prior to engaging with SigLLM. For our experiments, we focus on `mistral` and `gpt` models.

Core Interaction. Similarly to Orion, SigLLM proposes the usage of `sigllm.detect` to find anomalies in time series using LLMs. However, since LLMs are pre-trained models, we remove the need for a `.fit` function. Another benefit of SigLLM is that it primarily focuses on LLMs; therefore, we can expose hyperparameters to the SigLLM class for users to modify. We list the most common parameters:

- `interval`: the time gap between one sample and another.
- `decimal`: the number of decimal points to keep from the float representation.
- `window_size`: the size of the input window.

Moreover, we allow few-shot prompting by showing the LLM what a normal sequence would look like. For that, we introduce a new parameter, '`normal`,' to the prompting scheme.

Benchmarking. We build a benchmarking framework for SigLLM that extends the Orion-Bench system. The need for extension stems from the slight alteration in APIs, and the need to support both the prompting and forecasting paradigms. More specifically, the prompting paradigm features a few-shot pipeline, which requires "normal" sequences as an argument.

Creating Normal Sequences. We create a normal sequence for each signal in the benchmarking dataset. The normal sequence is extracted as a single window that falls within a region that is not part of the ground truth anomalies. We develop a `load_normal` function to support this functionality.

[2] https://huggingface.co/docs/transformers
[3] https://platform.openai.com/docs/overview

6.2 Time Series Foundation Models for Anomaly Detection

With the emergence and success of pretrained language models, new work has focused on training transformer models on a large collection of time series data [Rasul et al., 2023, Gao et al., 2024, Das et al., 2024, Ansari et al., 2024]. Most models are trained to predict the next value in a time series, which is a forecasting task. On the other hand, a model such as `UniTS` is a unified multi-task time series model for predictive and generative time series modeling. In this section, we explore the usage of TSFM for unsupervised time series anomaly detection through the paradigm of forecasting.

6.2.1 Task Formulation

Since the models are pre-trained, the objective is to predict the next steps ahead using the model for each window in $\mathcal{X}_{windows}$. In other words, these models are only invoked during the inference stage.

The methodology follows the same steps discussed in Section 6.1.3. For each window $x^{(i)}_{1\ldots w}$, we aim to find $x^{(i)}_{w+1\ldots w+h}$ (or $x^{(i)}_h$ for ease of notation). Let f denote any TSFM model; then, for each window, we predict h steps ahead $x^{(i)}_h = f(x^{(i)}_{1\ldots w})$. Then we apply the same post-processing operations to obtain $\tilde{\mathcal{X}}$ and consequently the error vector $\boldsymbol{e}$.

6.2.2 Time Series Foundation Models

In this thesis, we look at `UniTS` and `TimesFM`. Note that currently, both models only support univariate forecasting. Moreover, to our knowledge, both training corpora use datasets outside of the experimental datasets used in this thesis.

UniTS: Unified Multi-Task Time Series Model

`UniTS` was developed by a group of researchers from Harvard, MIT Lincoln Laboratory, and the University of Virginia [Gao et al., 2024]. Its primary goal is to encompass time series tasks into a unified framework. Their approach uses a modified transformers block

and special tokens (`GEN` for generative, `CLS` for predictive) in order to perform tasks such as time series forecasting, imputation, classification, etc. The model is pre-trained on 39 public datasets that are specific to time series forecasting and classification. Further details of the datasets used are shown in Appendix A.2.1.

The authors provide the code, dataset, and model checkpoints on github [4]. To directly access the model, we retrain the model using the publicly available code and store it on Orion's S3 bucket [5]. This allows users to load models directly from the server without intermediately locally saving the model weights.

TimesFM: Time Series Foundation Model

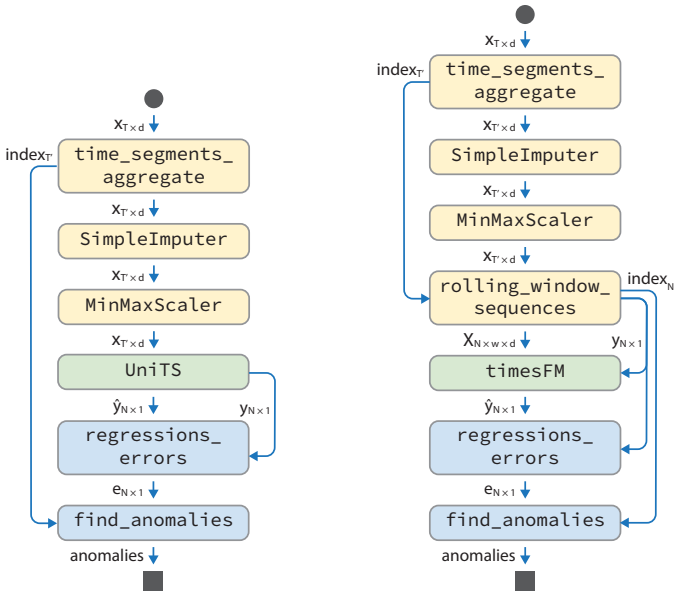
`TimesFM` was developed by Google Research for time series forecasting [Das et al., 2024]. `TimesFM` utilizes data from Google Trends, Wikipedia Pageviews, and other publicly available datasets. Moreover, they synthetically generate training data from an ARMA process while varying seasonality and trend. Overall, their training corpus reaches over 300 billion time points. We provide additional details on the datasets they used in Appendix A.2.2.

The model is publicly available on Hugging Face's `transformers` library [6]. We use the official APIs developed by the team (available on github) in order to load the model and run inference [7].

TSFM in Orion

Since the data modality handled by TSFM is time series, we introduce a new category of models in Orion, namely pretrained pipelines. In contrast to conventional pipelines, these models are independent of our training framework. Both `UniTS` and `TimesFM` have been integrated into the Orion system and are directly available to the end user. Figure 6-4 shows the pipeline representation of these models.

[4] https://github.com/mims-harvard/UniTS/releases/tag/ckpt
[5] https://sintel-orion.s3.amazonaws.com/pretrained/units.pth
[6] https://huggingface.co/google/timesfm-1.0-200m-pytorch
[7] https://github.com/google-research/timesfm



(a) `UniTS` (b) `TimesFM`

Figure 6-4: DAGs of TSFM pipelines. (a) `UniTS` pipeline; (b) `TimesFM` pipeline.

6.3 Results

Datasets. We examine these foundation models 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**; **Yahoo S5** which is based on production traffic to Yahoo systems; and **NAB**, which includes multiple types of time series data from various application domains. We consider five sub-datasets: Art, AWS, AdEx, Traf, and Tweets. The details of these datasets are shown in Section 3.1. In total, these datasets contain 492 univariate time series and 2,349 anomalies. The properties differ between datasets; for instance, the NASA and NAB datasets contain anomalies that are longer than those in Yahoo S5, and the majority of anomalies in Yahoo S5's A3 & A4 datasets are point anomalies.

Models. We used `Mistral-7B-Instruct-v0.2` for **Prompter** and **Detector**. Moreover, we used `gpt-3.5-turbo-instruct` for **Prompter** alone. Due to cost constraints, we explored the usage of `gpt` for **Detector** on a 5% sample of all datasets, which produced similar results to using `mistral`. We use all previous models detailed in Section 4.4 as comparison models.

Hyperparameters. For **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 (i.e, 100 steps for SMAP and MSL, and 40 for others). For **Detector**, we set the window size to 140 and the step size to 1. With a rolling window strategy of step size 1, it is important to keep the windows as small as possible while still ensuring that they are large enough to make useful predictions, as more context tends to be useful for LLMs. Our preliminary results suggested that a window size of 140 produced comparable results to a window size of 200, and was much better than a window size of 100. We set the horizon to 5. For `UniTS`, we set the window size to 250, which is consistent with other prediction-based approaches available in Orion. On the other hand, `TimesFM` requires the window size to be a power of two, so we chose the closest number, 256.

Computation. For `gpt`, we used `gpt-3.5-turbo` due to its superior performance on time series data (demonstrated by Gruver et al. [2023]) and its affordability. For `mistral`, we used the publicly available model hosted by Hugging Face [8]. We used an Intel i9-7920X 24 CPU core processor and 128GB RAM machine with 2 dedicated NVIDIA Titan RTX 24GB GPUs to run **Prompter** and **Detector** on `mistral`. We used the same machine for `UniTS` and `TimesFM`.

6.3.1 Qualitative Evaluation

Following the same set of time series examples shown in Section 4.4, we visualize the output of pretrained models when testing on these time series. Figure 6-5 shows an example of an extreme point occurring at the middle of the observation signal. All pretrained models successfully identify the point anomaly; `UniTS`, however, detects an additional false positive. It is also worthwhile to note that the signal forecasted by `UniTS` exhibits fluctuations due to the inclusion of the anomaly as part of the input.

Figure 6-6 shows an example of a contextual anomaly that is harder to detect. Recall that all deep learning models shown in Section 4.4 successfully identified the anomaly; however, pretrained models show lower precision. For forecasting-based models, only `TimesFM` predicts a time series that mimics the original behavior.

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

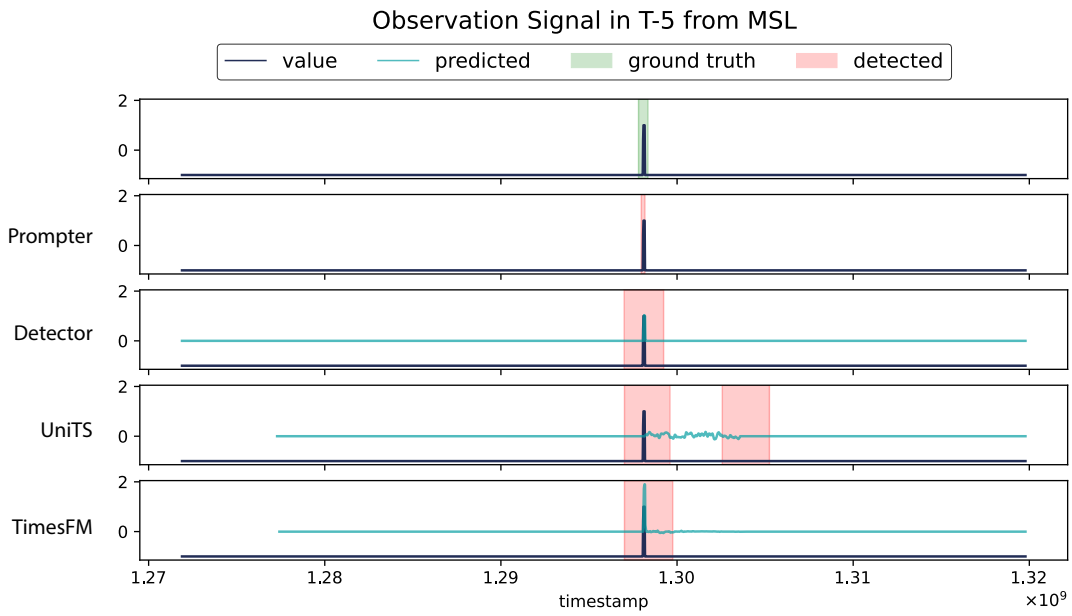


Figure 6-5: Visualizing `T-5` from MSL detections by **Prompter**, **Detector**, `UniTS`, and `TimesFM` where the signal has point anomalies.

	Precision	Recall	F1 Score
Prompter `mistral`	0.219 ± 0.108	0.311 ± 0.213	0.223 ± 0.104
Prompter `gpt`	0.162 ± 0.133	0.245 ± 0.191	0.133 ± 0.076
Detector	0.613 ± 0.184	0.514 ± 0.211	0.525 ± 0.167
`UniTS`	0.602 ± 0.210	0.514 ± 0.265	0.484 ± 0.226
`TimesFM`	0.654 ± 0.215	0.466 ± 0.254	0.472 ± 0.224

Table 6.2: Summary of Precision, Recall, and F_1 Score

6.3.2 Performance Evaluation

Are foundation models effective anomaly detectors? After running the models on the full datasets, we computed the precision, recall, and F_1 scores, shown in Table 6.2. Overall, `mistral` achieved better results than `gpt` for the **Prompter** method, with a 2× better F_1 score. In addition, **Detector** performed better overall than **Prompter**. This demonstrates how direct LLM application can produce erroneous output, where taking the intermediate step of forecasting yields a much better score. Moreover, TSFM performs competitively to each other, with `UniTS` having better recall and `TimesFM` achieving better precision scores. When comparing **Detector** with TSFM, we can see that **Detector** outperforms both `UniTS` and `TimesFM` in terms of overall F_1 score.

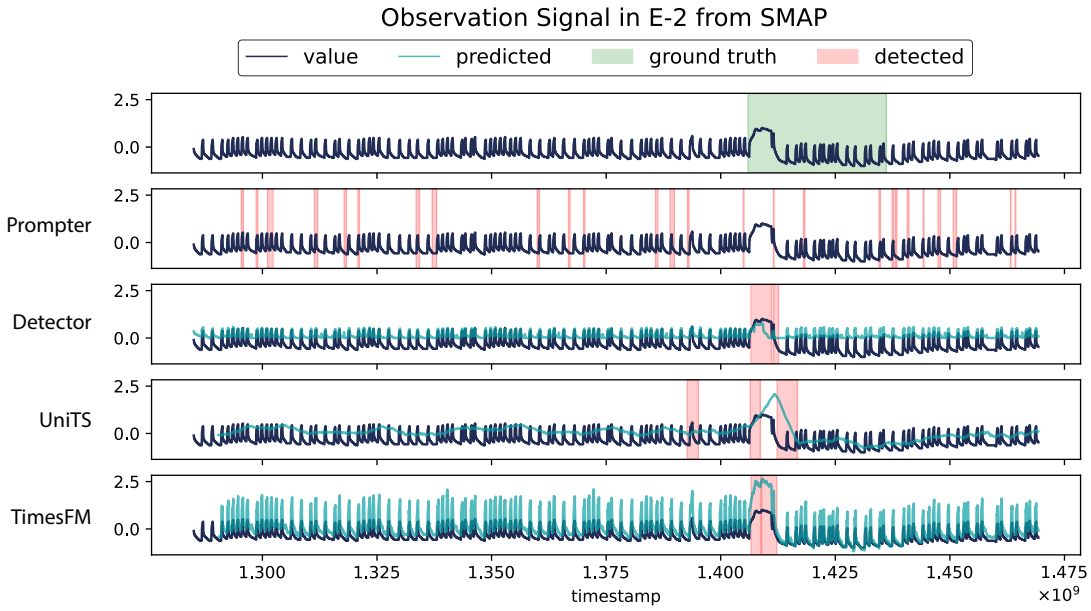


Figure 6-6: Visualizing `E-2` from SMAP detections by **Prompter**, **Detector**, `UniTS`, and `TimesFM` where the signal has contextual anomalies.

	NASA		Yahoo S5				NAB					
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	$\mu \pm \sigma$
`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**
`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
`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
`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
`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
`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
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
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
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
`UniTS`	0.541	0.560	0.640	0.718	0.027	0.066	0.364	0.476	0.667	0.649	0.615	0.484 ± 0.237
`TimesFM`	0.533	0.644	0.628	0.551	0.029	0.052	0.400	0.475	0.783	0.537	0.563	0.472 ± 0.235

Table 6.3: Benchmark Summary Results depicting F1 Score.

How do foundation models compare to existing approaches? Table 6.3 highlights the F_1 score obtained for each of the 11 datasets. `AER`, the current best deep learning model, is 22.8% better than the best LLM-based method and 26.6% better than the best TSFM-based approaches. Deep learning methods still perform better than FM-based methods by 10.9% on average.

FM-based methods can perform surprisingly well. The **Detector** method achieved an F_1 score 6.7% higher than that of `MAvg`, and only 5.7% lower than that of `ARIMA`, which is a reasonable model. Moreover, **Detector** surpasses the performance of `MAvg` in 6 out of 11

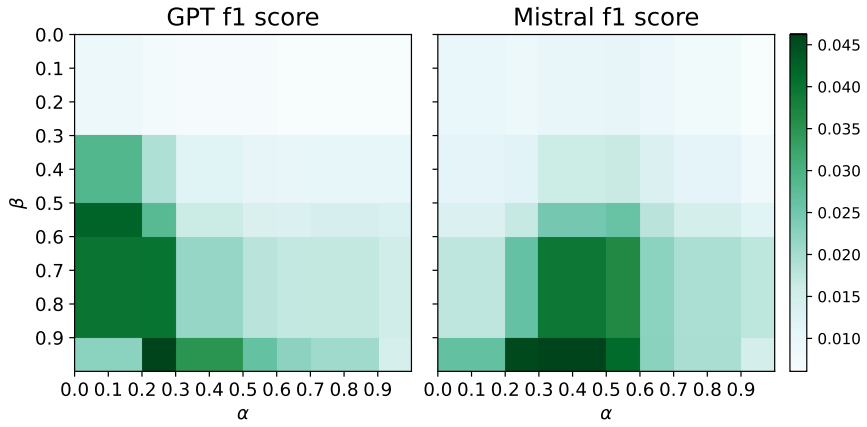


Figure 6-7: Optimizing the choice α and β values based on the average F_1 scores on all datasets.

datasets, and of `ARIMA` in 4 out of 11 datasets. We can best see the potential of **Detector** in the Tweets dataset, where the gap between the LLM's result and the highest result (from `MAvg`) is minimal, at only 1.4%. Moreover, FMs do not hold the highest score for any dataset. It is clear that there is a significant gap here, and an opportunity for improvement.

6.3.3 Ablation Study

Prompter. The **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 F_1 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. Figure 6-7 shows detection F_1 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 F_1 score; for `gpt`, $\alpha = 0.2$ and $\beta = 0.9$ yielded the best F_1 score as shown in Figure 6-7.

Detector Since we sampled multiple instances from the probability distribution (namely 10 samples in our experiments), we obtained a possible range of values that could be assigned to each particular point in time. We studied a multiple aggregation function to recreate a one-dimensional signal. Table 6.4 shows the detection F_1 score when the signal is recreated

	Variation		NASA		Yahoo S5				NAB					
			MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	$\mu \pm \sigma$
AE	with smoothing	mean	0.277	0.384	0.537	0.387	0.000	0.000	**0.400**	0.329	0.640	0.444	0.586	0.362 ± 0.199
		median	0.269	0.384	0.538	0.387	0.000	0.000	**0.400**	0.312	0.696	**0.480**	0.593	0.369 ± 0.210
		5%	0.294	0.400	0.542	0.387	0.000	0.000	**0.400**	0.308	**0.727**	0.417	0.615	0.372 ± 0.214
		95%	0.254	0.396	0.532	0.387	0.004	0.005	**0.400**	0.289	0.696	0.348	0.655	0.361 ± 0.214
	w/o smoothing	mean	0.412	0.350	0.563	0.762	0.060	0.129	0.235	0.282	0.625	0.368	0.325	0.374 ± 0.200
		median	0.412	0.337	0.572	0.762	0.060	0.127	0.235	0.286	0.625	0.359	0.333	0.373 ± 0.201
		5%	**0.429**	0.353	0.564	0.759	0.040	0.123	0.235	0.284	0.621	0.400	0.350	0.378 ± 0.203
		95%	0.406	0.337	0.608	0.765	0.114	0.167	0.235	0.288	0.600	0.387	0.342	0.386 ± 0.190
SE	with smoothing	mean	0.316	0.414	0.551	0.673	0.004	0.016	0.364	0.344	0.643	0.424	0.719	0.406 ± 0.228
		median	0.316	0.414	0.560	0.671	0.008	0.016	0.364	0.352	0.667	0.412	0.730	**0.410 ± 0.232**
		5%	0.333	0.400	0.552	0.662	0.000	0.014	0.364	**0.362**	0.621	0.400	0.730	0.403 ± 0.227
		95%	0.306	**0.431**	0.577	0.688	0.023	0.064	0.364	0.318	0.593	0.345	**0.762**	0.406 ± 0.225
	w/o smoothing	mean	0.344	0.257	0.567	**0.828**	0.324	0.315	0.333	0.279	0.541	0.280	0.220	0.390 ± 0.174
		median	0.344	0.247	0.583	0.824	0.323	0.315	0.333	0.281	0.541	0.259	0.220	0.388 ± 0.176
		5%	0.358	0.241	0.563	**0.828**	0.294	0.290	0.333	0.279	0.556	0.286	0.220	0.386 ± 0.178
		95%	0.390	0.238	**0.615**	0.796	**0.376**	**0.363**	0.235	0.287	0.588	0.293	0.246	0.402 ± 0.176

Table 6.4: F_1 Score of all variations of **Detector**

from mean, median, 5^{th}-percentile, and 95^{th}-percentile values of the predicted distribution. Moreover, we consider different error scores with and without smoothing.

We can see that, on average, squared error with smoothing on the median signal produced the best score, even though it was not the best performer on any individual dataset. The best configuration proved to be different for almost every dataset, with squared error producing the best results on Yahoo S5, and absolute error on NAB. One interesting observation is that the signal reconstructed from the $5^{th}/95^{th}$-percentile showed higher potential in revealing the location of anomalies.

6.4 Discussion

Prompting Challenges. Over a three-month experimental period, various prompts were employed, as laid out in Table C.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 adjusted 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

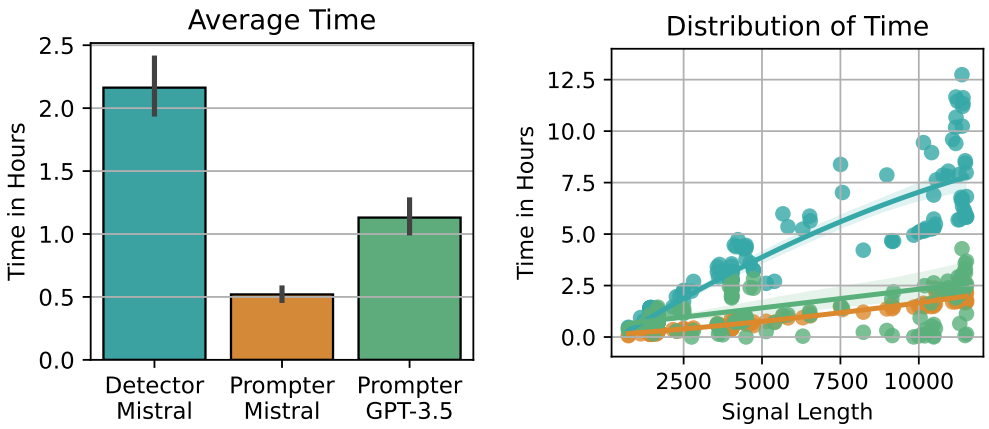


Figure 6-8: Recorded time for **Prompter** and **Detector**. (left) On average, **Detector** takes the longest to infer, almost double the time of **Prompter**. (right) Distribution of signal length and execution time.

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.

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 [Chang et al., 2023b], are notorious for memorizing training data [Biderman et al., 2023], 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. However, with TFSM, memorization can occur if the models were exposed to the testing data during their training.

Practicality of Usage. The appeal of using LLMs for this task lies in their ability to be used

in zero-shot, without necessitating any fine-tuning. However, this property is bottlenecked by the latency time. Figure 6-8 illustrates the average time it takes to use LLMs for each of the suggested approaches. It is unreasonable to wait half an hour to two hours for the model to produce a response, especially when deep learning models take less than an hour to train [Alnegheimish et al., 2024a]. Since these experiments were run in an offline setting, we can expect that real-time deployment would be more sensible, especially since the context size is much smaller, eliminating the need for rolling windows. In other cases, a single window would be sufficient, which takes approximately 5 seconds to infer depending on the window size.

In total, running **Prompter** experiments using the `gpt-3.5-turbo-instruct` version has cost us approximately $834.11 – an average of $1.69 per signal. For **Detector**, we ran a small-scale experiment where we sampled 22 signals of the data, roughly 5%. The total reached $95.08 – an average of $4.3 per signal – making **Detector** a more expensive method than **Prompter**.

6.5 Conclusion

In this chapter, we investigate a new paradigm for unsupervised time series anomaly detection through the lens of foundation models. Specifically, we examine the capability of large language models (LLMs) to perform anomaly detection in time series without prior task-specific training. To this end, we propose two methods: **Prompter**, which leverages prompting strategies, and **Detector**, which applies a forecasting-based approach. Our results demonstrate that LLMs identify anomalies more effectively under the forecasting paradigm (**Detector**) than with prompting (**Prompter**). Furthermore, we extend our study to time series foundation models (TSFMs), which are explicitly trained for time series forecasting. Despite their specialized training, we show that **Detector** surpasses both `TimesFM` and `UniTS` in F_1 score on average.

Chapter 7

Lessons from Industry

Over the course of this thesis, our models and systems were tested by industry teams. We learned which features successfully supported their workflows, and what challenges still remain. In this chapter, we summarize our findings.

Chapter outline:

- In Section 7.1, we collaborate with SES [1], a leading satellite operations company, to explore a case study on satellite telemetry data.
- In Section 7.2, we conduct verification testing on electric vehicles with Hyundai Motor Company.
- In Section 7.3, we summarize the learned lessons from conducting the aforementioned case studies.

7.1 AC3 Case Study

For over four years, we led a collaboration with world-leading communication satellite company, SES, that ended in 2024. SES monitors tens of thousands of signals on a daily basis. These signals report sensor readings collected at the satellite level. One major objective of the company's operations team is to detect unexpected behaviors (i.e., anomalies) in these

[1] https://www.ses.com/

(a) out-of-range events (b) hidden events

Figure 7-1: Anomalous event types within periodic signals.

signals. Based on the detected anomalies, the operations team can plan for the future of the satellite accordingly.

The **AC3** case study is a masked case study from the **A**ttitude **C**ontrol subsystem, where the anomalies are unknown to us. The real names of the columns are masked and the original values of the sensors are transformed, such as rescaling its value range, to hide any details that might reveal the meaning of the signal.

The remainder of section is outlined as follows: Section 7.1.1 describes the dataset used and Section 7.1.2 describes the methods that were used based on the Orion pipelines. Section 7.1.3 illustrates the overall results we collected. Section 7.1.4 discusses the results. We conclude in Section 7.1.5.

7.1.1 Dataset

This case study features data collected over 3 years and 11 months. The exact source of the data is unknown to us (the developers of Orion). However, these time series are similar to each other. Overall, there are 9 signals in this data, representing different observations and controls. Each data point is sampled every 0.5 seconds on average, creating over 200 million rows of data. We need to mark anomalous intervals within this dataset.

The general pattern of the data is a sinusoidal shape, and directly applying a threshold only discovers out-of-range events. We miss the anomalies that are buried in the signal. Figure 7-1 illustrates the differences between these two types of events. Recall in Section 2.3, we defined these hidden events as contextual anomalies. In this case study, events are typically hidden and must be localized. Even with an adaptive threshold mechanism, finding these events is difficult. Overall, there are 19 true anomalies in this case study, 5 of them are very short and the team of experts are not 100% certain if they are anomalies.

	base pipeline	input	aggregation
Method A	`signal processing` + windowed thresholding	univariate	0.5 seconds
Method B	`ARIMA`	univariate	6 hours
Method C	`AER`	univariate	6 hours
Method D	`signal processing` + static thresholding	univariate	0.5 seconds
Method E	`GANF`	univariate	6 minutes
Method F	`GANF`	multivariate	6 minutes
Method G	`AER`	multivariate	6 minutes
Method H	`AER` + interpolation	multivariate	6 minutes

Table 7.1: Summary of the methods used in the experiments.

7.1.2 Methods

In this case study, we used the Orion system to detect anomalies. We applied multiple pipelines to the dataset and reported the detected anomalies to the operations team, who then told us whether these detection were correct. We focus on three primary pipelines from Orion:

- `ARIMA`, a very common classical method that has been used for anomaly detection for decades [Box and Pierce, 1970].
- `AER`, the best performing model on OrionBench [Wong et al., 2022].
- `GANF`, this model was specifically requested by the team which had been featured in a news article [2] that caught their attention [Dai and Chen, 2022].

In addition, based on the description of the data, specifically that we are dealing with periodic signals that have hidden events, we develop a few methods, including a signal processing method. Table 7.1 summarizes these methods. We describe each method, its underlying pipeline, and their configurations below:

- **Method A.** This pipeline relies on signal preprocessing, removing the fundamental frequency from the signal to extract the periodicity, then computing the energy ag-

[2] https://news.mit.edu/2022/artificial-intelligence-anomalies-data-0225

gregated at *one day* intervals. Finally, it applies windowed thresholding to the energy signal.

- **Method B.** This is a classical method that uses `ARIMA` on the original signal in a univariate setting, with an aggregation level of six hours, i.e. `interval=21600` seconds.
- **Method C.** In addition to using a classic method, we use `AER`, which is the best-performing pipeline in OrionBench. The method is applied with the same settings as Method B.
- **Method D.** This method is similar to Method A, but applies a global static threshold rather than a windowed approach. The static threshold is set to two standard deviations away from the mean.
- **Method E.** This pipeline uses a `GANF` model, applied in a univariate setting to all signals including observation signals and control signals. The aggregation level is set to six minutes, i.e. `interval=360` seconds.
- **Method F.** This method is similar to Method E, however the input is multivariate.
- **Method G.** This method is similar to Method C, which is an `AER` model, with the aggregation level set to six minutes instead of six hours. This pipeline is also applied to multivariate inputs.
- **Method H.** This method is similar to Method G, however we swapped the naive imputation method `SimpleImputer`, which fills missing values with the average value of the signal, to an interpolation method.

All univariate methods are applied to the observation signals only, unless stated otherwise.

7.1.3 Results

Table 7.2 shows the number of flagged anomalies for each method, and their average length. The operations team reviewed the flagged anomalies and provided feedback on whether the detected event is correct or not. We note that in the multivariate case, the detected anomalies

	# anomalies	avg. length of anomalies	# correctly flagged	comment
Method A	28	3 days ± 2	0	
Method B	30	34 days ± 34	2	too long
Method C	23	7 days ± 3	4	2 unique
Method D	94	1 days ± 0	0	
Method E	51	4 days ± 2	10	5 unique
Method F	80	3 days ± 1	20	5 unique
Method G	15	2 days ± 1	0	
Method H	17	3 days ± 1	3	3 unique

Table 7.2: Results of each method and the number of correctly flagged anomalies.

can include the same time interval appearing in different signals. Therefore, we clarify how many correctly flagged anomalies are unique intervals.

Classic versus deep learning methods. Methods A and D, both signal processing techniques, failed to successfully identify any anomalies. These methods are sensitive to noise flagging any events that are considered as normal. Method B, which used ARIMA, detected a set of anomalies with an average duration of longer than a month – much too long for experts to review or use in decision-making. Therefore, Method B cannot be adopted by SES.

Multivariate versus univariate methods. Methods based on the `GANF` pipeline (E and F) detected the exact same anomalies in both univariate and multivariate settings (5 unique events). However, there was a clear difference between `AER`-based pipelines (C and H). Method C, a univariate pipelines, and Method H, a multivariate pipeline, detected completely independent sets of anomalies, bringing the total number of detections by `AER` to 5 unique events.

Ensembling. Different methods detected various events that others did not catch. Method C detected 2 unique events. Methods E and F detected the same 5 unique intervals, and Method H detected 2 events previously discovered by E and F, as well as one additional unique event. In total, using all these methods, we detected at least 8 different anomalies. Note that during the feedback process, we only learned whether an interval was correctly flagged – it is possible we actually detected more anomalies, if certain intervals contained more than one.

Based on these results, we see that Methods A, D, and G did not detect any anomalies. Moreover, although Method B discovered 2 anomalies, the duration of the detected interval is too long to be practical. Overall, we conclude that Methods C, E, F, and H are the only ones that produce viable results.

7.1.4 Evaluation and Discussion

Evaluating the performance of these methods is non-trivial given the opaqueness of defining normal and anomalous events. In an effort to produce valuable metrics, we perform the following. Given that there are 1442 days recorded in the data, and assuming that an event is 2 days in length, the total number of possible events is: 1442 days $\rightarrow$ 721 possible events. We can use this to determine true and false positive rates for the different methods.

$$\text{TPR} = \frac{\text{number of unique events}}{\text{number of true anomalies}}, \quad \text{FPR} = \frac{\text{number of incorrect events}}{\text{number of possible events}}$$

For example, for Method C, based on the number of ground truth anomalies and the number of flagged anomalies, we get a **True Positive Rate (TPR)** $= \mathbf{2/19} = \mathbf{11}\%$ and a **False Positive Rate (FPR)** $= \mathbf{(23-4)/721} = \mathbf{19/721} = \mathbf{2.6}\%$. We summarize these evaluation scores in Table 7.3. Given the large amount of data, Orion was successful in finding anomalies without raising too many false alarms, especially with method H. False positives are important to consider because without them, methods could mark an unlimited number of events as anomalous to increase the true positive rate.

Moreover, the evaluation metrics reported in this case study are on the lower side compared to the performances we have seen on public data in Chapter 5. We would like to note that this has to do with two main properties:

1. **Anomaly Ambiguity:** The anomalies present in this dataset are particularly hard to find, even to the team of experts. After inspecting the events detected by these methods and qualitatively assessing them, there is clearly a significant shift from the typical pattern of the signal. However, to the team of experts, this change in pattern is not "problematic" to them, marking it as not anomalous. This nuanced definition of anomalies drastically influenced the reported metrics.

	TPR (↑)	FPR (↓)	Precision (↑)	F_1 Score (↑)
Method C	$2/19 = 11$ %	$19/721 = 2.6\%$	0.095	0.100
Method E	$5/19 = 26$ %	$41/721 = 5.7\%$	0.109	0.154
Method F	$5/19 = 26$ %	$60/721 = 8.3\%$	0.077	0.119
Method H	$3/19 = 16$ %	$14/721 = 1.9\%$	0.176	0.167

Table 7.3: Scores for each method in AC3 case study.

2. **Evaluation strategy:** In our report, we evaluated all methods with a conservative approach. First, the team was only confident that there are 14 true anomalies out of 19, however, we decided to proceed with finding all 19 of them. Moreover, the detected event might overlap with more than one true anomaly, however, the team did not share this information with us. Therefore, we assume we only found single anomaly based on the expert's feedback.

7.1.5 Conclusion

This case study presents a real evaluation of whether the pipelines provided in Orion can find anomalies in industrial data. Testing 8 methods, we revealed 8 distinct anomalies out of 19. The best-performing method is an `AER`-based pipeline which achieves an $F_1 = 0.167$.

7.2 Electrical Vehicles (EV) Case Study

Electrical vehicles undergo various tests to verify if they are functioning correctly. In this case study, we look at vehicle control logic in Hyundai Motor Company's electrical vehicles. The validation tests are conducted in various ways, from offline unit testing to real-time vehicle testing. One of their current verification methods involves test cases, which are carefully designed by engineers based on their own expertise. Specifically, Vehicle Control Unit (VCU), which has over 50 functions, has more than 200 test cases to validate each function. These tests are written manually by the engineers. As the functionalities of electric vehicles become more complex, the number of test cases continues to grow. Moreover, the impact of control settings on the behavior of signals significantly limits the coverage of

verification tests. Rather than depending on manually engineered tests, in this case study, we investigate the feasibility of using Orion to test the VCU's i-Pedal function.

7.2.1 Dataset

This study focuses on time series data from the i-Pedal function of the VCU. The "i-Pedal" function determines the acceleration and deceleration operations of the vehicle. Note that – unlike in combustion-engine vehicles where one must press the brake pedal to decelerate – pressing the pedal of an electric vehicle causes acceleration and releasing immediately decelerates. The data consists of 18 time series and was gathered from driving an electric vehicle under normal conditions. It includes acceleration, speed, torque command, and other signals. Then, engineers synthetically introduce anomalies to the collected signals. The anomalies are divided into two categories: (1) *functional* anomalies related to control logic malfunctions, and (2) *driving-feeling* anomalies related to smoothness and responsiveness of the vehicle, which can be simulated by perturbing map values or state transitions so that vehicle response no longer follows the desired feel curves. After initial data exploration, the team selected 8 real signals out of the 18 and synthesized 3 more. Overall, 11 signals in the dataset were utilized, with 18 total anomalies.

7.2.2 Pipelines

In this case study, the team directly used Orion and focused on four pipelines:

- `AER`: the best-performing pipeline in OrionBench [Wong et al., 2022].
- `LSTM DT`: a competitive baseline in the literature, and currently second place in OrionBench [Hundman et al., 2018].
- `TadGAN`: our proposed GAN-based model [Geiger et al., 2020].
- `LSTM AE`: a well-known baseline in the field [Malhotra et al., 2016].

7.2.3 Results

The team used OrionBench on their own data to benchmark the selected pipelines.

	Weighted Segment				Overlapping Segment			
Pipeline	Accuracy	F_1	Recall	Precision	F_1	Recall	Precision	# anomalies
`AER`	0.966	0.858	0.851	0.864	0.875	0.777	1.000	14
`LSTM DT`	0.916	0.668	0.740	0.606	0.736	0.777	0.700	13
`TadGAN`	0.756	0.352	0.620	0.246	0.571	0.555	0.588	11
`LSTM AE`	0.838	0.303	0.330	0.283	0.412	0.722	0.288	13

Table 7.4: Scores for each pipeline in the EV case study.

Overall Performance. Table 7.4 shows how each pipeline performed in discovering the anomalies present in the time series. `AER` had the best performance with regards to all metrics. If we focus on weighted F_1 scores, we can see a 19% improvement over the next-best model, which is `LSTM DT`. The results obtained in this study are consistent with our benchmarks of public datasets (see Table C.3 in the Appendix).

Effect of Signal Processing. To enhance performance, three signals were generated from the data.

- **Jerk**: this signal is generated by calculating the time derivative of acceleration values.
- **Label**: this is a categorical signal representing possible speed states of the vehicle. These states include: stationary, decelerating, accelerating, and cruising.
- **Delta**: this signal represents the absolute difference between the vehicle speed and the active position switch value.

Ultimately, these derived variables helped improve pipeline performance. The F_1 score improved 7.6% (0.782 $\rightarrow$ 0.858) when the derived variables were added, compared to when only the original variables were used.

7.2.4 Conclusion

This case study was designed to test the feasibility of using unsupervised time series anomaly detection to verify the behavior of the i-Pedal function of the VCU. By testing various pipelines, we saw how `AER` managed to find 14 out of 18 anomalies without raising any false alarms. Moreover, incorporating additional signals that effectively describe data features

improved the model's performance. However, with more signals, training time also increases, especially for `TadGAN`. We recommend carefully synthesizing signals to account for this trade-off. During the course of this study, the Hyundai team used the Orion system directly. They relied completely on the open-source software and online documentation, with little to no intervention by the Orion development team.

7.3 Lessons Learned

When conducting these case studies, we were presented with several challenges. These challenges are often never faced when dealing with public datasets that are in pristine conditions and ready for the machine learning model. Moreover, without a clear ground truth set, the evaluation of models can be nuanced. we detail what we have learned below:

L1. Confirmation of Anomalies is Hard. The definition of what is an anomaly is vague when working with both industries. With SES, the ground truth was not definitive, claiming that 5 events that were marked as an anomaly are "short" and they are not 100% of their correctness. Moreover, Hyundai worked on synthetically introducing the anomaly to their dataset in order to be certain of the detection.

L2. Importance of Detection Length. In the AC3 case study, the team informed us that anomalies typically span a length of seconds, or maybe minutes, but definitely not months. This emphasized the importance of dealing with the time series in its fine-granularity.

L3. Aggregation of Data. Fine-granular data posed another challenge during visualization. To allow expert revision and annotation, we show the results on MTV. However, the AC3 case study has over 200 million rows, it was hard to visualize the signals in raw format. This situation required intricate engineering of the visualization tool to allow multiple aggregations such that viewers could switch granularity while maintaining low latency.

L4. Value of Signal Processing. Deep learning has shown great ability of extracting temporal features from time series data and learning its underlying non-linear relationships. However, what Hyundai showed is that there is value in extracting features based on the expert's knowledge, resulting in better overall detection.

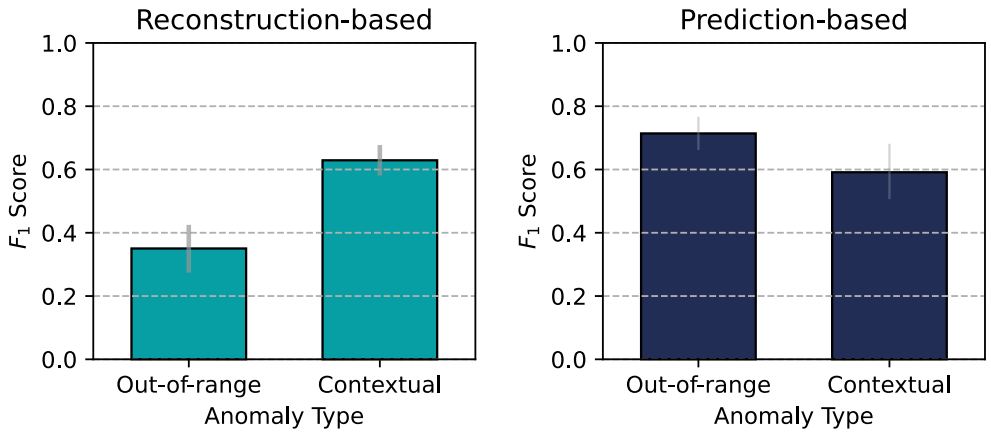


Figure 7-2: Average F_1 score performance of reconstruction– and prediction-based pipelines based on the anomaly type.

L5. Evaluation of Pipelines. Since Hyundai synthetically injected anomalies to their data, the evaluation was streamlined. However, we faced challenges in the AC3 case study. First, the total number of possible events, i.e. non-normal sequences, is unknown. Moreover, hypothetically, a detected event may contain more than one ground truth anomalies. However, the feedback style for this particular case study did not reveal whether this was true. We had to assume each detection found only a single event.

L6. Practicality of our Systems. The EV case study was conducted by a senior engineer at Hyundai, who independently learned to use our system through the available online documentation and open-source software. The engineer was able not only to fully customize and build a pipeline, but also to develop new primitives tailored to the specificity of their data. They subsequently executed the benchmark using OrionBench. This successful external adoption is a testament to both the maturity and the usability of our systems.

L7. Blind Testing as the Norm. Blind testing should become the standard practice for validating anomaly detection systems in industry. When a system (or algorithm) claims strong performance, industry establishes a sandbox environment by providing time series signals with anomalies embedded at undisclosed locations. The system is then tasked with identifying these anomalies without prior knowledge of their positions. Such practices offer a transparent and rigorous means of validation and should be encouraged, as they strengthen user confidence in the reliability and robustness of the system.

L8. Difficulty of Pipeline Selection. Pipeline selection is inherently challenging, even when public benchmarks are available. The most practical approach is to draw comparisons between benchmark datasets and the characteristics of a user's own data. For instance, Figure 7-2 categorizes the performance by anomaly type. If a dataset is expected to contain predominantly out-of-range anomalies, prediction-based pipelines are preferable, whereas reconstruction-based pipelines should be avoided. On the other hand, when contextual anomalies are anticipated, reconstruction-based pipelines generally yield better performance. In cases where the anomaly type is uncertain, `AER` offers a robust alternative by combining both prediction and reconstruction approaches.

Chapter 8

Conclusion and Future Work

8.1 Synopsis

In this dissertation, we tackle the *usability* challenges of unsupervised time series anomaly detection models, and present our models and systems to improve their adoption.

In Chapter 2, we describe the time series as part of the scope of this thesis and the general process of how machine learning models can be used for unsupervised time series anomaly detection. We also summarize recent advances in unsupervised anomaly detection models and systems in Chapter 3.

In Chapter 4, we formally define the problem and propose three models to solve it. First, we use a generative adversarial network in TadGAN to find anomalies using a combination of the reconstruction and critic scores. Second, we use a prediction- and reconstruction-based model, AER, to combine the advantages of each approach. Third, we propose incorporating contextual signals into our MixedLSTM model to learn a better representation of the time series. We finish the chapter by introducing two approaches for range-based evaluation.

In Chapter 5, we present our Orion system, which abstracts models into a universal representation of primitives and pipelines. Orion allows users to find anomalies on their own data using any available pipeline in a plug-and-play fashion. It provides users with a complete experience, including data loading, pipeline selection and evaluation, and visualization. We complement Orion with a benchmarking framework OrionBench that periodically evaluates results and publishes a public scoreboard.

In Chapter 6, we closely examine the ability of pretrained models to perform the time series anomaly detection task. We investigate large language models by converting time-series into text, then prompting the LLM to either detect the anomaly or to forecast the signal in order to ultimately find the anomaly. Moreover, the recent proposal of time series foundation models raised the question of whether they can detect anomalies without any additional training.

In Chapter 7, we conduct two case studies, applying our Orion system and our pipelines to find anomalies in satellite telemetry and electric vehicle data. We summarize our learnings and describe what challenges we faced at the modeling and evaluation stages.

8.1.1 Open-Source Contributions

Both Orion [1] and SigLLM [2] are open source and can be downloaded on pypi. Over the years, both libraries have garnered communities. As of writing this thesis, Orion has been downloaded 120,000 times and has over a thousand stars on github, Table 8.1 lays out some of these achievements. This has been a collective effort by me (`sarahmish`) and my collaborators, Table 8.2 shows the contribution statistics.

Library	# Releases	# Downloads	# Stargazers	# Questions	Forks	Created
Orion	22	120,000	1,298	141	191	2019
SigLLM	4	6,000	74	11	24	2023

Table 8.1: Library statistics

					Code Contribution	
		Commits	PR	Issues	++	−−
Orion	general	553	196	502	145,232	27,251
	`sarahmish`	303	125	198	96,573 (66.5%)	12,547 (46%)
SigLLM	general	80	24	44	16,969	6,324
	`sarahmish`	43	20	27	9,893 (58.3%)	3,082 (48.7%)

Table 8.2: Contributions to Orion and SigLLM library.

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

8.2 Future Work

We are still at the beginnings of exploring the capabilities of foundation models, particularly in unsupervised time series anomaly detection. As future work, we will explore the following directions.

8.2.1 Multivariate Time Series

- *Expanding to multivariate output.* Currently, our developed models, including `TadGAN`, `AER`, and `MixedLSTM` as well as both the Orion and OrionBench systems, focus on multivariate input and univariate output (as described in Section 2.3.2). In order to detect anomalies in all channels of the multivariate time series, the user must create replicas of the pipeline and alter the target channel in each copy. For pipelines to become more efficient, the model needs to (1) be able to perform the detection on multiple channels; (2) indicate which channel caused the anomaly to be flagged.

- *Handling multivariate input in large language models.* Our experiments in utilizing large language models focused primarily on univariate time series. To handle multivariate input, we can explore concatenating the channels (i.e. flattening the signal). However, this can produce an arbitrarily long sequence that cannot be fed into the LLM in a single sequence. We also propose interleaving the channel values such that all values at a particular timestamp are located next to each other. The last approach we suggest is to perform a mixing operation to convert a multivariate time series into a univariate one. This approach can result in information loss.

8.2.2 Explanation

- *Providing explanations alongside detections.* Part of the manual process of visually inspecting a signal to find anomalies involves naturally explaining the behavior of the normal signal and how the anomaly deviates from it. For machine learning to be successful, it must be able to present to the user the reasoning behind why it identified a certain region as an anomaly. With the advancements of large language models,

we inch towards curating natural explanations for anomalous segments. This can be explored as a post-hoc step to any model, or it can be incorporated into the detection process (within the prompt) in **Prompter**.

8.2.3 Closing the Loop

- *Continual learning.* In this dissertation, we describe the end user workflow of loading the data, selecting a model, evaluating its performance, detecting anomalies, and annotating them. This scope does not address how these annotations are used later during the feedback process. We explored the training of a semi-supervised model on the collected annotations; however, without a sufficient number of annotations, the model performs less competitively than unsupervised models. Rather than training a completely new model, the unsupervised models can evolve and continue to learn from expert feedback.

Appendix A

Additional Details

A.1 arXiv Papers

We scanned arXiv for papers proposing a new method or model for time series anomaly detection. We used arXiv's advanced search query with the following settings:

- **terms:** "anomaly detection time series" present in the abstract.
- **subject:** all classifications.
- **date:** specific year "2024" by "submission date (original)".

The search [1] returned 199 papers, out of which 109 papers were proposing a new model. Any paper that mentioned "supervised" or "semi-supervised" in the abstract was removed from the final count.

[1] https://arxiv.org/search/advanced?advanced=&terms-0-operator=AND&terms-0-term=anomaly+detection+time+series&terms-0-field=abstract&classification-physics_archives=all&classification-include_cross_list=include&date-filter_by=specific_year&date-year=2024&date-from_date=&date-to_date=&date-date_type=submitted_date_first&abstracts=show&size=50&order=announced_date_first&start=0

A.2 Time Series Foundation Models Training Corpus

A.2.1 UniTS

The authors trained the `UniTS` model on 39 datasets from several sources focusing on time series forecasting and classification. Table A.1 lists all these datasets [Gao et al., 2024]. The datasets are from finance, electricity, traffic, climate, healthcare, etc. Overall, there are 21 datasets for forecasting and 18 for classification. To our knowledge, these datasets do not include any of the time series anomaly detection datasets listed in Section 3.1.

A.2.2 TimesFM

`TimesFM` utilizes data from Google Trends [2], Wiki Pageviews [3], and other publicly available datasets. Moreover, they synthetically generate training data via an ARMA process while varying seasonality and trend. Overall, their training corpus reaches over 300 billion time points, consisting mainly of Wiki Pageviews.

Google Trends. This dataset captures trending search queries over time. The authors selected 22,000 queries between 2007 and 2022 with multiple granularity levels, including hourly, daily, weekly and monthly. This datasets contains roughly 500 million time points.

Wiki Pageviews. This dataset captures the hourly views of all Wikimedia pages. The authors collect data between 2012 and 2024, then aggregate the time series to hourly, daily, weekly and monthly granularities. This dataset contains roughly 300 billion time points.

Synthetic Data. The authors create synthetic time series via an ARMA process, varying the seasonal patterns, trends, and step functions. In addition, they mix multiple processes together to create a more complex time series. Overall, they create 3 million synthetic time series, each with length 2048.

Other Data Sources. The authors utilize publicly available data source such as the M4, Eletricity, Traffic, and Weather datasets [Makridakis et al., 2022, Zhou et al., 2021].

While the authors do not mention the utilization of datasets mentioned in Section 3.1 specifically, there is speculation that it was part of the training corpus.

[2] https://trends.google.com/

[3] https://en.wikipedia.org/wiki/Wikipedia:Pageview_statistics

A.3 OrionBench

A.3.1 Limitations

We acknowledge several limitations of our framework and results. Black box pipelines such as Microsoft's `Azure AD` service lack certain levels of transparency. In Figure 5-11, we noticed that `Azure AD` improved in `0.1.6` $\rightarrow$ `0.1.7`. However, unlike with other pipelines where we can cross reference changes in behavior to code modification or even updated dependency package releases, we have no knowledge of what caused this improvement. Nevertheless, we are still able to monitor the performance of black box pipelines and have some confidence in their stability.

In addition, benchmarks are notorious for requiring massive computing resources, and this case is no different. While the models can vary in ussde, to perform a comprehensive benchmark, we utilize MIT supercloud [Reuther et al., 2018]. When computing resources are limited, on-demand benchmark runs become difficult. We aim to alleviate this challenge with continuous and periodic running benchmarks. Moreover, with every introduction to a new model, a benchmark must be run to add the model to the leaderboard.

Lastly, and most importantly, there is no guarantee that these pipelines will deliver the same performance on real-world datasets. A clear example was demonstrated in Section 5.3.4 Scenario 2 where `GANF` produced valuable results for the end-user, however, its results in the benchmark were not as promising as those of some other pipelines. This stresses the importance of pipeline selection based on the characteristics of the datasets and anomalies. Further research is needed to understand the suitability of unsupervised pipelines for a given dataset.

A.3.2 Primitives & Pipelines

Primitive Template

Abstractions in OrionBench of primitives and pipelines are universal representations of end-to-end models, from a signal to a set of detected anomalies. Compared to standard `scikit-learn`-like code, this requires one additional step of creating `json` files to define

these primitives. Figure A-1 showcases a template that helps contributors guide their own primitive.

Once primitives are built, they can be stacked to create a pipeline similar to the example shown in Figure A-2. This example shows the `json` file representation of `LSTM DT` pipeline.

Pipelines

Currently in *OrionBench*, there are 12 readily available pipelines, all unsupervised. All pipelines and their hyperparameter settings for the benchmark can be explored directly: https://github.com/sintel-dev/Orion/tree/master/orion/pipelines/verified. Below we provide further detail on the mechanisms behind each pipeline.

`ARIMA` [Box and Pierce, 1970, Pena et al., 2013]. `ARIMA` is an autoregressive integrated moving average model which is a classic statistical analysis model. It is a forecasting model that learns autocorrelations in the time series to predict future values. It has been adapted for anomaly detection. The pipeline computes the prediction error between the original signal and the forecasted one using simple point-wise errors. Then, it pinpoints anomalies based on when the error exceeds a certain threshold. Specifically, the `ARIMA` pipeline uses a moving window-based thresholding technique defined in the `find_anomalies` primitive.

`AER` [Wong et al., 2022]. `AER` is an autoencoder with regression pipeline. It combines prediction and reconstruction models. More specifically, it produces bi-directional predictions (forward & backward) while simultaneously reconstructing the original time series by optimizing a joint objective function. The error is then computed as a point-wise error for both forward and backward predictions. Dynamic time warping is used for reconstruction, which computes the euclidean distance between two time series where one might lag behind the other. The total error is then computed as a point-wise product of the three aforementioned errors.

`LSTM DT` [Hundman et al., 2018]. `LSTM DT` is a prediction-based pipeline using an LSTM model. Similar to `ARIMA`, it computes the residual between the original signal and predicted one using smoothed point-wise error. Then it applies a non-parametric thresholding method to reduce the amount of false positives.

```
{
    "name": "orion.primitives.primitive.PrimitiveName",
    "contributors": ["Author <email>"],
    "documentation": "reference to documentation or paper if available.",
    "description": "short description.",
    "classifiers": {
        "type": "postprocessor",
        "subtype": "anomaly_detector"
    },
    "modalities": [],
    "primitive": "orion.primitives.primitive.PrimitiveName",
    "fit": {
        "method": "fit",
        "args": [
            {
                "name": "X",
                "type": "ndarray"
            },
            {
                "name": "y",
                "type": "ndarray"
            }
        ]
    },
    "produce": {
        "method": "detect",
        "args": [
            {
                "name": "X",
                "type": "ndarray"
            },
            {
                "name": "y",
                "type": "ndarray"
            }
        ],
        "output": [
            {
                "name": "y",
                "type": "ndarray"
            }
        ]
    },
    "hyperparameters": {
        "fixed": {
            "hyper_name": {
                "type": "str",
                "default": "value"
            }
        }
    }
}
```

Figure A-1: Primitive template. The first section of the `json` describes metadata, the second part contains functional information including the names of the methods and their arguments, and the third part defines the hyperparameters of the primitive.

```
{
    "primitives": [
        "mlstars.custom.timeseries_preprocessing.time_segments_aggregate",
        "sklearn.impute.SimpleImputer",
        "sklearn.preprocessing.MinMaxScaler",
        "mlstars.custom.timeseries_preprocessing.rolling_window_sequences",
        "keras.Sequential.LSTMTimeSeriesRegressor",
        "orion.primitives.timeseries_errors.regression_errors",
        "orion.primitives.timeseries_anomalies.find_anomalies"
    ],
    "init_params": {
        "mlstars.custom.timeseries_preprocessing.time_segments_aggregate#1": {
            "time_column": "timestamp",
            "interval": 21600,
            "method": "mean"
        },
        "sklearn.preprocessing.MinMaxScaler#1": {
            "feature_range": [
                -1,
                1
            ]
        },
        "mlstars.custom.timeseries_preprocessing.rolling_window_sequences#1": {
            "target_column": 0,
            "window_size": 250
        },
        "keras.Sequential.LSTMTimeSeriesRegressor": {
            "epochs": 35
        },
        "orion.primitives.timeseries_anomalies.find_anomalies#1": {
            "window_size_perc": 30,
            "fixed_threshold": false
        }
    }
}
```

Figure A-2: `LSTM DT` pipeline example. This is the content present in the `json` file of the pipeline. The first section defines the stack of primitives used in the pipeline, which will be computed to the graph shown previously in Figure 5-1a. The `init` argument initializes some of the hyperparameters for each primitive.

`TadGAN` [Geiger et al., 2020]. `TadGAN` is a reconstruction pipeline that uses generative adversarial networks to generate a synthetic time series. To sample a "similar" time series, the model uses an encoder to map the original time series to the latent dimension. There are three possible strategies to compute the errors between the real and synthetic time series, specifically point-wise errors, area difference, and dynamic time warping. Most datasets are set to dynamic time warping (dtw) as error.

`MP` [Yeh et al., 2016]. `MP` is a matrix profile method that seeks to find discordance in time series. The pipeline computes the matrix profile of a signal, which essential provides the closest nearest neighbor for each segment. Based on these values, segments with large distance values to their nearest neighbors are anomalous. We use `find_anomalies` to set the threshold dynamically.

`VAE` [Park et al., 2018]. `VAE` is a variational autoencoder consisting of an encoder and a decoder with LSTM layers. Similar to previous pipelines, it adopts reconstruction errors to compute the deviation between the original and reconstructed signals.

`LSTM AE` [Malhotra et al., 2016]. `LSTM AE` is an autoencoder with an LSTM encoder and decoder. This is a simpler variant of `VAE`. It also uses reconstruction errors to measure the difference between the original and reconstruction signals.

`Dense AE`. `Dense AE` is an autoencoder with properties similar to that of `LSTM AE`, with the exception of the encoder and decoder layers.

`GANF` [Dai and Chen, 2022]. `GANF` is a density-based method that uses normalizing flows to learn the distribution of the data, with a graph structure to overcome the challenge of high dimensionality. The model outputs an *anomaly measure* that indicates where the anomalies might be. To convert the output into a list of intervals, we add a `find_anomalies` primitive.

`Azure AD` [Ren et al., 2019]. `Azure AD` is a black box pipeline which connects to Microsoft's anomaly detection service [4]. To use this pipeline, the user needs a subscription to the service. The user can then update the `subscription_key` and `endpoint` in the pipeline `json` for usage.

[4] https://azure.microsoft.com/en-us/products/cognitive-services/anomaly-detector/

`AnomTransformer` [Xu et al., 2022b]. `AnomTransformer` is a transformer-based model using a new *anomaly-attention* mechanism to compute the association discrepancy. The model amplifies the discrepancies between normal and abnormal time points using a minimax strategy. The threshold is set based on the attention values.

A.3.3 Data Format

A time series is a collection of data points indexed by time. Time series can be stored in many forms. We define a time series as a set of time points, which we represent through integers denoting *timestamps*, and a corresponding set of values observed at each respective timestamp. Note that no prior pre-processing is required, as all pre-processing steps are part of the pipeline, e.g. imputations, scaling, etc.

A.4 Evaluation

This section provides further details on our evaluation setup and results. Code for reproducing Figures and Tables is provided at https://github.com/sarahmish/orionbench-paper

A.4.1 Evaluation Setup

Results presented in Section 5.3 are reported based on version `0.5.2` of Orion which is also released on pip [5]. We recommend setting up a new Python environment before installing Orion. Currently, the library is supported in `python 3.8`, `3.9`, `3.10`, and `3.11`.

Evaluation Strategy. Measuring the performance of unsupervised time series anomaly detection pipelines requires going beyond the usual classification metrics. OrionBench compares detected anomalies with ground truth labels according to well-defined metrics. This can be done with either weighted segment or overlapping segment Alnegheimish et al. [2022]. For evaluations in this paper, we use *overlapping segment* exclusively. Using overlapping segment, for each experiment run (which is an evaluation of one pipeline on one signal), we record the number of true positives (TP), false positive (FP), and false negatives (FN)

[5] https://pypi.org/project/orion-ml/0.5.2/

obtained. Because anomalies are scarce, and many signals contain one or zero, in many cases precision and recall scores will be undefined on a signal level. Therefore, we compute the scores on a dataset level.

$$precision = \frac{\sum_{s \in \mathcal{S}} TP_s}{\sum_{s \in \mathcal{S}} TP_s + FP_s} \qquad recall = \frac{\sum_{s \in \mathcal{S}} TP_s}{\sum_{s \in \mathcal{S}} TP_s + FN_s}$$

For a given dataset with a set of signals $\mathcal{S}$, we compute the total true positives, false positives, and false negatives for every signal in that set. Then we compute the score for each pipeline according to the metric of interest, whether it is precision, recall, or f1 score. The computation of f1 score is standard from precision and recall ($f1 = 2 \times \frac{precision \times recall}{precision + recall}$).

Recorded Information. During the benchmark process, information regarding performance, computation, diagnostics, etc. gets recorded. Below, we list all the information we store for each experiment. An experiment is defined as a single pipeline trained on a single time series and then used for detection for the same time series.

- *dataset:* the dataset that the signal belongs to, e.g. `SMAP`.
- *pipeline:* the name of the pipeline, e.g. `AER`.
- *signal:* the name of the signal, e.g. `S-1`.
- *iteration:* each experiment can be run for k iterations.
- *f1, precision, recall:* the evaluated metrics (in many cases, this is undefined).
- *tn, fp, fn, tp:* the evaluated number of true negatives, false positives, false negatives, and true positives, respectively. In an overlapping segment approach, *tn* does not have a value, given the nature of evaluation.
- *status:* whether the experiment ran from beginning to end without issue.
- *elapsed:* how much runtime each experiment required (includes training and inference).
- *run_id:* the process identification number.

The benchmark results are saved as `.csv` files and stored directly in the Github repositories: https://github.com/sintel-dev/Orion/tree/master/benchmark/results. Moreover, the pipelines used in each experiment are saved for reproduciblity measures. Due to their large size, we store these pipelines on a local server. However, our plan is to make these pipelines public, such that they can be used and inspected.

A.4.2 Computational Cost Across Releases

In addition to quality stability as shown in Figure 5-11, we can monitor the runtime execution of the benchmark. We illustrate the average runtime for each pipeline across 15 releases in Figure B-5. There is a clear improvement in average runtime in release `0.2.1`. This increase in speed traces back to an internal change in the API code. More specifically, pipeline builds were adjusted to only occur once, to reduce overhead during the `fit` and `detect` processes. Looking back at the development plan, this is reflected in Issue #261 where we see exact alterations made to the code.

Moreover, in version `0.4.0`, the package migrated to `TensorFlow 2.0` which consequently made the pipelines faster in GPU mode. However, in version `0.4.1` the pipelines were executed without GPU, which is evident from the slight increase in runtime.

Task	Dataset	Train Size	Sequence Length	Class
Forecast	NNS_P112 [98]	409	112	Finance
	ECL_P60 [102]	18221	96	Electricity
	ECL_P192 [102]	18125	96	Electricity
	ECL_P336 [102]	17981	96	Electricity
	ECL_P720 [102]	17597	96	Electricity
	ETTh_P96 [127]	8449	96	Electricity
	ETTh_P192 [127]	8333	96	Electricity
	ETTh_P336 [127]	8209	96	Electricity
	ETTh_P720 [127]	7825	96	Electricity
	Exchange_P192 [153]	5024	96	Electricity
	Exchange_P336 [153]	4880	96	Electricity
	Traffic_P192 [11]	581	96	Traffic
	Traffic_P336 [11]	581	96	Traffic
	Traffic_P60 [11]	12089	96	Traffic
	Traffic_P192 [92]	11993	96	Traffic
	Traffic_P336 [92]	11849	96	Traffic
	Traffic_P720 [92]	11465	96	Traffic
	Weather_P96 [110]	36696	96	Weather
	Weather_P192 [110]	36600	96	Weather
	Weather_P336 [110]	36456	96	Weather
	Weather_P720 [110]	36072	96	Weather
Classification	Spoken&Japanese_Digits [99]	965	60	Audio
	Spoken&Arabic&Digits [99]	270	29	Audio
	Spoken&Arabic&Digits [99]	6599	93	Audio
	Heartbeat [61]	204	405	Audio
	ECG0006 [55]	580	140	ECG
	ECG0006&FetalECGThorax [95]	1800	750	ECG
	Blink [19]	500	510	EEG
	FaceDetection [40]	5890	62	EEG
	SelfRegulation&PzP [27]	201	115	EEG
	ElectricalDevices [60]	8926	96	Sensors
	Trace [60]	100	275	Sensors
	FbDw [27]	3636	500	Sensors
	MotionSenseHAR [75]	966	200	Human Activity
	EMOPath [27]	968	180	Human Activity
	OpenSMILElibrary [123]	3115	315	Human Activity
	Chinatown [20]	20	1	Traffic
	MelbournePedestrian [23]	1194	24	Traffic
	PEMS-SF [23]	267	144	Traffic

Table A.1: `UniTS` pre-training data

Appendix B

Figures

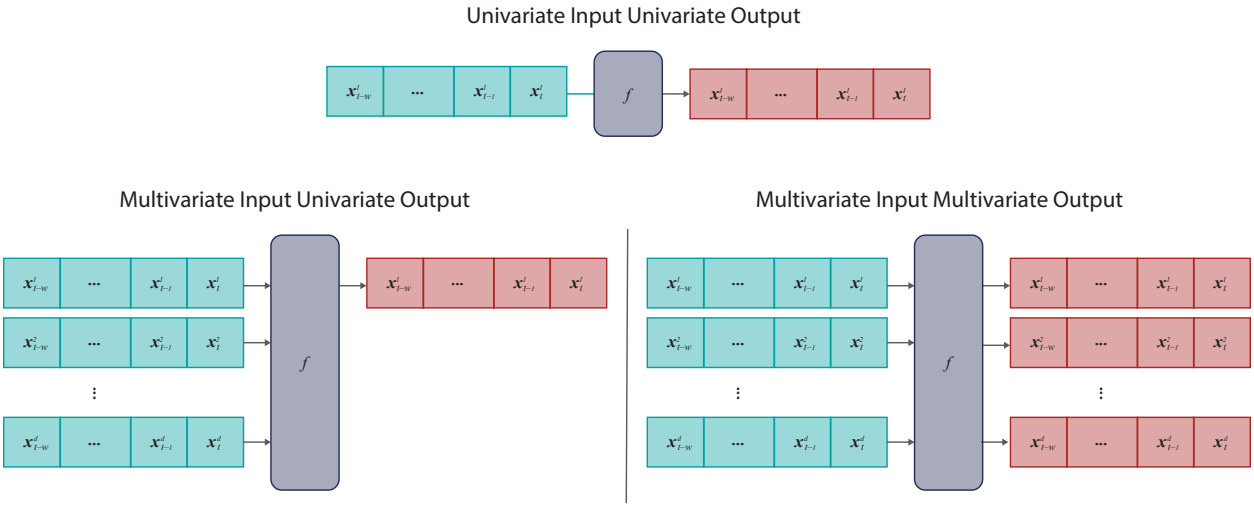
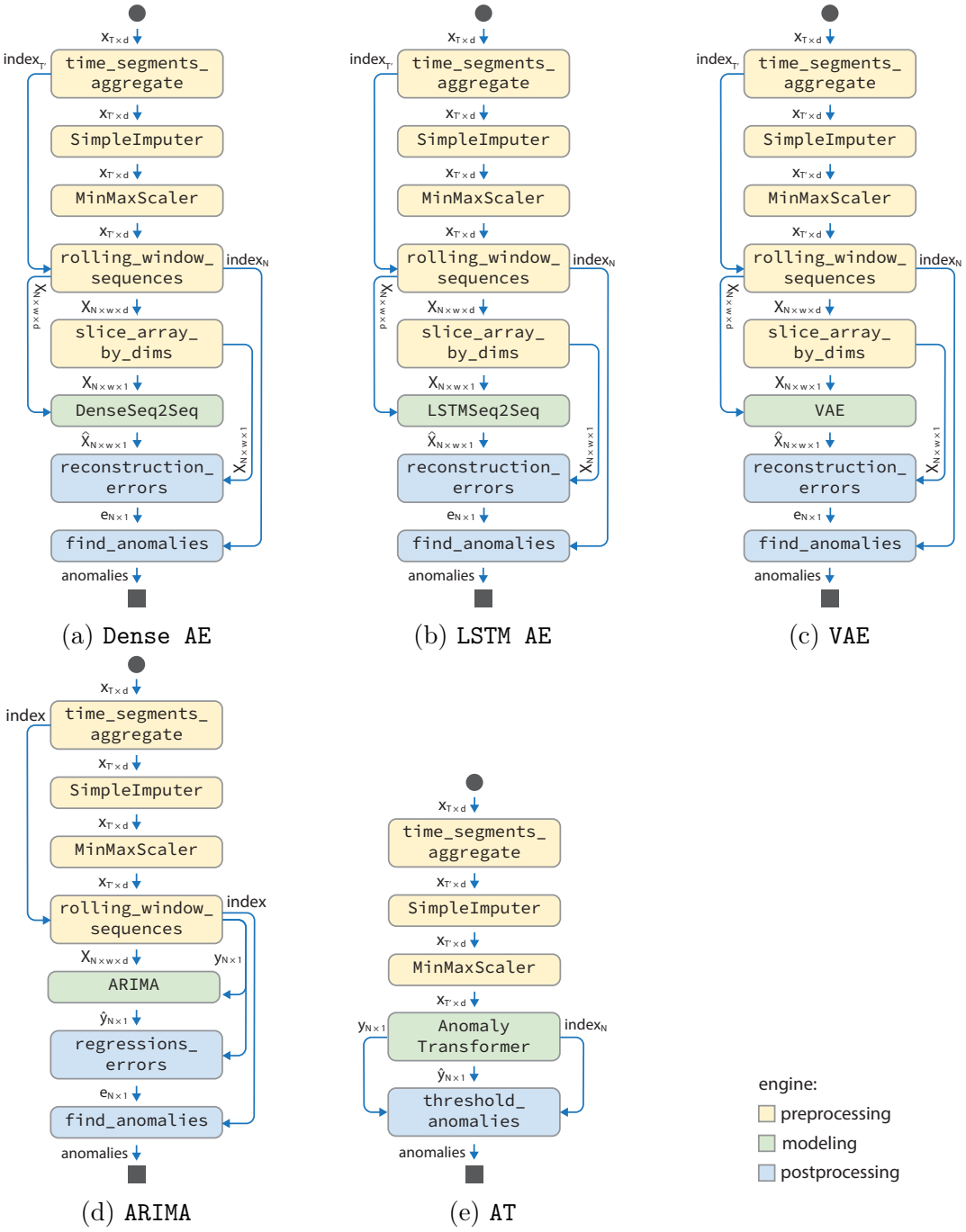


Figure B-1: Model configurations for time series reconstruction.



(a) `Dense AE` (b) `LSTM AE` (c) `VAE`

(d) `ARIMA` (e) `AT`

Figure B-2: Directed acyclic graphs (DAGs) of Orion pipelines. (a) Dense auto-encoder; (b) LSTM auto-encoder; (c) Variational auto-encoder; (d) ARIMA; (e) Anomaly transformer (`AT`).

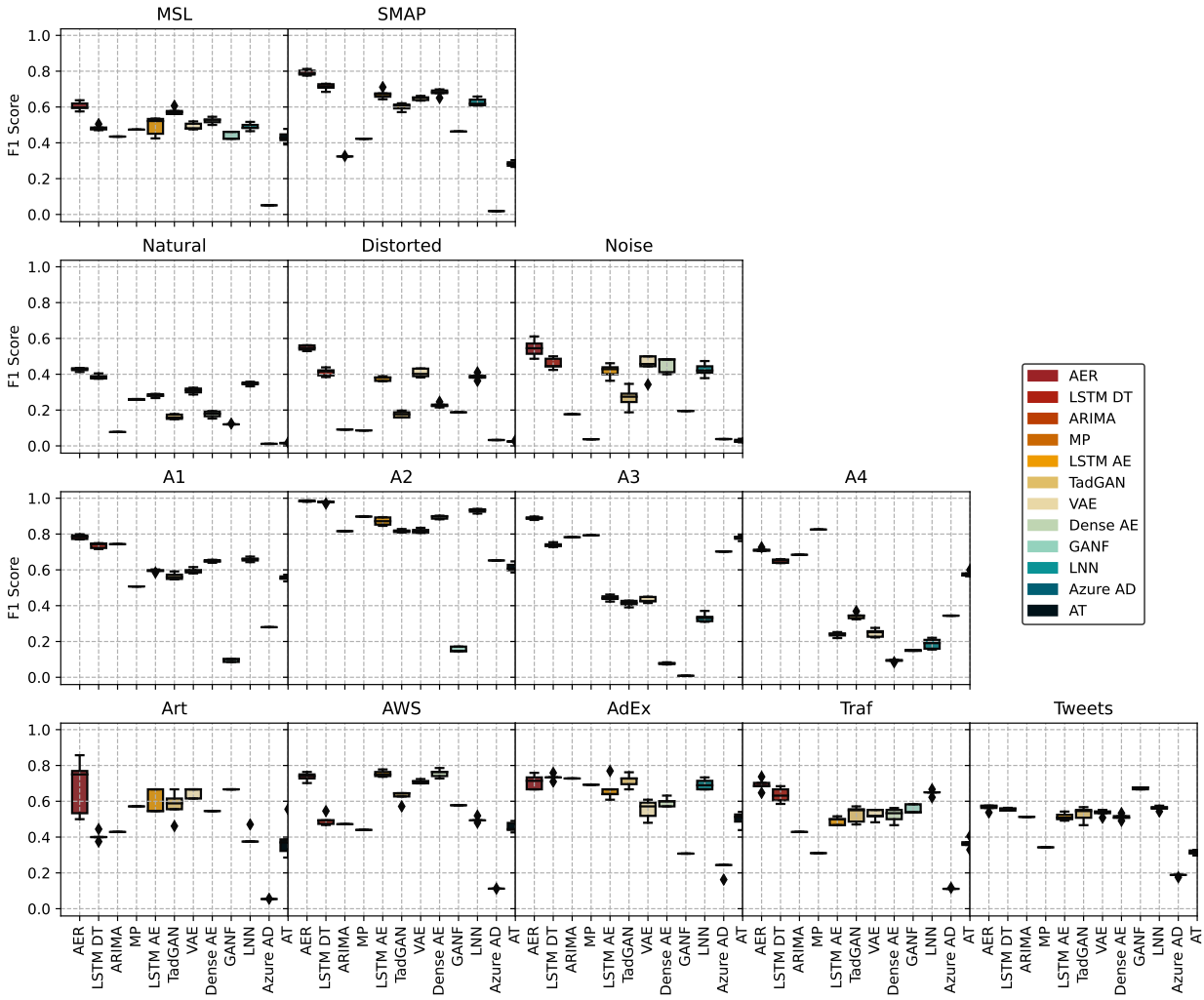


Figure B-3: Distribution of F1 Score per Dataset. This figure is a detailed version of Figure 5-8, showing every dataset separately. On average, `AER` is the highest-scoring pipeline for most datasets, with the exception of AWS, AdEx, and Tweets. A pipeline's performance changes from one dataset to another, indicating that there is no single pipeline that will work perfectly for all datasets. One of our insights here is that point anomalies in A3 & A4 present a challenge for reconstruction-based pipelines such as `LSTM AE`, `TadGAN`, `VAE`, and `Dense AE`.

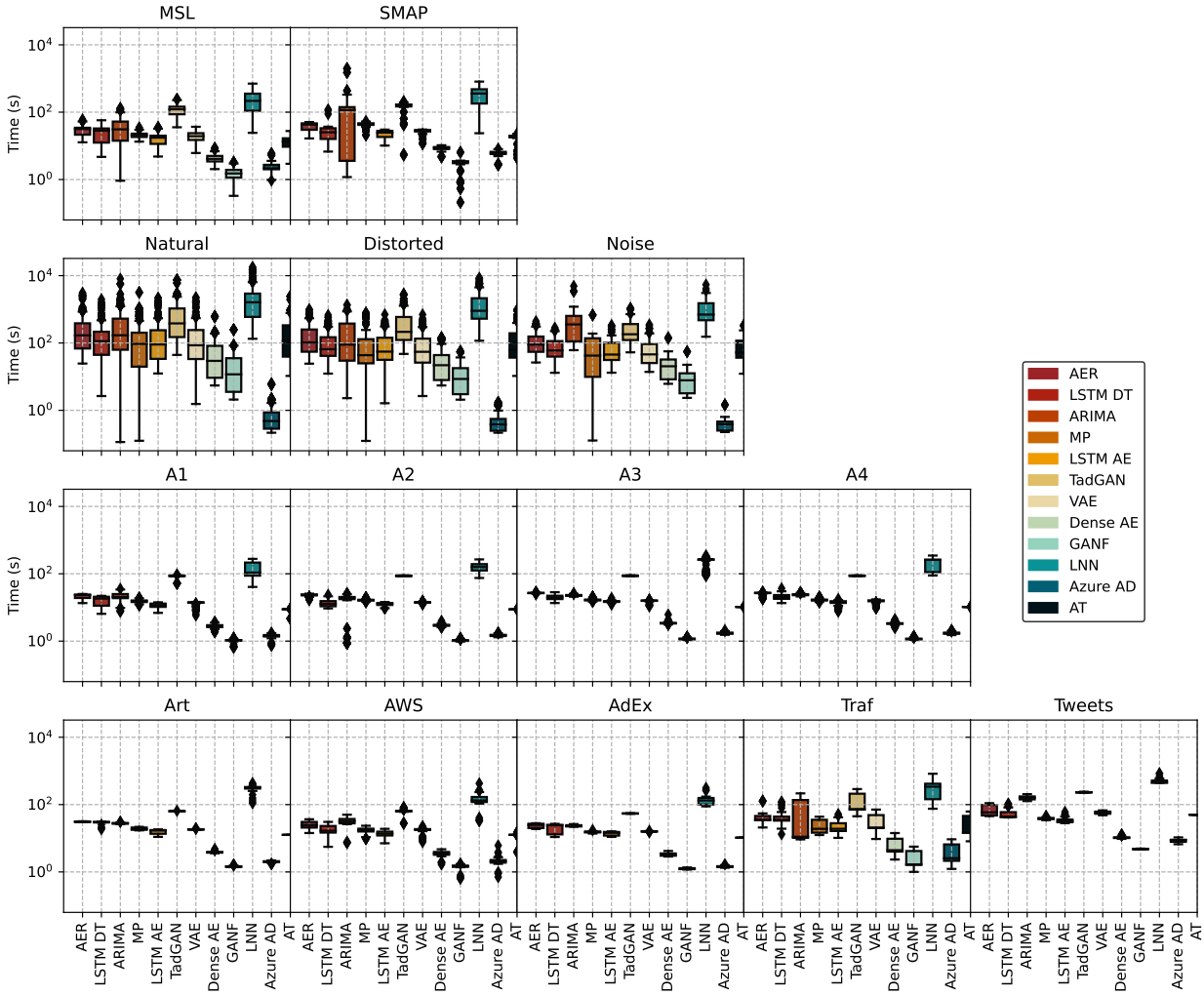


Figure B-4: Runtime (in seconds) per dataset. Runtime is recorded as the time it takes to train a pipeline using `fit` and then run inference using `detect`. Pipeline scalability is important for many end users. `TadGAN` takes minutes to run, while other pipelines finish in seconds. The fastest pipelines are `GANF` and `Azure AD`. `Azure AD` is an inference-only pipeline, and `GANF` is fast to train.

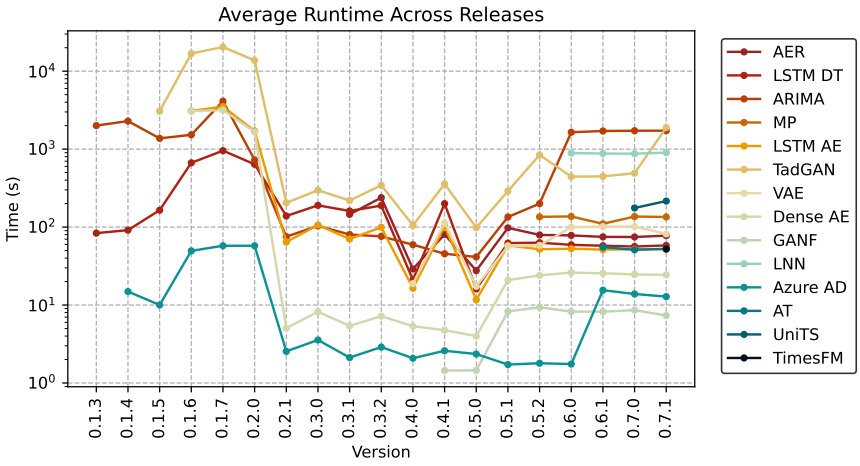


Figure B-5: Average runtimes (in seconds) across Orion versions. Deep learning pipelines with LSTM layers are more sporadic across releases due to the availability of GPU. Pipelines such as `Azure AD` are black box pipelines that run inference alone, which is speedy. In version `0.2.1`, we migrated our benchmark to MIT Supercloud.

Appendix C

Tables

Table C.1: Examples of prompts used in **Prompter** with their respective observed output. $\{x_{1..w}\}$ is a placeholder for 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` [1] or sklearn's `IsolationForest` [2]. (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 start and end, 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 assistant 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**.

Table C.2: (left) Leaderboard showing number of datasets in which each pipeline outperformed ARIMA. (right) Rank of pipelines computed from five independent runs.

Pipeline	Outperform ARIMA [Box and Pierce, 1970]
AER, 2022	13
LSTM DT, 2018	10
LSTM AE, 2016	9
TadGAN, 2020	9
VAE, 2018	9
Dense AE, 2014	9
LNN, 2021	9
GANF, 2022	8
MP, 2016	7
AT, 2022	2
Azure AD, 2019	0

Pipeline	Run #1	#2	#3	#4	#5
AER	1	1	1	1	1
LSTM DT	3	2	2	2	2
LSTM AE	2	5	3	4	4
TadGAN	6	7	5	5	6
VAE	4	3	4	7	5
Dense AE	7	4	6	6	7
LNN	5	6	7	3	3
GANF	8	8	8	8	8
MP	9	9	9	9	9
AT	10	10	10	10	10
Azure AD	11	11	11	11	11

C.1 Benchmark Results

C.1.1 Leaderboard

With every release, we present a leaderboard similar to Table C.2(left). It depicts the number of times each pipeline outperformed ARIMA in F1 Score (maximum # datasets). Specifically, Table C.2(left) is generated from benchmark version 0.5.2. It provides an overall view of how deep learning models perform compared to a classical method such as ARIMA. AER fluctuates between 11 and 12, beating ARIMA on almost every dataset, with the exception of AdEx dataset.

Table C.2(right) shows the rank of each pipeline in 5 different benchmark runs. This rank is calculated from the leaderboard. If two pipelines have the same number of wins, the average F1 score is used as a tiebreaker.

[1] https://numpy.org/doc/stable/reference/generated/numpy.convolve.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

C.1.2 Performance Across Releases

Detailed information on benchmark runs is stored directly in the repository: https://github.com/sintel-dev/Orion/tree/master/benchmark/results. The following tables report the F1 score, precision, and recall metrics for each pipeline across all releases as of today.

- version `0.7.1`, pipelines 14, datasets 12, release date: March 17$^{\text{th}}$ 2025.
- version `0.7.0`, pipelines 13, datasets 12, release data: December 18$^{\text{th}}$ 2024.
- version `0.6.1`, pipelines 12, datasets 12, release date: October 4$^{\text{th}}$ 2024.
- version `0.6.0`, pipelines 11, datasets 12, release date: February 13$^{\text{th}}$ 2024.
- version `0.5.2`, pipelines 10, datasets 12, release date: October 19$^{\text{th}}$ 2023.
- version `0.5.1`, pipelines 9, datasets 12, release date: August 16$^{\text{th}}$ 2023.
- version `0.5.0`, pipelines 9, datasets 11, release date: May 23$^{\text{rd}}$ 2023.
- version `0.4.1`, pipelines 9, datasets 11, release date: January 31$^{\text{st}}$ 2023.
- version `0.4.0`, pipelines 8, datasets 11, release date: November 10$^{\text{th}}$ 2022.
- version `0.3.2`, pipelines 7, datasets 11, release date: July 4$^{\text{th}}$ 2022.
- version `0.3.1`, pipelines 7, datasets 11, release date: April 26$^{\text{th}}$ 2022.
- version `0.3.0`, pipelines 6, datasets 11, release date: March 31$^{\text{st}}$ 2022.
- version `0.2.1`, pipelines 6, datasets 11, release date: February 18$^{\text{th}}$ 2022.
- version `0.2.0`, pipelines 6, datasets 11, release date: October 11$^{\text{th}}$ 2021.
- version `0.1.7`, pipelines 6, datasets 11, release date: May 4$^{\text{th}}$ 2021.
- version `0.1.6`, pipelines 6, datasets 11, release date: March 8$^{\text{th}}$ 2021.
- version `0.1.5`, pipelines 4, datasets 11, release date: December 25$^{\text{th}}$ 2020.

- version `0.1.4`, pipelines 3, datasets 11, release date: October 16$^{\text{th}}$ 2020.
- version `0.1.3`, pipelines 2, datasets 11, release date: September 29$^{\text{th}}$ 2020.

	NASA		UCR			Yahoo S5				NAB				
Pipeline	MSL	SMAP	Natural	Distorted	Noise	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score														
AER	0.605±0.02	0.792±0.02	0.427±0.01	0.547±0.01	0.546±0.05	0.784±0.01	0.984±0.00	0.889±0.01	0.712±0.01	0.682±0.16	0.737±0.02	0.708±0.04	0.691±0.03	0.565±0.02
LSTM DT	0.483±0.01	0.713±0.02	0.386±0.01	0.411±0.02	0.460±0.03	0.736±0.02	0.978±0.00	0.738±0.01	0.649±0.01	0.404±0.03	0.495±0.03	0.734±0.02	0.634±0.04	0.556±0.01
ARIMA	0.435±0.00	0.324±0.00	0.077±0.00	0.092±0.00	0.176±0.00	0.744±0.00	0.816±0.00	0.782±0.00	0.684±0.00	0.429±0.00	0.472±0.00	0.727±0.00	0.429±0.00	0.513±0.00
MP	0.474±0.00	0.423±0.00	0.259±0.00	0.086±0.00	0.037±0.00	0.507±0.00	0.897±0.00	0.793±0.00	0.825±0.00	0.571±0.00	0.440±0.00	0.692±0.00	0.310±0.00	0.343±0.00
LSTM AE	0.493±0.05	0.672±0.03	0.281±0.01	0.375±0.01	0.419±0.04	0.595±0.01	0.871±0.02	0.444±0.01	0.238±0.01	0.594±0.07	0.754±0.02	0.665±0.06	0.490±0.02	0.513±0.02
TadGAN	0.576±0.02	0.601±0.02	0.162±0.01	0.176±0.02	0.269±0.06	0.564±0.02	0.817±0.01	0.413±0.02	0.342±0.02	0.577±0.08	0.627±0.03	0.716±0.04	0.527±0.05	0.528±0.04
VAE	0.492±0.02	0.649±0.01	0.308±0.02	0.407±0.02	0.449±0.06	0.595±0.01	0.817±0.01	0.433±0.02	0.247±0.02	0.646±0.03	0.710±0.01	0.554±0.05	0.524±0.03	0.536±0.02
Dense AE	0.524±0.02	0.680±0.02	0.177±0.02	0.228±0.01	0.452±0.04	0.649±0.01	0.894±0.01	0.077±0.00	0.093±0.01	0.545±0.00	0.756±0.02	0.595±0.02	0.523±0.04	0.512±0.02
GANF	0.438±0.02	0.463±0.00	0.121±0.00	0.188±0.00	0.195±0.00	0.095±0.01	0.156±0.01	0.008±0.00	0.150±0.00	0.667±0.00	0.578±0.00	0.308±0.00	0.556±0.02	0.673±0.01
LNN	0.490±0.02	0.627±0.02	0.347±0.01	0.386±0.02	0.425±0.04	0.658±0.01	0.930±0.01	0.333±0.02	0.187±0.03	0.394±0.04	0.496±0.01	0.694±0.03	0.648±0.02	0.562±0.01
Azure AD	0.051±0.00	0.019±0.00	0.011±0.00	0.033±0.00	0.038±0.00	0.280±0.00	0.653±0.00	0.702±0.00	0.344±0.00	0.054±0.00	0.112±0.00	0.228±0.04	0.112±0.00	0.186±0.01
AT	0.431±0.03	0.283±0.02	0.016±0.00	0.026±0.00	0.028±0.01	0.555±0.01	0.613±0.02	0.779±0.01	0.577±0.01	0.384±0.10	0.456±0.03	0.501±0.04	0.365±0.03	0.315±0.01
Precision														
AER	0.601±0.05	0.832±0.02	0.338±0.01	0.478±0.02	0.502±0.05	0.809±0.02	0.983±0.01	0.992±0.00	0.927±0.01	0.602±0.13	0.804±0.04	0.566±0.04	0.562±0.02	0.572±0.03
LSTM DT	0.370±0.01	0.655±0.02	0.327±0.01	0.345±0.03	0.418±0.02	0.690±0.02	0.972±0.01	0.988±0.01	0.902±0.01	0.327±0.01	0.417±0.05	0.580±0.02	0.491±0.04	0.514±0.02
ARIMA	0.455±0.00	0.307±0.00	0.085±0.00	0.115±0.00	0.167±0.00	0.684±0.00	0.772±0.00	0.998±0.00	0.955±0.00	0.375±0.00	0.405±0.00	0.727±0.00	0.429±0.00	0.444±0.00
MP	0.346±0.00	0.291±0.00	0.217±0.00	0.047±0.00	0.019±0.00	0.448±0.00	0.824±0.00	0.952±0.00	0.946±0.00	0.500±0.00	0.314±0.00	0.600±0.00	0.205±0.00	0.324±0.00
LSTM AE	0.489±0.07	0.638±0.04	0.192±0.01	0.313±0.02	0.343±0.04	0.625±0.00	0.835±0.04	0.944±0.01	0.672±0.01	0.627±0.04	0.827±0.04	0.602±0.04	0.447±0.01	0.571±0.03
TadGAN	0.493±0.02	0.531±0.03	0.115±0.01	0.130±0.01	0.195±0.05	0.636±0.02	0.809±0.01	0.739±0.03	0.586±0.02	0.485±0.07	0.604±0.05	0.707±0.07	0.429±0.04	0.559±0.05
VAE	0.466±0.02	0.595±0.01	0.221±0.01	0.352±0.03	0.407±0.06	0.584±0.01	0.727±0.02	0.842±0.01	0.653±0.03	0.629±0.05	0.720±0.02	0.484±0.06	0.508±0.04	0.608±0.02
Dense AE	0.573±0.02	0.724±0.02	0.140±0.02	0.227±0.01	0.487±0.07	0.726±0.01	0.954±0.01	0.964±0.01	0.553±0.01	0.600±0.00	0.832±0.02	0.657±0.06	0.483±0.04	0.586±0.01
GANF	0.712±0.03	0.786±0.00	0.084±0.00	0.159±0.00	0.160±0.00	0.286±0.00	0.282±0.02	1.000±0.00	0.977±0.01	1.000±0.00	0.867±0.00	1.000±0.00	0.630±0.06	0.650±0.01
LNN	0.414±0.02	0.537±0.02	0.270±0.01	0.310±0.02	0.365±0.04	0.639±0.01	0.881±0.02	0.956±0.02	0.600±0.05	0.313±0.03	0.405±0.01	0.555±0.03	0.490±0.02	0.524±0.01
Azure AD	0.026±0.00	0.009±0.00	0.006±0.00	0.019±0.00	0.021±0.00	0.167±0.00	0.484±0.00	0.542±0.00	0.217±0.00	0.028±0.00	0.060±0.00	0.131±0.02	0.059±0.00	0.105±0.00
AT	0.282±0.03	0.170±0.01	0.008±0.00	0.013±0.00	0.014±0.00	0.497±0.01	0.494±0.03	0.85±0.01	0.758±0.01	0.253±0.10	0.325±0.02	0.352±0.03	0.229±0.02	0.188±0.01
Recall														
AER	0.611±0.02	0.755±0.01	0.577±0.01	0.641±0.02	0.600±0.06	0.761±0.01	0.985±0.00	0.805±0.01	0.578±0.01	0.800±0.22	0.680±0.02	0.945±0.05	0.900±0.08	0.558±0.02
LSTM DT	0.694±0.02	0.782±0.02	0.472±0.01	0.509±0.02	0.512±0.05	0.788±0.02	0.984±0.00	0.590±0.01	0.507±0.01	0.533±0.07	0.613±0.02	1.000±0.00	0.900±0.04	0.606±0.02
ARIMA	0.417±0.00	0.343±0.00	0.070±0.00	0.076±0.00	0.188±0.00	0.815±0.00	0.865±0.00	0.643±0.00	0.533±0.00	0.500±0.00	0.567±0.00	0.727±0.00	0.429±0.00	0.606±0.00
MP	0.750±0.00	0.776±0.00	0.320±0.00	0.500±0.00	0.625±0.00	0.584±0.00	0.985±0.00	0.679±0.00	0.732±0.00	0.667±0.00	0.733±0.00	0.818±0.00	0.643±0.00	0.364±0.00
LSTM AE	0.500±0.04	0.710±0.01	0.525±0.01	0.470±0.01	0.538±0.03	0.569±0.01	0.912±0.01	0.290±0.01	0.145±0.01	0.567±0.09	0.693±0.01	0.745±0.10	0.543±0.04	0.467±0.02
TadGAN	0.694±0.05	0.696±0.04	0.273±0.02	0.270±0.02	0.438±0.08	0.507±0.02	0.824±0.01	0.287±0.01	0.242±0.01	0.733±0.15	0.653±0.04	0.727±0.00	0.686±0.08	0.503±0.06
VAE	0.522±0.02	0.713±0.02	0.511±0.02	0.485±0.02	0.500±0.08	0.608±0.02	0.933±0.01	0.291±0.01	0.152±0.02	0.667±0.00	0.700±0.00	0.655±0.08	0.543±0.04	0.479±0.01
Dense AE	0.483±0.02	0.642±0.02	0.239±0.02	0.228±0.02	0.425±0.03	0.587±0.01	0.841±0.01	0.040±0.00	0.051±0.00	0.500±0.00	0.693±0.03	0.545±0.00	0.571±0.05	0.455±0.02
GANF	0.317±0.02	0.328±0.00	0.217±0.00	0.228±0.00	0.250±0.00	0.057±0.01	0.108±0.01	0.002±0.00	0.081±0.00	0.500±0.00	0.433±0.00	0.182±0.00	0.500±0.00	0.697±0.00
LNN	0.600±0.03	0.755±0.02	0.486±0.02	0.513±0.01	0.512±0.05	0.679±0.01	0.985±0.00	0.201±0.02	0.111±0.02	0.533±0.07	0.640±0.01	0.927±0.04	0.957±0.04	0.606±0.03
Azure AD	0.806±0.00	0.940±0.00	0.197±0.00	0.130±0.00	0.250±0.00	0.848±0.00	1.000±0.00	0.998±0.00	0.837±0.00	1.000±0.00	0.833±0.00	0.909±0.00	1.000±0.00	0.818±0.00
AT	0.928±0.03	0.854±0.02	0.690±0.03	0.750±0.03	0.762±0.05	0.628±0.02	0.809±0.02	0.718±0.02	0.465±0.02	0.867±0.07	0.767±0.04	0.873±0.05	0.914±0.03	0.958±0.05

Table C.3: Benchmark Results

Table C.4: Benchmark Summary Results Version `0.7.1`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score												
`AER`	0.657	0.764	0.468	0.782	0.978	0.893	0.716	0.750	0.702	0.733	0.611	0.606
`LSTM DT`	0.455	0.732	0.414	0.740	0.978	0.744	0.638	0.400	0.537	0.759	0.611	0.588
`ARIMA`	0.525	0.411	0.153	0.728	0.856	0.797	0.686	0.308	0.382	0.727	0.467	0.514
`MP`	0.474	0.423	0.051	0.507	0.897	0.793	0.825	0.571	0.440	0.692	0.305	0.343
`LSTM AE`	0.548	0.652	0.311	0.613	0.885	0.460	0.245	0.545	0.750	0.720	0.533	0.448
`TadGAN`	0.578	0.573	0.175	0.619	0.827	0.402	0.345	0.500	0.710	0.692	0.486	0.588
`VAE`	0.519	0.667	0.325	0.565	0.813	0.424	0.246	0.615	0.700	0.615	0.538	0.542
`Dense AE`	0.537	0.677	0.213	0.656	0.894	0.086	0.088	0.545	0.778	0.556	0.600	0.500
`GANF`	0.462	0.463	0.144	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`LNN`	0.489	0.646	0.367	0.668	0.927	0.339	0.191	0.375	0.500	0.643	0.591	0.563
`Azure AD`	0.051	0.019	0.004	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
`AT`	0.389	0.270	0.020	0.583	0.603	0.771	0.569	0.387	0.440	0.500	0.377	0.306
Precision												
`AER`	0.676	0.839	0.382	0.821	0.970	0.997	0.931	0.600	0.741	0.579	0.500	0.606
`LSTM DT`	0.354	0.693	0.352	0.689	0.970	0.988	0.891	0.333	0.486	0.611	0.500	0.571
`ARIMA`	0.477	0.303	0.144	0.670	0.769	0.998	0.955	0.286	0.342	0.727	0.438	0.486
`MP`	0.346	0.291	0.027	0.448	0.824	0.952	0.946	0.500	0.314	0.600	0.200	0.324
`LSTM AE`	0.541	0.622	0.228	0.658	0.852	0.941	0.653	0.600	0.808	0.643	0.500	0.520
`TadGAN`	0.511	0.518	0.125	0.697	0.829	0.764	0.569	0.500	0.688	0.600	0.391	0.571
`VAE`	0.488	0.635	0.246	0.557	0.725	0.837	0.660	0.571	0.700	0.533	0.583	0.615
`Dense AE`	0.581	0.717	0.183	0.739	0.976	0.977	0.541	0.600	0.875	0.714	0.562	0.556
`GANF`	0.750	0.786	0.106	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`LNN`	0.397	0.546	0.288	0.642	0.876	0.942	0.590	0.300	0.413	0.529	0.433	0.526
`Azure AD`	0.026	0.009	0.002	0.167	0.484	0.542	0.217	0.029	0.060	0.089	0.062	0.099
`AT`	0.245	0.162	0.010	0.513	0.492	0.847	0.753	0.240	0.314	0.345	0.236	0.182
Recall												
`AER`	0.639	0.701	0.604	0.747	0.985	0.808	0.582	1.000	0.667	1.000	0.786	0.606
`LSTM DT`	0.639	0.776	0.504	0.798	0.985	0.596	0.497	0.500	0.600	1.000	0.786	0.606
`ARIMA`	0.583	0.642	0.164	0.798	0.965	0.663	0.535	0.333	0.433	0.727	0.500	0.545
`MP`	0.750	0.776	0.432	0.584	0.985	0.679	0.732	0.667	0.733	0.818	0.643	0.364
`LSTM AE`	0.556	0.687	0.488	0.573	0.920	0.305	0.151	0.500	0.700	0.818	0.571	0.394
`TadGAN`	0.667	0.642	0.296	0.556	0.825	0.273	0.248	0.500	0.733	0.818	0.643	0.606
`VAE`	0.556	0.701	0.480	0.573	0.925	0.284	0.151	0.667	0.700	0.727	0.500	0.485
`Dense AE`	0.500	0.642	0.256	0.590	0.825	0.045	0.048	0.500	0.700	0.455	0.643	0.455
`GANF`	0.333	0.328	0.224	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`LNN`	0.639	0.791	0.504	0.697	0.985	0.207	0.114	0.500	0.633	0.818	0.929	0.606
`Azure AD`	0.806	0.940	0.820	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818
`AT`	0.944	0.821	0.700	0.674	0.780	0.708	0.457	1.000	0.733	0.909	0.929	0.970

Table C.5: Benchmark Summary Results Version `0.7.0`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score												
`AER`	0.583	0.760	0.482	0.791	0.987	0.892	0.716	0.571	0.727	0.690	0.686	0.585
`LSTM DT`	0.476	0.712	0.387	0.741	0.980	0.737	0.647	0.400	0.480	0.759	0.550	0.551
`ARIMA`	0.525	0.411	0.153	0.728	0.856	0.797	0.686	0.308	0.382	0.727	0.467	0.514
`MP`	0.474	0.423	0.051	0.507	0.897	0.793	0.825	0.571	0.440	0.692	0.305	0.343
`LSTM AE`	0.514	0.681	0.339	0.616	0.877	0.432	0.213	0.545	0.750	0.640	0.500	0.448
`TadGAN`	0.602	0.588	0.162	0.564	0.842	0.417	0.326	0.556	0.627	0.640	0.563	0.576
`VAE`	0.519	0.653	0.332	0.560	0.811	0.412	0.229	0.667	0.700	0.615	0.500	0.567
`Dense AE`	0.545	0.688	0.204	0.667	0.894	0.091	0.093	0.545	0.764	0.636	0.483	0.517
`GANF`	0.462	0.463	0.147	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`LNN`	0.478	0.638	0.345	0.679	0.936	0.315	0.191	0.375	0.494	0.690	0.651	0.563
`Azure AD`	0.051	0.019	0.004	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
`AT`	0.415	0.279	0.018	0.566	0.608	0.783	0.551	0.429	0.441	0.526	0.371	0.292
Precision												
`AER`	0.583	0.790	0.402	0.819	0.990	0.995	0.928	0.500	0.800	0.556	0.571	0.594
`LSTM DT`	0.362	0.658	0.336	0.700	0.980	0.987	0.898	0.333	0.400	0.611	0.423	0.528
`ARIMA`	0.477	0.303	0.144	0.670	0.769	0.998	0.955	0.286	0.342	0.727	0.438	0.486
`MP`	0.346	0.291	0.027	0.448	0.824	0.952	0.946	0.500	0.314	0.600	0.200	0.324
`LSTM AE`	0.500	0.636	0.247	0.639	0.847	0.943	0.637	0.600	0.808	0.571	0.444	0.520
`TadGAN`	0.532	0.505	0.118	0.622	0.833	0.769	0.542	0.417	0.568	0.571	0.500	0.576
`VAE`	0.488	0.600	0.254	0.559	0.711	0.839	0.648	0.667	0.700	0.533	0.500	0.630
`Dense AE`	0.600	0.721	0.173	0.757	0.955	0.957	0.575	0.600	0.840	0.636	0.467	0.600
`GANF`	0.750	0.786	0.111	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`LNN`	0.393	0.548	0.268	0.663	0.891	0.913	0.597	0.300	0.404	0.556	0.483	0.526
`Azure AD`	0.026	0.009	0.002	0.167	0.484	0.542	0.217	0.029	0.060	0.089	0.062	0.099
`AT`	0.266	0.171	0.009	0.511	0.486	0.857	0.741	0.273	0.295	0.370	0.232	0.173
Recall												
`AER`	0.583	0.731	0.600	0.764	0.985	0.808	0.583	0.667	0.667	0.909	0.857	0.576
`LSTM DT`	0.694	0.776	0.456	0.787	0.980	0.588	0.505	0.500	0.600	1.000	0.786	0.576
`ARIMA`	0.583	0.642	0.164	0.798	0.965	0.663	0.535	0.333	0.433	0.727	0.500	0.545
`MP`	0.750	0.776	0.432	0.584	0.985	0.679	0.732	0.667	0.733	0.818	0.643	0.364
`LSTM AE`	0.528	0.731	0.540	0.596	0.910	0.280	0.128	0.500	0.700	0.727	0.571	0.394
`TadGAN`	0.694	0.701	0.260	0.517	0.850	0.286	0.234	0.833	0.700	0.727	0.643	0.576
`VAE`	0.556	0.716	0.480	0.562	0.945	0.273	0.139	0.667	0.700	0.727	0.500	0.515
`Dense AE`	0.500	0.657	0.248	0.596	0.840	0.048	0.050	0.500	0.700	0.636	0.500	0.455
`GANF`	0.333	0.328	0.220	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`LNN`	0.611	0.761	0.484	0.697	0.985	0.191	0.114	0.500	0.633	0.909	1.000	0.606
`Azure AD`	0.806	0.940	0.828	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818
`AT`	0.944	0.761	0.704	0.635	0.810	0.721	0.438	1.000	0.867	0.909	0.929	0.939

Table C.6: Benchmark Summary Results Version `0.6.1`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score												
`AER`	0.533	0.781	0.489	0.784	0.987	0.878	0.712	0.714	0.727	0.690	0.703	0.562
`LSTM DT`	0.466	0.694	0.390	0.735	0.980	0.743	0.637	0.400	0.507	0.714	0.585	0.603
`ARIMA`	0.525	0.411	0.153	0.728	0.856	0.797	0.686	0.308	0.382	0.727	0.467	0.514
`MP`	0.474	0.423	0.051	0.507	0.897	0.793	0.825	0.571	0.440	0.692	0.305	0.343
`LSTM AE`	0.462	0.662	0.330	0.600	0.864	0.444	0.247	0.667	0.737	0.500	0.467	0.508
`TadGAN`	0.568	0.610	0.177	0.593	0.805	0.377	0.308	0.667	0.667	0.696	0.516	0.531
`VAE`	0.538	0.627	0.354	0.570	0.809	0.427	0.244	0.667	0.712	0.480	0.483	0.508
`Dense AE`	0.493	0.688	0.204	0.644	0.891	0.084	0.086	0.545	0.778	0.600	0.563	0.533
`GANF`	0.462	0.463	0.143	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`LNN`	0.477	0.654	0.363	0.661	0.925	0.305	0.197	0.400	0.506	0.710	0.636	0.592
`Azure AD`	0.051	0.019	0.004	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
`AT`	0.449	0.303	0.021	0.583	0.617	0.772	0.576	0.385	0.423	0.474	0.338	0.315
Precision												
`AER`	0.513	0.820	0.411	0.805	0.990	0.995	0.922	0.625	0.800	0.556	0.565	0.581
`LSTM DT`	0.358	0.638	0.329	0.690	0.975	0.988	0.896	0.333	0.439	0.588	0.444	0.550
`ARIMA`	0.477	0.303	0.144	0.670	0.769	0.998	0.955	0.286	0.342	0.727	0.438	0.486
`MP`	0.346	0.291	0.027	0.448	0.824	0.952	0.946	0.500	0.314	0.600	0.200	0.324
`LSTM AE`	0.429	0.627	0.242	0.630	0.814	0.941	0.681	0.667	0.778	0.462	0.438	0.577
`TadGAN`	0.481	0.540	0.130	0.629	0.777	0.716	0.531	0.556	0.636	0.667	0.471	0.548
`VAE`	0.599	0.558	0.271	0.556	0.709	0.856	0.640	0.667	0.724	0.429	0.467	0.577
`Dense AE`	0.515	0.741	0.171	0.725	0.949	0.976	0.534	0.600	0.875	0.667	0.500	0.593
`GANF`	0.750	0.786	0.105	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`LNN`	0.404	0.573	0.285	0.659	0.872	0.929	0.620	0.333	0.408	0.550	0.467	0.553
`Azure AD`	0.026	0.009	0.002	0.167	0.484	0.542	0.217	0.029	0.060	0.089	0.062	0.099
`AT`	0.297	0.193	0.011	0.517	0.498	0.855	0.747	0.250	0.297	0.333	0.206	0.189
Recall												
`AER`	0.556	0.746	0.604	0.764	0.985	0.786	0.580	0.833	0.667	0.909	0.929	0.545
`LSTM DT`	0.667	0.761	0.480	0.787	0.985	0.595	0.495	0.500	0.600	0.909	0.857	0.667
`ARIMA`	0.583	0.642	0.164	0.798	0.965	0.663	0.535	0.333	0.433	0.727	0.500	0.545
`MP`	0.750	0.776	0.432	0.584	0.985	0.679	0.732	0.667	0.733	0.818	0.643	0.364
`LSTM AE`	0.500	0.701	0.520	0.573	0.920	0.291	0.151	0.667	0.700	0.545	0.500	0.455
`TadGAN`	0.694	0.701	0.276	0.562	0.835	0.256	0.217	0.833	0.700	0.727	0.571	0.515
`VAE`	0.583	0.716	0.512	0.584	0.940	0.284	0.151	0.667	0.700	0.545	0.500	0.455
`Dense AE`	0.472	0.642	0.252	0.579	0.840	0.044	0.047	0.500	0.700	0.545	0.643	0.485
`GANF`	0.333	0.328	0.224	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`LNN`	0.583	0.761	0.500	0.663	0.985	0.182	0.117	0.500	0.667	1.000	1.000	0.636
`Azure AD`	0.806	0.940	0.824	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818
`AT`	0.917	0.701	0.704	0.669	0.810	0.704	0.469	0.833	0.733	0.818	0.929	0.939

Table C.7: Benchmark Summary Results Version `0.6.0`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score												
`AER`	0.587	0.819	0.476	0.799	0.987	0.892	0.709	0.714	0.741	0.690	0.703	0.638
`LSTM DT`	0.471	0.726	0.393	0.728	0.985	0.744	0.646	0.400	0.468	0.786	0.585	0.603
`ARIMA`	0.525	0.411	0.153	0.728	0.856	0.797	0.686	0.308	0.382	0.727	0.467	0.514
`MP`	0.474	0.423	0.051	0.507	0.897	0.793	0.825	0.571	0.440	0.692	0.305	0.343
`LSTM AE`	0.545	0.662	0.327	0.595	0.867	0.466	0.239	0.667	0.741	0.500	0.500	0.475
`TadGAN`	0.560	0.605	0.170	0.578	0.817	0.416	0.340	0.500	0.623	0.818	0.452	0.554
`VAE`	0.494	0.613	0.324	0.592	0.803	0.438	0.23	0.667	0.689	0.583	0.483	0.533
`Dense AE`	0.559	0.692	0.207	0.667	0.892	0.07	0.101	0.545	0.764	0.600	0.563	0.508
`GANF`	0.462	0.463	0.147	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`LNN`	0.517	0.618	0.362	0.652	0.938	0.331	0.191	0.375	0.481	0.714	0.667	0.575
`Azure AD`	0.051	0.019	0.015	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
Precision												
`AER`	0.564	0.867	0.395	0.830	0.990	0.993	0.917	0.625	0.833	0.556	0.565	0.611
`LSTM DT`	0.364	0.671	0.335	0.665	0.985	0.986	0.905	0.333	0.383	0.647	0.444	0.550
`ARIMA`	0.477	0.303	0.144	0.670	0.769	0.998	0.955	0.286	0.342	0.727	0.438	0.486
`MP`	0.346	0.291	0.027	0.448	0.824	0.952	0.946	0.500	0.314	0.600	0.200	0.324
`LSTM AE`	0.512	0.615	0.239	0.633	0.827	0.948	0.649	0.667	0.833	0.462	0.444	0.538
`TadGAN`	0.538	0.516	0.124	0.629	0.845	0.762	0.588	0.400	0.613	0.818	0.412	0.562
`VAE`	0.444	0.600	0.243	0.574	0.701	0.852	0.646	0.667	0.677	0.538	0.467	0.593
`Dense AE`	0.594	0.714	0.178	0.748	0.939	0.944	0.590	0.600	0.840	0.667	0.500	0.577
`GANF`	0.750	0.786	0.111	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`LNN`	0.434	0.520	0.288	0.632	0.895	0.954	0.590	0.300	0.388	0.588	0.500	0.525
`Azure AD`	0.026	0.009	0.008	0.167	0.484	0.542	0.217	0.029	0.06	0.089	0.062	0.099
Recall												
`AER`	0.611	0.776	0.600	0.770	0.985	0.809	0.578	0.833	0.667	0.909	0.929	0.667
`LSTM DT`	0.667	0.791	0.476	0.803	0.985	0.597	0.502	0.500	0.600	1.000	0.857	0.667
`ARIMA`	0.583	0.642	0.164	0.798	0.965	0.663	0.535	0.333	0.433	0.727	0.500	0.545
`MP`	0.750	0.776	0.432	0.584	0.985	0.679	0.732	0.667	0.733	0.818	0.643	0.364
`LSTM AE`	0.583	0.716	0.516	0.562	0.910	0.309	0.146	0.667	0.667	0.545	0.571	0.424
`TadGAN`	0.583	0.731	0.272	0.534	0.790	0.286	0.240	0.667	0.633	0.818	0.500	0.545
`VAE`	0.556	0.627	0.488	0.612	0.940	0.295	0.140	0.667	0.700	0.636	0.500	0.485
`Dense AE`	0.528	0.672	0.248	0.601	0.850	0.036	0.055	0.500	0.700	0.545	0.643	0.455
`GANF`	0.333	0.328	0.220	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`LNN`	0.639	0.761	0.488	0.674	0.985	0.200	0.114	0.500	0.633	0.909	1.000	0.636
`Azure AD`	0.806	0.940	0.176	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table C.8: Benchmark Summary Results Version `0.5.2`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score												
`AER`	0.587	0.775	0.474	0.780	0.987	0.869	0.686	0.769	0.750	0.733	0.611	0.581
`LSTM DT`	0.485	0.707	0.417	0.724	0.987	0.744	0.644	0.400	0.494	0.759	0.667	0.600
`ARIMA`	0.435	0.326	0.090	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`MP`	0.474	0.423	0.051	0.507	0.897	0.793	0.825	0.571	0.440	0.692	0.305	0.343
`LSTM AE`	0.479	0.662	0.332	0.619	0.874	0.460	0.227	0.667	0.750	0.615	0.471	0.533
`TadGAN`	0.568	0.590	0.173	0.552	0.806	0.408	0.321	0.571	0.603	0.583	0.529	0.606
`VAE`	0.486	0.649	0.339	0.556	0.817	0.415	0.236	0.462	0.737	0.538	0.483	0.533
`Dense AE`	0.537	0.641	0.202	0.640	0.885	0.078	0.102	0.545	0.800	0.632	0.500	0.508
`GANF`	0.462	0.463	0.147	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`Azure AD`	0.051	0.019	0.015	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
Precision												
`AER`	0.564	0.806	0.385	0.816	0.990	0.992	0.920	0.714	0.808	0.579	0.500	0.621
`LSTM DT`	0.373	0.650	0.352	0.680	0.990	0.988	0.892	0.333	0.404	0.611	0.545	0.568
`ARIMA`	0.455	0.311	0.102	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`MP`	0.346	0.291	0.027	0.448	0.824	0.952	0.946	0.500	0.314	0.600	0.200	0.324
`LSTM AE`	0.486	0.639	0.245	0.652	0.833	0.932	0.675	0.667	0.808	0.533	0.400	0.593
`TadGAN`	0.511	0.517	0.125	0.624	0.758	0.736	0.532	0.500	0.576	0.538	0.450	0.606
`VAE`	0.474	0.583	0.259	0.540	0.723	0.857	0.619	0.429	0.778	0.467	0.467	0.593
`Dense AE`	0.581	0.672	0.172	0.715	0.949	0.974	0.566	0.600	0.880	0.750	0.500	0.577
`GANF`	0.750	0.786	0.111	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`Azure AD`	0.026	0.009	0.008	0.167	0.484	0.542	0.217	0.029	0.060	0.089	0.062	0.099
Recall												
`AER`	0.611	0.746	0.616	0.747	0.985	0.773	0.547	0.833	0.700	1.000	0.786	0.545
`LSTM DT`	0.694	0.776	0.512	0.775	0.985	0.596	0.504	0.500	0.633	1.000	0.857	0.636
`ARIMA`	0.417	0.343	0.080	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`MP`	0.750	0.776	0.432	0.584	0.985	0.679	0.732	0.667	0.733	0.818	0.643	0.364
`LSTM AE`	0.472	0.687	0.512	0.590	0.920	0.306	0.137	0.667	0.700	0.727	0.571	0.485
`TadGAN`	0.639	0.687	0.280	0.494	0.860	0.282	0.230	0.667	0.633	0.636	0.643	0.606
`VAE`	0.500	0.731	0.492	0.573	0.940	0.274	0.146	0.500	0.700	0.636	0.500	0.485
`Dense AE`	0.500	0.612	0.244	0.579	0.830	0.040	0.056	0.500	0.733	0.545	0.500	0.455
`GANF`	0.333	0.328	0.220	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`Azure AD`	0.806	0.940	0.176	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table C.9: Benchmark Summary Results Version `0.5.1`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score												
`AER`	0.575	0.803	0.482	0.799	0.987	0.898	0.712	0.500	0.750	0.667	0.703	0.571
`LSTM DT`	0.471	0.730	0.393	0.743	0.980	0.734	0.639	0.400	0.494	0.710	0.632	0.560
`ARIMA`	0.435	0.326	0.090	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.533	0.658	0.311	0.593	0.852	0.452	0.252	0.545	0.737	0.667	0.500	0.542
`TadGAN`	0.571	0.592	0.172	0.547	0.809	0.427	0.324	0.667	0.645	0.727	0.486	0.552
`VAE`	0.475	0.653	0.340	0.599	0.806	0.424	0.227	0.667	0.700	0.609	0.516	0.542
`Dense AE`	0.500	0.692	0.199	0.656	0.902	0.080	0.094	0.545	0.764	0.600	0.467	0.508
`GANF`	0.462	0.463	0.147	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`Azure AD`	0.051	0.019	0.015	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
Precision												
`AER`	0.568	0.850	0.402	0.830	0.99	0.995	0.931	0.500	0.808	0.526	0.565	0.600
`LSTM DT`	0.364	0.667	0.332	0.696	0.975	0.979	0.898	0.333	0.404	0.550	0.500	0.500
`ARIMA`	0.455	0.311	0.102	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.513	0.608	0.224	0.629	0.802	0.946	0.690	0.600	0.778	0.615	0.444	0.615
`TadGAN`	0.473	0.529	0.124	0.621	0.803	0.736	0.569	0.556	0.625	0.727	0.391	0.640
`VAE`	0.432	0.600	0.256	0.582	0.714	0.834	0.639	0.667	0.700	0.583	0.471	0.615
`Dense AE`	0.571	0.714	0.166	0.739	0.955	0.975	0.558	0.600	0.840	0.667	0.438	0.577
`GANF`	0.750	0.786	0.111	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`Azure AD`	0.026	0.009	0.008	0.167	0.484	0.542	0.217	0.029	0.060	0.089	0.062	0.099
Recall												
`AER`	0.583	0.761	0.604	0.770	0.985	0.818	0.577	0.500	0.700	0.909	0.929	0.545
`LSTM DT`	0.667	0.806	0.484	0.798	0.985	0.587	0.496	0.500	0.633	1.000	0.857	0.636
`ARIMA`	0.417	0.343	0.080	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.556	0.716	0.508	0.562	0.910	0.297	0.154	0.500	0.700	0.727	0.571	0.485
`TadGAN`	0.722	0.672	0.280	0.489	0.815	0.300	0.226	0.833	0.667	0.727	0.643	0.485
`VAE`	0.528	0.716	0.504	0.618	0.925	0.284	0.138	0.667	0.700	0.636	0.571	0.485
`Dense AE`	0.444	0.672	0.248	0.590	0.855	0.042	0.051	0.500	0.700	0.545	0.500	0.455
`GANF`	0.333	0.328	0.220	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`Azure AD`	0.806	0.94	0.176	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table C.10: Benchmark Summary Results Version `0.5.0`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`AER`	0.632	0.784	0.767	0.978	0.878	0.708	0.769	0.727	0.667	0.686	0.603
`LSTM DT`	0.481	0.708	0.726	0.973	0.740	0.638	0.400	0.474	0.759	0.649	0.560
`ARIMA`	0.435	0.324	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.514	0.686	0.600	0.888	0.443	0.227	0.667	0.800	0.522	0.500	0.483
`TadGAN`	0.482	0.573	0.612	0.818	0.371	0.327	0.615	0.645	0.538	0.512	0.551
`VAE`	0.538	0.635	0.556	0.803	0.457	0.257	0.545	0.700	0.593	0.571	0.567
`Dense AE`	0.545	0.683	0.629	0.877	0.087	0.098	0.545	0.741	0.632	0.571	0.500
`GANF`	0.462	0.463	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`Azure AD`	0.051	0.019	0.280	0.653	0.702	0.344	0.054	0.112	0.244	0.111	0.189
Precision											
`AER`	0.600	0.845	0.795	0.970	0.991	0.923	0.714	0.800	0.526	0.571	0.633
`LSTM DT`	0.368	0.662	0.678	0.961	0.989	0.883	0.333	0.391	0.611	0.522	0.500
`ARIMA`	0.455	0.307	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.529	0.658	0.630	0.846	0.929	0.679	0.667	0.880	0.500	0.444	0.560
`TadGAN`	0.426	0.539	0.690	0.806	0.703	0.558	0.571	0.625	0.467	0.379	0.528
`VAE`	0.500	0.580	0.549	0.703	0.869	0.680	0.600	0.700	0.500	0.571	0.630
`Dense AE`	0.600	0.729	0.723	0.964	0.977	0.549	0.600	0.833	0.750	0.571	0.609
`GANF`	0.750	0.786	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`Azure AD`	0.026	0.009	0.167	0.484	0.542	0.217	0.028	0.060	0.141	0.059	0.107
Recall											
`AER`	0.667	0.731	0.742	0.985	0.789	0.575	0.833	0.667	0.909	0.857	0.576
`LSTM DT`	0.694	0.761	0.781	0.985	0.591	0.499	0.500	0.600	1.000	0.857	0.636
`ARIMA`	0.417	0.343	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.716	0.573	0.935	0.291	0.137	0.667	0.733	0.545	0.571	0.424
`TadGAN`	0.556	0.612	0.551	0.830	0.252	0.231	0.667	0.667	0.636	0.786	0.576
`VAE`	0.583	0.701	0.562	0.935	0.310	0.158	0.500	0.700	0.727	0.571	0.515
`Dense AE`	0.500	0.642	0.556	0.805	0.046	0.054	0.500	0.667	0.545	0.571	0.424
`GANF`	0.333	0.328	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table C.11: Benchmark Summary Results Version `0.4.1`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`AER`	0.583	0.778	0.787	0.978	0.895	0.691	0.750	0.690	0.733	0.632	0.606
`LSTM DT`	0.457	0.707	0.743	0.980	0.748	0.651	0.421	0.474	0.733	0.649	0.571
`ARIMA`	0.435	0.324	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.493	0.662	0.588	0.885	0.446	0.237	0.667	0.712	0.615	0.552	0.492
`TadGAN`	0.543	0.620	0.558	0.828	0.428	0.321	0.571	0.585	0.583	0.516	0.559
`VAE`	0.533	0.634	0.575	0.833	0.444	0.230	0.545	0.689	0.615	0.483	0.533
`Dense AE`	0.545	0.683	0.646	0.902	0.082	0.075	0.545	0.755	0.600	0.581	0.483
`GANF`	0.462	0.463	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`Azure AD`	0.051	0.019	0.280	0.653	0.702	0.344	0.054	0.112	0.244	0.111	0.189
Precision											
`AER`	0.583	0.831	0.818	0.970	0.992	0.918	0.600	0.714	0.579	0.500	0.606
`LSTM DT`	0.348	0.639	0.691	0.975	0.986	0.901	0.308	0.391	0.579	0.522	0.500
`ARIMA`	0.455	0.307	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.486	0.605	0.623	0.849	0.939	0.658	0.667	0.724	0.533	0.533	0.536
`TadGAN`	0.489	0.538	0.631	0.826	0.766	0.560	0.500	0.543	0.538	0.471	0.543
`VAE`	0.513	0.590	0.561	0.742	0.855	0.639	0.600	0.677	0.533	0.467	0.593
`Dense AE`	0.600	0.750	0.722	0.960	0.952	0.507	0.600	0.870	0.667	0.529	0.560
`GANF`	0.750	0.786	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`Azure AD`	0.026	0.009	0.167	0.484	0.542	0.217	0.028	0.060	0.141	0.059	0.107
Recall											
`AER`	0.583	0.731	0.758	0.985	0.816	0.553	1.000	0.667	1.000	0.857	0.606
`LSTM DT`	0.667	0.791	0.803	0.985	0.603	0.510	0.667	0.600	1.000	0.857	0.667
`ARIMA`	0.417	0.343	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.731	0.556	0.925	0.293	0.145	0.667	0.700	0.727	0.571	0.455
`TadGAN`	0.611	0.731	0.500	0.830	0.297	0.225	0.667	0.633	0.636	0.571	0.576
`VAE`	0.556	0.687	0.590	0.950	0.300	0.140	0.500	0.700	0.727	0.500	0.485
`Dense AE`	0.500	0.627	0.584	0.850	0.043	0.041	0.500	0.667	0.545	0.643	0.424
`GANF`	0.333	0.328	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table C.12: Benchmark Summary Results Version `0.4.0`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`AER`	0.579	0.770	0.793	0.978	0.888	0.721	0.800	0.727	0.690	0.667	0.571
`LSTM DT`	0.472	0.717	0.744	0.987	0.735	0.652	0.400	0.545	0.759	0.585	0.579
`ARIMA`	0.435	0.324	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.548	0.681	0.611	0.877	0.456	0.233	0.545	0.764	0.667	0.452	0.500
`TadGAN`	0.558	0.610	0.568	0.824	0.427	0.320	0.471	0.656	0.720	0.556	0.559
`VAE`	0.500	0.648	0.594	0.809	0.450	0.236	0.500	0.712	0.560	0.500	0.525
`Dense AE`	0.554	0.683	0.665	0.889	0.074	0.094	0.545	0.727	0.632	0.533	0.508
`Azure AD`	0.051	0.021	0.279	0.653	0.702	0.344	0.054	0.113	0.250	0.112	0.189
Precision											
`AER`	0.550	0.855	0.812	0.970	0.996	0.930	0.667	0.800	0.556	0.545	0.600
`LSTM DT`	0.357	0.667	0.701	0.990	0.987	0.894	0.333	0.500	0.611	0.444	0.512
`ARIMA`	0.455	0.307	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.541	0.662	0.648	0.850	0.956	0.692	0.600	0.840	0.562	0.412	0.556
`TadGAN`	0.480	0.515	0.647	0.828	0.718	0.551	0.364	0.618	0.643	0.455	0.543
`VAE`	0.475	0.603	0.577	0.708	0.866	0.634	0.500	0.724	0.500	0.500	0.571
`Dense AE`	0.621	0.729	0.752	0.944	0.947	0.558	0.600	0.800	0.750	0.500	0.577
`Azure AD`	0.026	0.011	0.167	0.484	0.542	0.217	0.028	0.061	0.145	0.059	0.107
Recall											
`AER`	0.611	0.701	0.775	0.985	0.802	0.589	1.000	0.667	0.909	0.857	0.545
`LSTM DT`	0.694	0.776	0.792	0.985	0.586	0.514	0.500	0.600	1.000	0.857	0.667
`ARIMA`	0.417	0.343	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.556	0.701	0.579	0.905	0.299	0.140	0.500	0.700	0.818	0.500	0.455
`TadGAN`	0.667	0.746	0.506	0.820	0.304	0.225	0.667	0.700	0.818	0.714	0.576
`VAE`	0.528	0.701	0.612	0.945	0.304	0.145	0.500	0.700	0.636	0.500	0.485
`Dense AE`	0.500	0.642	0.596	0.840	0.038	0.051	0.500	0.667	0.545	0.571	0.455
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table C.13: Benchmark Summary Results Version `0.3.2`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`AER`	0.600	0.785	0.745	0.985	0.881	0.721	0.471	0.727	0.733	0.600	0.562
`LSTM DT`	0.500	0.680	0.741	0.978	0.734	0.633	0.400	0.481	0.786	0.611	0.603
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.493	0.715	0.620	0.863	0.447	0.241	0.667	0.712	0.609	0.516	0.516
`TadGAN`	0.595	0.645	0.531	0.829	0.416	0.350	0.714	0.645	0.818	0.541	0.580
`Dense AE`	0.507	0.661	0.665	0.887	0.082	0.098	0.400	0.778	0.632	0.581	0.542
`Azure AD`	0.050	0.027	0.279	0.653	0.702	0.344	0.053	0.068	0.250	0.068	0.269
Precision											
`AER`	0.618	0.810	0.760	0.985	0.983	0.907	0.364	0.800	0.579	0.462	0.581
`LSTM DT`	0.375	0.614	0.700	0.970	0.980	0.886	0.333	0.388	0.647	0.500	0.550
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.515	0.700	0.641	0.817	0.942	0.665	0.667	0.724	0.583	0.471	0.552
`TadGAN`	0.521	0.568	0.567	0.769	0.745	0.582	0.625	0.625	0.818	0.435	0.556
`Dense AE`	0.548	0.719	0.752	0.934	0.952	0.570	0.500	0.875	0.750	0.529	0.615
`Azure AD`	0.026	0.014	0.167	0.484	0.542	0.217	0.027	0.036	0.145	0.035	0.161
Recall											
`AER`	0.583	0.761	0.730	0.985	0.798	0.598	0.667	0.667	1.000	0.857	0.545
`LSTM DT`	0.750	0.761	0.787	0.985	0.587	0.492	0.500	0.633	1.000	0.786	0.667
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.472	0.731	0.601	0.915	0.293	0.147	0.667	0.700	0.636	0.571	0.485
`TadGAN`	0.694	0.746	0.500	0.900	0.289	0.250	0.833	0.667	0.818	0.714	0.606
`Dense AE`	0.472	0.612	0.596	0.845	0.043	0.054	0.333	0.700	0.545	0.643	0.485
`Azure AD`	0.806	0.567	0.848	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table C.14: Benchmark Summary Results Version `0.3.1`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`AER`	0.579	0.778	0.786	0.992	0.896	0.716	0.533	0.678	0.759	0.667	0.581
`LSTM DT`	0.486	0.703	0.752	0.985	0.743	0.635	0.400	0.545	0.733	0.667	0.580
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.500	0.690	0.625	0.867	0.446	0.238	0.667	0.764	0.583	0.500	0.500
`TadGAN`	0.512	0.658	0.566	0.858	0.422	0.331	0.714	0.625	0.750	0.588	0.559
`Dense AE`	0.554	0.661	0.650	0.874	0.087	0.090	0.545	0.778	0.526	0.516	0.464
`Azure AD`	0.050	0.020	0.279	0.653	0.702	0.344	0.053	0.068	0.250	0.068	0.269
Precision											
`AER`	0.550	0.831	0.810	1.000	0.996	0.928	0.444	0.690	0.611	0.545	0.621
`LSTM DT`	0.366	0.642	0.716	0.985	0.988	0.886	0.333	0.500	0.579	0.545	0.556
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.500	0.641	0.658	0.824	0.961	0.672	0.667	0.840	0.538	0.444	0.556
`TadGAN`	0.440	0.564	0.626	0.815	0.735	0.577	0.625	0.588	0.692	0.500	0.543
`Dense AE`	0.621	0.719	0.732	0.922	0.956	0.519	0.600	0.875	0.625	0.471	0.565
`Azure AD`	0.026	0.010	0.167	0.484	0.542	0.216	0.027	0.036	0.145	0.035	0.161
Recall											
`AER`	0.611	0.731	0.764	0.985	0.814	0.583	0.667	0.667	1.000	0.857	0.545
`LSTM DT`	0.722	0.776	0.792	0.985	0.595	0.495	0.500	0.600	1.000	0.857	0.606
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.746	0.596	0.915	0.291	0.145	0.667	0.700	0.636	0.571	0.455
`TadGAN`	0.611	0.791	0.517	0.905	0.296	0.232	0.833	0.667	0.818	0.714	0.576
`Dense AE`	0.500	0.612	0.584	0.830	0.046	0.049	0.500	0.700	0.455	0.571	0.394
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table C.15: Benchmark Summary Results Version `0.3.0`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`LSTM DT`	0.476	0.741	0.739	0.990	0.753	0.644	0.400	0.537	0.714	0.703	0.556
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.500	0.658	0.584	0.877	0.444	0.262	0.667	0.724	0.667	0.471	0.475
`TadGAN`	0.575	0.659	0.564	0.853	0.439	0.370	0.615	0.656	0.640	0.541	0.559
`Dense AE`	0.523	0.661	0.665	0.891	0.072	0.109	0.400	0.764	0.571	0.552	0.517
`Azure AD`	0.050	0.020	0.279	0.653	0.702	0.344	0.053	0.068	0.250	0.068	0.269
Precision											
`LSTM DT`	0.362	0.697	0.710	0.995	0.985	0.888	0.333	0.486	0.588	0.565	0.513
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.500	0.598	0.630	0.830	0.935	0.709	0.667	0.750	0.615	0.400	0.538
`TadGAN`	0.490	0.550	0.638	0.811	0.784	0.599	0.571	0.645	0.571	0.435	0.543
`Dense AE`	0.586	0.719	0.752	0.925	0.946	0.595	0.500	0.840	0.600	0.533	0.600
`Azure AD`	0.026	0.010	0.167	0.484	0.542	0.216	0.027	0.036	0.145	0.035	0.161
Recall											
`LSTM DT`	0.694	0.791	0.77	0.985	0.610	0.505	0.500	0.600	0.909	0.929	0.606
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.731	0.545	0.930	0.291	0.160	0.667	0.700	0.727	0.571	0.424
`TadGAN`	0.694	0.821	0.506	0.900	0.305	0.267	0.667	0.667	0.727	0.714	0.576
`Dense AE`	0.472	0.612	0.596	0.860	0.037	0.060	0.333	0.700	0.545	0.571	0.455
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table C.16: Benchmark Summary Results Version `0.2.1`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`LSTM DT`	0.460	0.703	0.752	0.980	0.733	0.643	0.400	0.481	0.643	0.684	0.568
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.480	0.690	0.600	0.870	0.436	0.243	0.545	0.750	0.615	0.571	0.525
`TadGAN`	0.558	0.650	0.559	0.890	0.412	0.374	0.500	0.677	0.692	0.500	0.567
`Dense AE`	0.529	0.661	0.652	0.899	0.087	0.092	0.545	0.741	0.632	0.552	0.517
`Azure AD`	0.050	0.020	0.279	0.653	0.702	0.344	0.053	0.068	0.250	0.068	0.269
Precision											
`LSTM DT`	0.359	0.654	0.716	0.975	0.991	0.895	0.333	0.388	0.529	0.542	0.512
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.462	0.653	0.610	0.825	0.943	0.663	0.600	0.808	0.533	0.571	0.571
`TadGAN`	0.480	0.567	0.642	0.837	0.746	0.635	0.500	0.656	0.600	0.409	0.559
`Dense AE`	0.562	0.719	0.746	0.960	0.956	0.553	0.600	0.833	0.750	0.533	0.600
`Azure AD`	0.026	0.010	0.167	0.484	0.542	0.216	0.027	0.036	0.145	0.035	0.161
Recall											
`LSTM DT`	0.639	0.761	0.792	0.985	0.581	0.502	0.500	0.633	0.818	0.929	0.636
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.731	0.590	0.920	0.283	0.149	0.500	0.700	0.727	0.571	0.485
`TadGAN`	0.667	0.761	0.494	0.950	0.284	0.265	0.500	0.700	0.818	0.643	0.576
`Dense AE`	0.500	0.612	0.579	0.845	0.046	0.050	0.500	0.667	0.545	0.571	0.455
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table C.17: Benchmark Summary Results Version `0.2.0`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`LSTM DT`	0.447	0.671	0.724	0.975	0.751	0.637	0.400	0.488	0.733	0.619	0.535
`ARIMA`	0.435	0.326	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.500	0.667	0.593	0.859	0.422	0.223	0.667	0.724	0.720	0.552	0.540
`TadGAN`	0.465	0.646	0.549	0.843	0.492	0.388	0.667	0.623	0.741	0.476	0.523
`Dense AE`	0.563	0.710	0.656	0.889	0.084	0.094	0.545	0.800	0.632	0.500	0.517
`Azure AD`	0.061	0.021	0.276	0.653	0.702	0.344	0.053	0.068	0.286	0.068	0.269
Precision											
`LSTM DT`	0.343	0.600	0.680	0.966	0.990	0.887	0.333	0.385	0.579	0.464	0.500
`ARIMA`	0.455	0.311	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.500	0.635	0.614	0.825	0.934	0.667	0.667	0.750	0.643	0.533	0.567
`TadGAN`	0.400	0.546	0.600	0.814	0.792	0.584	0.556	0.613	0.625	0.357	0.531
`Dense AE`	0.643	0.772	0.731	0.944	0.953	0.544	0.600	0.880	0.750	0.500	0.600
`Azure AD`	0.032	0.011	0.166	0.484	0.542	0.216	0.027	0.036	0.169	0.035	0.161
Recall											
`LSTM DT`	0.639	0.761	0.775	0.985	0.605	0.497	0.500	0.667	1.000	0.929	0.576
`ARIMA`	0.417	0.343	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.701	0.573	0.895	0.273	0.134	0.667	0.700	0.818	0.571	0.515
`TadGAN`	0.556	0.791	0.506	0.875	0.357	0.291	0.833	0.633	0.909	0.714	0.515
`Dense AE`	0.500	0.657	0.596	0.840	0.044	0.051	0.500	0.733	0.545	0.500	0.455
`Azure AD`	0.806	0.940	0.815	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table C.18: Benchmark Summary Results Version `0.1.7`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`LSTM DT`	0.466	0.689	0.739	0.975	0.746	0.645	0.400	0.500	0.759	0.585	0.551
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.507	0.667	0.601	0.880	0.445	0.231	0.667	0.712	0.667	0.516	0.508
`TadGAN`	0.517	0.634	0.551	0.841	0.484	0.376	0.571	0.689	0.769	0.563	0.559
`Dense AE`	0.507	0.693	0.665	0.904	0.078	0.090	0.545	0.786	0.600	0.581	0.533
`Azure AD`	0.061	0.021	0.276	0.653	0.702	0.344	0.053	0.068	0.286	0.068	0.269
Precision											
`LSTM DT`	0.358	0.619	0.684	0.966	0.984	0.894	0.333	0.413	0.611	0.444	0.528
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.487	0.623	0.624	0.861	0.945	0.654	0.667	0.724	0.615	0.471	0.577
`TadGAN`	0.434	0.543	0.642	0.801	0.792	0.585	0.500	0.677	0.667	0.500	0.543
`Dense AE`	0.548	0.733	0.752	0.971	0.950	0.532	0.600	0.846	0.667	0.529	0.593
`Azure AD`	0.032	0.011	0.166	0.484	0.542	0.216	0.027	0.036	0.169	0.035	0.161
Recall											
`LSTM DT`	0.667	0.776	0.803	0.985	0.601	0.504	0.500	0.633	1.000	0.857	0.576
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.528	0.716	0.579	0.900	0.291	0.140	0.667	0.700	0.727	0.571	0.455
`TadGAN`	0.639	0.761	0.483	0.885	0.348	0.277	0.667	0.700	0.909	0.643	0.576
`Dense AE`	0.472	0.657	0.596	0.845	0.040	0.049	0.500	0.733	0.545	0.643	0.485
`Azure AD`	0.806	0.940	0.815	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table C.19: Benchmark Summary Results Version `0.1.6`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`LSTM DT`	0.480	0.718	0.747	0.975	0.739	0.649	0.400	0.474	0.741	0.684	0.583
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.474	0.667	0.593	0.865	0.438	0.276	0.667	0.764	0.615	0.452	0.552
`TadGAN`	0.529	0.654	0.555	0.822	0.487	0.377	0.714	0.645	0.741	0.486	0.567
`Dense AE`	0.515	0.667	0.648	0.897	0.080	0.091	0.545	0.786	0.600	0.581	0.517
`Azure AD`	0.061	0.021	0.276	0.653	0.702	0.344	0.053	0.070	0.019	0.068	0.269
Precision											
`LSTM DT`	0.375	0.680	0.690	0.966	0.991	0.893	0.333	0.391	0.625	0.542	0.538
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.450	0.635	0.629	0.833	0.931	0.711	0.667	0.840	0.533	0.412	0.640
`TadGAN`	0.451	0.573	0.592	0.792	0.764	0.575	0.625	0.625	0.625	0.391	0.559
`Dense AE`	0.567	0.712	0.719	0.955	0.975	0.586	0.600	0.846	0.667	0.529	0.600
`Azure AD`	0.032	0.011	0.166	0.484	0.542	0.216	0.027	0.037	0.009	0.035	0.161
Recall											
`LSTM DT`	0.667	0.761	0.815	0.985	0.589	0.510	0.500	0.600	0.909	0.929	0.636
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.701	0.562	0.900	0.286	0.171	0.667	0.700	0.727	0.500	0.485
`TadGAN`	0.639	0.761	0.522	0.855	0.358	0.280	0.833	0.667	0.909	0.643	0.576
`Dense AE`	0.472	0.627	0.590	0.845	0.042	0.049	0.500	0.733	0.545	0.643	0.455
`Azure AD`	0.806	0.940	0.815	1.000	0.998	0.837	1.000	0.700	0.909	1.000	0.818

Table C.20: Benchmark Summary Results Version `0.1.5`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`LSTM DT`	0.532	0.704	0.735	0.980	0.743	0.653	0.400	0.462	0.467	0.615	0.548
`ARIMA`	0.344	0.307	0.744	0.816	0.782	0.684	0.429	0.472	0.538	0.429	0.513
`TadGAN`	0.575	0.644	0.626	0.700	0.494	0.381	0.714	0.677	0.800	0.450	0.592
`Azure AD`	0.061	0.021	0.271	0.653	0.697	0.337	0.053	0.068	0.019	0.068	0.269
						Precision					
`LSTM DT`	0.431	0.667	0.690	0.980	0.991	0.899	0.333	0.375	0.368	0.480	0.500
`ARIMA`	0.393	0.300	0.684	0.772	0.998	0.955	0.375	0.405	0.467	0.429	0.444
`TadGAN`	0.490	0.523	0.652	0.700	0.795	0.588	0.625	0.656	0.714	0.346	0.553
`Azure AD`	0.032	0.011	0.163	0.484	0.541	0.212	0.027	0.036	0.009	0.035	0.161
						Recall					
`LSTM DT`	0.694	0.746	0.787	0.980	0.594	0.513	0.500	0.600	0.636	0.857	0.606
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.636	0.429	0.606
`TadGAN`	0.694	0.836	0.601	0.700	0.359	0.281	0.833	0.700	0.909	0.643	0.636
`Azure AD`	0.806	0.940	0.787	1.000	0.978	0.816	1.000	0.733	0.909	1.000	0.818

Table C.21: Benchmark Summary Results Version `0.1.4`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`LSTM DT`	0.468	0.708	0.738	0.978	0.728	0.634	0.400	0.468	0.437	0.667	0.564
`ARIMA`	0.344	0.307	0.744	0.816	0.782	0.684	0.429	0.472	0.538	0.429	0.513
`Azure AD`	0.061	0.021	0.277	0.653	0.692	0.333	0.053	0.068	0.019	0.068	0.269
						Precision					
`LSTM DT`	0.379	0.662	0.679	0.970	0.984	0.890	0.333	0.383	0.333	0.545	0.489
`ARIMA`	0.393	0.300	0.684	0.772	0.998	0.955	0.375	0.405	0.467	0.429	0.444
`Azure AD`	0.032	0.011	0.184	0.484	0.537	0.216	0.027	0.036	0.009	0.035	0.161
						Recall					
`LSTM DT`	0.611	0.761	0.809	0.985	0.577	0.492	0.500	0.600	0.636	0.857	0.667
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.636	0.429	0.606
`Azure AD`	0.806	0.940	0.562	1.000	0.972	0.729	1.000	0.733	0.909	1.000	0.818

Table C.22: Benchmark Summary Results Version `0.1.3`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`LSTM DT`	0.495	0.750	0.757	0.987	0.756	0.643	0.400	0.531	0.452	0.718	0.620
`ARIMA`	0.489	0.424	0.753	0.856	0.783	0.693	0.429	0.576	0.538	0.545	0.513
						Precision					
`LSTM DT`	0.364	0.680	0.721	0.975	0.986	0.890	0.333	0.472	0.350	0.560	0.579
`ARIMA`	0.393	0.300	0.690	0.772	0.998	0.955	0.375	0.567	0.467	0.429	0.444
						Recall					
`LSTM DT`	0.774	0.836	0.798	1.000	0.613	0.503	0.500	0.607	0.636	1.000	0.667
`ARIMA`	0.647	0.724	0.829	0.961	0.644	0.543	0.500	0.586	0.636	0.750	0.606

Bibliography

Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016. ISSN 1084–8045. doi: https://doi.org/10.1016/j.jnca.2015.11.016. URL https://www.sciencedirect.com/science/article/pii/S1084804515002891.

Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. *Visualization of time-oriented data*. Springer Science & Business Media, 2011.

Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C. Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, Lorenzo Stella, Ali Caner Turkmen, and Yuyang Wang. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *Journal of Machine Learning Research*, 21(116):1–6, 2020.

Sarah Alnegheimish, Najat Alrashed, Faisal Aleissa, Shahad Althobaiti, Dongyu Liu, Mansour Alsaleh, and Kalyan Veeramachaneni. Cardea: An open automated machine learning framework for electronic health records. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 536–545. IEEE, 2020.

Sarah Alnegheimish, Dongyu Liu, Carles Sala, Laure Berti-Equille, and Kalyan Veeramachaneni. Sintel: a machine learning framework to extract insights from signals. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, pages 1855–1865. Association for Computing Machinery, 2022.

Sarah Alnegheimish, Laure Berti-Equille, and Kalyan Veeramachaneni. OrionBench: benchmarking time series generative models in the service of the end-user. In *2024 IEEE International Conference on Big Data (BigData)*, pages 1215–1222, 2024a.

Sarah Alnegheimish, Linh Nguyen, Laure Berti-Equille, and Kalyan Veeramachaneni. Can large language models be anomaly detectors for time series? In *2024 IEEE 11th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2024b.

Sarah Alnegheimish, Zelin He, Matthew Reimherr, Akash Chandrayan, Abhinav Pradhan, and Luca D'Angelo. M2AD: detecting anomalies in heterogeneous multivariate time series from multiple systems. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2025.

Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1), 2015.

Arundo Analytics. Anomaly detection toolkit, 4 2020. URL https://github.com/arundo/adtk.

Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 15–27. Springer, 2002.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the Language of Time Series. *Transactions on Machine Learning Research*, 2024.

Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. USAD: UnSupervised Anomaly Detection on Multivariate Time Series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3395–3404, 2020.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2):281–305, 2012. ISSN 1532-4435.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 359–370, 1994.

Aadyot Bhatnagar, Paul Kassianik, Chenghao Liu, Tian Lan, Wenzhuo Yang, Rowan Cassius, Doyen Sahoo, Devansh Arpit, Sri Subramanian, Gerald Woo, et al. Merlion: A machine learning library for time series. *arXiv preprint arXiv:2109.09265*, 2021.

Stella Biderman, USVSN Sai Prashanth, Lintang Sutawika, Hailey Schoelkopf, Quentin Gregory Anthony, Shivanshu Purohit, and Edward Raff. Emergent and predictable memorization in large language models. *Advances in Neural Information Processing Systems*, 36, 2023.

Peter Bloomfield. *Fourier analysis of time series: an introduction*. John Wiley & Sons, 2004.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen,

Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2022.

Anastasia Borovykh, Sander Bohte, and Cornelis W. Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2018.

George EP Box and David A Pierce. Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models. *Journal of the American statistical Association*, 65(332):1509–1526, 1970.

Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

Defu Cao, Furong Jia, Sercan O Arik, Tomas Pfister, Yixiang Zheng, Wen Ye, and Yan Liu.

TEMPO: Prompt-based generative pre-trained transformer for time series forecasting. *arXiv preprint arXiv:2310.04948*, 2023.

Lei Cao, Wenbo Tao, Sungtae An, Jing Jin, Yizhou Yan, Xiaoyu Liu, Wendong Ge, Adam Sah, Leilani Battle, Jimeng Sun, Remco Chang, Brandon Westover, Samuel Madden, and Michael Stonebraker. Smile: A system to support machine learning on eeg data at scale. *Proceedings of the VLDB Endowment*, 12(12):2230–2241, 2019.

Sayan Chakraborty, Smit Shah, Kiumars Soltani, Anna Swigart, Luyao Yang, and Kyle Buckingham. Building an automated and self-aware anomaly detection system. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1465–1475. IEEE, 2020.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

Ching Chang, Wen-Chih Peng, and Tien-Fu Chen. LLM4TS: Two-stage fine-tuning for time-series forecasting with pre-trained llms. *arXiv preprint arXiv:2308.08469*, 2023a.

Kent K Chang, Mackenzie Cramer, Sandeep Soni, and David Bamman. Speak, memory: An archaeology of books known to ChatGPT/GPT-4. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023b. doi: 10.18653/v1/2023.emnlp-main.453.

Wanpracha Art Chaovalitwongse, Ya-Ju Fan, and Rajesh 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*, 37(6):1005–1016, 2007.

Chris Chatfield and Haipeng Xing. *The analysis of time series: an introduction with R*. Chapman and hall/CRC, 2019.

K Chen, SC Lu, and HS Teng. Adaptive real-time anomaly detection using inductively generated sequential patterns. In *Fifth Intrusion Detection Workshop, SRI International, Menlo Park, CA*, 1990.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

Robert B Cleveland, William S Cleveland, Jean E McRae, Irma Terpenning, et al. Stl: A seasonal-trend decomposition. *J. off. Stat*, 6(1):3–73, 1990.

Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017.

Andrew A. Cook, Göksel Mısırlı, and Zhong Fan. Anomaly detection for IoT time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2020. doi: 10.1109/JIOT.2019.2958185.

Maria Cvach. Monitor Alarm Fatigue: An integrative review. *Biomedical Instrumentation & Technology*, 46(4):268–277, 2012. doi: 10.2345/0899-8205-46.4.268. URL https://array.aami.org/doi/abs/10.2345/0899-8205-46.4.268.

Enyan Dai and Jie Chen. Graph-augmented normalizing flows for anomaly detection of multiple time series. In *International Conference on Learning Representations*, 2022.

Xuewu Dai and Zhiwei Gao. From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis. *IEEE Transactions on Industrial Informatics*, 9(4):2226–2238, 2013.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

Dennis DeCoste. Automated learning and monitoring of limit functions. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 1997.

Samuel Dooley, Gurnoor Singh Khurana, Chirag Mohapatra, Siddartha V Naidu, and Colin White. Forecastpfn: Synthetically-trained zero-shot forecasting. In *Advances in Neural Information Processing Systems*, volume 36, pages 2403–2426. Curran Associates, Inc., 2023.

Cheng Feng, Long Huang, and Denis Krompass. Only the curve shape matters: Training foundation models for zero-shot multivariate time series forecasting through next curve shape prediction. *arXiv preprint arXiv:2402.07570*, 2024.

Jingkun Gao, Xiaomin Song, Qingsong Wen, Pichao Wang, Liang Sun, and Huan Xu. Robusttad: Robust time series anomaly detection via decomposition and convolutional neural networks. *arXiv preprint arXiv:2002.09545*, 2020.

Shanghua Gao, Teddy Koker, Owen Queen, Tom Hartvigsen, Theodoros Tsiligkaridis, and Marinka Zitnik. Units: A unified multi-task time series model. *Advances in Neural Information Processing Systems*, 37:140589–140631, 2024.

Azul Garza, Cristian Challu, and Max Mergenthaler-Canseco. TimeGPT-1. *arXiv preprint arXiv:2310.03589*, 2023.

Alexander Geiger, Dongyu Liu, Sarah Alnegheimish, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Tadgan: Time series anomaly detection using generative adversarial networks. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 33–43. IEEE, 2020.

Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE*, 11(4):1–31, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 27, 2014.

Clive WJ Granger. Some properties of time series data and their use in econometric model specification. *Journal of econometrics*, 16(1):121–130, 1981.

Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew G Wilson. Large language models are zero-shot time series forecasters. *Advances in Neural Information Processing Systems*, 36, 2023.

Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, 2013.

Riyaz Ahamed Ariyaluran Habeeb, Fariza Nasaruddin, Abdullah Gani, Ibrahim Abaker Targio Hashem, Ejaz Ahmed, and Muhammad Imran. Real-time big data processing for anomaly detection: A survey. *International Journal of Information Management*, 45: 289–307, 2019.

Lu Han, Han-Jia Ye, and De-Chuan Zhan. The capacity and robustness trade-off: Revisiting the channel independent strategy for multivariate time series forecasting. *arXiv preprint arXiv:2304.05206*, 2023.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7657–7666, 2021.

Yangdong He and Jiabao Zhao. Temporal convolutional networks for anomaly detection in time series. In *Journal of Physics: Conference Series*, volume 1213, page 042050. IOP Publishing, 2019.

Zelin He, Matthew Reimherr, Sarah Alnegheimish, and Akash Chandrayan. Weakly-supervised multi-sensor anomaly detection with time-series foundation models. In *NeurIPS Workshop on Time Series in the Age of Large Models*, 2024.

Zelin He, Sarah Alnegheimish, and Matthew Reimherr. Harnessing vision-language models for time series anomaly detection. *arXiv preprint arXiv:2506.06836*, 2025.

Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

Waleed Hilal, S. Andrew Gadsden, and John Yawney. Financial Fraud: a review of anomaly detection techniques and recent advances. *Expert Systems with Applications*, 193:116429, 2022. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2021.116429. URL https://www.sciencedirect.com/science/article/pii/S0957417421017164.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1031.

Bing Hu, Yanping Chen, and Eamonn Keogh. Time series classification under more realistic assumptions. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 578–586. SIAM, 2013.

Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 387–395, 2018.

Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

David Iverson. Inductive system health monitoring. In *International Conference on Artificial Intelligence*, 2004.

David Iverson. Data mining applications for space mission operations system health monitoring. In *SpaceOps 2008 Conference*, page 3212. American Institute of Aeronautics and Astronautics, 2008.

Vincent Jacob, Fei Song, Arnaud Stiegler, Bijan Rad, Yanlei Diao, and Nesime Tatbul. Exathlon: a benchmark for explainable anomaly detection over time series. *Proc. VLDB Endow.*, 14(11):2613–2626, July 2021. ISSN 2150-8097. doi: 10.14778/3476249.3476307. URL https://doi.org/10.14778/3476249.3476307.

Waqas Javed, Bryan McDonnel, and Niklas Elmqvist. Graphical perception of multiple time series. *IEEE TVCG*, 16(6):927–934, 2010.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

Siwon Kim, Kukjin Choi, Hyun-Soo Choi, Byunghan Lee, and Sungroh Yoon. Towards a rigorous evaluation of time-series anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7194–7201, 2022.

Sung Jae Kim, Jeong Soo Eo, Sarah Alnegheimish, and Kalyan Veeramachaneni. Anomaly detection for electric vehicle data: A case for the i-pedal function. *The 37th International Electric Vehicle Symposium and Exhibition*, 2024.

Jing Yu Koh, Daniel Fried, and Russ R Salakhutdinov. Generating images with multimodal language models. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 21487–21506. Curran Associates, Inc., 2023.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In Eduardo Blanco and Wei Lu, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012/.

Kwei-Harng Lai, Daochen Zha, Guanchu Wang, Junjie Xu, Yue Zhao, Devesh Kumar, Yile Chen, Purav Zumkhawaka, Minyang Wan, Diego Martinez, and Xia Hu. Tods: An automated time series outlier detection system. *arXiv preprint arXiv:2009.09822*, 2020.

Kwei-Herng Lai, Daochen Zha, Junjie Xu, Yue Zhao, Guanchu Wang, and Xia Hu. Revisiting time series outlier detection: Definitions and benchmarks. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Nikolay Laptev, Saeed Amizadeh, and Ian Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1939–1947, 2015.

Alexander Lavin and Subutai Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 38–44. IEEE, 2015.

Sean M. Law. STUMPY: A Powerful and Scalable Python Library for Time Series Data Mining. *Journal of Open Source Software*, 4(39):1504, 2019.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2980–2988, 2017.

Dongyu Liu, Sarah Alnegheimish, Alexandra Zytek, and Kalyan Veeramachaneni. MTV: visual analytics for detecting, investigating, and annotating anomalies in multivariate time series. *Proceedings of ACM Human-Computer Interaction*, 6(CSCW), April 2022. doi: 10.1145/3512950. URL https://doi.org/10.1145/3512950.

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICMD)*, pages 413–422, 2008.

Tiedong Liu and Bryan Kian Hsiang Low. Goat: Fine-tuned LLaMA outperforms GPT-4 on arithmetic tasks. *arXiv preprint arXiv:2305.14201*, 2023.

Helmut Lütkepohl and Markus Krätzig. *Applied time series econometrics*. Cambridge university press, 2004.

Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: A deep learning approach. *IEEE transactions on intelligent transportation systems*, 16(2):865–873, 2014.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, 38(4): 1346–1364, 2022.

Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. In *Proceedings of the 2016 International Conference on Machine Learning (ICML) Anomaly Detection Workshop*, 2016.

Larry M Manevitz and Malik Yousef. One-class SVMs for document classification. *Journal of Machine Learning Research*, 2(Dec):139–154, 2001.

José-Antonio Martínez-Heras and Alessandro Donati. Enhanced Telemetry Monitoring with Novelty Detection. *AI Magazine*, 35(4):37, 2014. ISSN 0738-4602.

Biswanath Mukherjee, L Todd Heberlein, and Karl N Levitt. Network intrusion detection. *IEEE network*, 8(3):26–41, 1994.

Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. TSB-UAD: An end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(8):1697–1711, 2022.

Daehyung Park, Yuuna Hoshi, and Charles C. Kemp. A Multimodal Anomaly Detector for Robot-Assisted Feeding Using an LSTM-Based Variational Autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

Eduardo HM Pena, Marcos VO de Assis, and Mario Lemes Proença. Anomaly detection using forecasting methods arima and hwds. In *2013 32nd International Conference of the Chilean Computer Science Society (SCCC)*, pages 63–66. IEEE, 2013.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Hena Ghonia, Rishika Bhagwatkar, Arian Khorasani, Mohammad Javad Darvishi Bayazi, George Adamopoulos, Roland Riachi, Nadhir Hassen, et al. Lag-Llama: Towards Foundation Models for Probabilistic Time Series Forecasting. *arXiv preprint arXiv:2310.08278*, 2023.

Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-Series Anomaly Detection Service at Microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 3009–3017, 2019.

Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, et al. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance Extreme Computing Conference*, pages 1–6, 2018.

Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of pca for traffic anomaly detection. In *Proc. of the 2007 ACM SIGMETRICS*, pages 109–120, 2007.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In S. Koyejo, S. Mohamed, A. Agarwal,

D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 36479–36494. Curran Associates, Inc., 2022.

Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, MLSDA'14, page 4–11, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450331593. doi: 10.1145/2689746.2689747. URL https://doi.org/10.1145/2689746.2689747.

Osman Salem, Alexey Guerassimov, Ahmed Mehaoua, Anthony Marcus, and Borko Furht. Sensor fault and patient anomaly detection and classification in medical wireless sensor networks. In *2013 IEEE International Conference on Communications (ICC)*, pages 4373–4378, 2013. doi: 10.1109/ICC.2013.6655254.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*, 2022.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162/.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, 28, 2015.

Micah J Smith, Carles Sala, James Max Kanter, and Kalyan 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*, pages 785–800, 2020.

Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan 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*, pages 2828–2837, 2019.

Jithin S Sunny, C Pawan K Patro, Khushi Karnani, Sandeep C Pingle, Feng Lin, Misa Anekoji, Lawrence D Jones, Santosh Kesari, and Shashaanka Ashili. Anomaly detection framework for wearables data: a perspective review on data concepts, data analysis algorithms and prospects. *Sensors*, 22(3):756, 2022.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27, 2014.

Nesime Tatbul, Tae Jun Lee, Stan Zdonik, Mejbah Alam, and Justin Gottschlich. Precision and Recall for Time Series. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018.

J Martínez Torres, PJ Garcia Nieto, L Alejano, and AN Reyes. Detection of outliers in gas emissions from urban areas using functional data analysis. *Journal of hazardous materials*, 186(1):144–149, 2011.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. TranAD: deep transformer networks for anomaly detection in multivariate time series data. *Proceedings of the VLDB Endowment*, 15(6):1201–1214, February 2022.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: a generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Kalyan Veeramachaneni, Ignacio Arnaldo, Vamsi Korrapati, Constantinos Bassias, and Ke Li. Aiˆ2: Training a big data machine to defend. In *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, pages 49–54. IEEE, 2016.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. Survey Certification.

Phillip Wenig, Sebastian Schmidl, and Thorsten Papenbrock. TimeEval: A benchmarking toolkit for time series anomaly detection algorithms. *Proceedings of the VLDB Endowment*, 15(12):3678–3681, 2022.

Lawrence Wong, Dongyu Liu, Laure Berti-Equille, Sarah Alnegheimish, and Kalyan Veeramachaneni. AER: auto-encoder with regression for time series anomaly detection. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 1152–1161, 2022.

Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 1–10, 2022a.

Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy. In *International Conference on Learning Representations*, 2022b.

Lei Xu, Shubhra Kanti Karmaker Santu, and Kalyan Veeramachaneni. MLFriend: Interactive prediction task recommendation for event-driven time-series data. *arXiv preprint arXiv:1906.12348*, 2019.

Hao Xue and Flora D. Salim. PromptCast: a new prompt-based learning paradigm for time series forecasting. *arXiv preprint arXiv:2210.08964*, 2023.

Takehisa Yairi, Yoshinobu Kawahara, Ryohei Fujimaki, Yuichi Sato, and Kazuo Machida. Telemetry-Mining: A Machine Learning Approach to Anomaly Detection and Fault Diagnosis for Space Systems. In *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pages 466–476. IEEE, 2006.

Nong Ye et al. A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, volume 166, page 169. Citeseer, 2000.

Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. Matrix Profile I: all pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1317–1322, 2016. doi: 10.1109/ICDM.2016.0179.

Ye Yuan, Guangxu Xun, Fenglong Ma, Yaqing Wang, Nan Du, Kebin Jia, Lu Su, and Aidong Zhang. Muvan: A multi-view attention network for multivariate temporal data. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 717–726. IEEE, 2018.

Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V Chawla. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 1409–1416, 2019.

Rui Zhang, Shaoyan Zhang, Sethuraman Muthuraman, and Jianmin 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*, pages 31–37, 2007.

Dequan Zheng, Fenghuan Li, and Tiejun Zhao. Self-adaptive statistical process control for anomaly detection in time series. *Expert Systems with Applications*, 57:324–336, 2016.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115, 2021.

Walter Zucchini and Iain L MacDonald. *Hidden Markov models for time series: an introduction using R*. Chapman and Hall/CRC, 2009.