Orion – A Machine Learning Framework for Unsupervised Time Series Anomaly Detection

by

Sarah Alnegheimish

B.S., King Saud University (2017)

Submitted to the Center for Computational Science and Engineering

and

Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Science in Computational Science and Engineering

and

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author . . . . . .
Center for Computational Science and Engineering
Department of Electrical Engineering and Computer Science
May 6, 2022

Certified by . . . . . .
Kalyan Veeramachaneni
Principal Research Scientist
Laboratory for Information and Decision Systems
Thesis Supervisor

Certified by . . . . . .
Youssef Marzouk
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by . . . . . .
Nicolas Hadjiconstantinou
Professor of Mechanical Engineering
Co-Director, Center for Computational Science and Engineering

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Orion – A Machine Learning Framework for Unsupervised Time Series Anomaly Detection

by

Sarah Alnegheimish

Submitted to the Center for Computational Science and Engineering
and
Department of Electrical Engineering and Computer Science
on May 6, 2022, in partial fulfillment of the
requirements for the degrees of
Master of Science in Computational Science and Engineering
and
Master of Science in Electrical Engineering and Computer Science

Abstract

With the recent proliferation of temporal observation data comes an increasing demand for time series anomaly detection. New methods to detect anomalies using machine learning are continuously emerging. However, algorithms alone only solve one aspect of the problem – finding anomalies. Existing systems often fail to encompass an end-to-end detection process, to facilitate comparative analysis of various anomaly detection methods, or to incorporate human knowledge to refine output. This precludes current methods from being used in real-world settings by practitioners who are not machine learning experts.

In this thesis, we introduce *Orion*, a machine learning framework for unsupervised time series anomaly detection. The framework supports all the steps of the anomaly detection process. It includes a pipeline hub to maintain many state-of-the-art approaches for time series anomaly detection including statistical and machine learning based methods. *Orion* logs the entire anomaly detection journey, providing detailed documentation of the status of a signal and anomalies over time. It enables users to analyze signals, compare methods, and investigate anomalies through an interactive visualization tool, where they can annotate events by modifying existing events, creating new ones, and removing them. Using these annotations, the framework aims to leverage human knowledge to improve the performance of the pipeline. We demonstrate the effectiveness and efficiency of *Orion* through a series of experiments from benchmarking to AutoML on three public time series datasets: NASA, Yahoo, and Numenta. In addition, we showcase the usability of our framework through a study conducted on a real-world use case involving spacecraft experts tasked with anomaly analysis tasks. *Orion*'s framework, code, and datasets are open-sourced at https://github.com/sintel-dev/Orion.

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

Acknowledgments

The last three years have been filled with excitement, knowledge, and growth. I have experienced many highs (and lows), and navigating through a global pandemic. I would like to express my appreciation to the wonderful people who have made a unequivocal impact on my journey.

Foremost, I would like to express my sincere gratitude to my advisor, Kalyan Veeramachaneni, thank you for being a great mentor and a continuous supporter. I have learned tremendously from our research, meetings, collaborations, and casual chats. Thank you for guiding me through this journey every step of the way, and recognizing my potential even when I was blind to it. Today, I am a better researcher than I was when I first got here, and I am looking forward to the next Ph.D. chapter.

Being at MIT is a dream come true, I am fortunate to be a part of the Data to AI Lab. I would like to thank fellow members of the DAI Lab for all the intellectual discussions and creating wonderful memories, particularly the hiking and ski trips. I would like extend my gratitude to Professor Laure Berti-Equille and Dongyu Liu for guiding me through my research, and for all the good times of our collaboration. To Carles Sala and Plamen Kolev, I have thoroughly enjoyed all our discussions around code design, review, and development, it definitely honed my software skills. Thank you to Cara Giaimo and Arash Akhgari for their support and always improving my research.

To my friends — Arwa, Abdullah, Abdulaziz, Ebrahim, Fahad, Lama, Mohammed, Mossab, and Yasmeen — thank you for your friendship and making Boston feel like home. I will always remember the great memories, especially the times when we filled the cold winters of Boston with ski trips. To Fai, thank you for making the NY to Boston journey frequently, I was fortunate to share the past year with you.

I am grateful to my family who have been my biggest supporters, I owe my success to you. To my mother, Iman, no words can ever describe my gratitude. Thank you for dedicating your life to us, providing us with the best resources, teaching us the to become the best version of ourselves, and investing your time and career in our

education. To my father, Abdulaziz, thank you for being a great example of an Academic. I am forever grateful to your support and unconditional love. To my siblings, thank you for adding joy to my journey. Hadeel, you have been and will continue to be a role-model to me, thank you for helping me overcome barriers and gain confidence. Norah, thank you for teaching me that there are no limits to what you can achieve, and helping me recognize my potential. Mohammed, a chef and signal processing genius, thank you for being a dependable resource. Yasmeen, thank you for filling my gallery with Luigi's photos, it lightens my day (even though I deny it) and makes me laugh when I needed it the most. Ibrahim, thank you for enriching my life — hneow – and sharing my love of football.

Lastly, I want to thank King Abdulaziz City for Science and Technology (KACST) for being my fellowship sponsor during my Masters. I am especially grateful for the Center for Complex Systems (CCS) at KACST for providing an excellent program to support young researchers. I would like to extend my appreciation to Dr. Anas Alfaris and Dr. Ahmad Alabdulkareem for their continued support of my academic journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid growth of temporal data over the past years has led to an increasing demand for time series analysis. Many industries have set up processes and workflows for analyzing time series in order to better monitor, control, and optimize products and services they offer. These workflows usually rely on data visualization, domain knowledge and some simple rule-based decision making. Even with the proliferation of Machine Learning (ML) in vision and language systems, integration of ML into these workflows is still an open problem. One of the most promising ML-based workflows is anomaly detection. In this thesis, we focus on time series anomaly detection, presenting the current challenges facing its practitioners and proposing solutions.

The detection of anomalies in time series data is a critical task with many monitoring applications. Effective Anomaly Detection (AD) methods can identify deviations from normal behavior and notify users, sounding the alarm about potential problems. Researchers have been developing these methods for decades [31]. As data has become larger, more complex, and increasingly multidimensional, traditional distance- [18], density- [74] or isolation-based [51] approaches have begun to perform less competitively in real-world scenarios.

Since then, machine learning and deep learning-based methods have garnered increased attention [73, 60, 72, 42]. With the wide diversity of available methods, choosing a particular method or evaluating which one is most suitable can be difficult. Depending on the application, it may be insufficient to rely on data benchmarks [47]

to determine the best methods, particularly when there is no ground truth. Ideally, it would be possible to test and compare the performances of all methods on a data set of interest. Furthermore, users may still run into the following problems.

Services often neglect the **human-in-the-loop** dimension, which is vital both for validating detected anomalies and for differentiating between true errors and legitimate exceptions. To confirm and compare anomalies, users — generally domain experts, machine learning researchers, or data scientists — resort to visualisation to inspect signals and view different aggregation levels. Usually, users need to shift to a programming language that they are comfortable with (e.g., `MATLAB`, `R`, or `python`) to complete this process, which can result in loss of information.

Many existing solutions consider individual time series in isolation – even though in most real-world situations, thousands of **multivariate time series** are correlated and monitored continuously. For instance, detecting collective and correlated anomalies across complex time series is often important in health monitoring. By bringing all the information into a single platform, we create a thorough knowledge base that enables decision-making.

Users may want to **customize** or compose an anomaly detection system for their own use cases. However, designing a platform to analyze a specific type of time series, detect anomalies therein, and integrate domain knowledge into the resulting validation is complicated, and there is no solution that supports this type of technical workflow. Such a solution would require a systematic definition of procedures and tasks, a data standardization module, a comprehensive set of application programming interfaces (APIs), a modular and extensible machine learning pipeline design, and an interactive investigation of anomalies.

To address these problems, we introduce project *Orion*. The framework tackles the problem of anomaly detection end-to-end, from the first step of time series ingestion through machine learning modeling, interactive visualization, and user feedback. It is a comprehensive, streamlined ecosystem that targets various user needs.

	Publication
Sintel	**Alnegheimish, S.**, Liu, D., Sala, C., Berti-Equille, L., & Veeramachaneni, K. (2022). *Sintel: A Machine Learning Framework to Extract Insights from Signals*. To appear in ACM International Conference on Management of Data (SIGMOD).
MTV	Liu, D., **Alnegheimish, S.**, Zytek, A., & Veeramachaneni, K. (2022, April). *MTV: Visual Analytics for Detecting, Investigating, and Annotating Anomalies in Multivariate Time Series*. In Proceedings of the ACM on Human-Computer Interaction (CSCW).
TadGAN	Geiger, A., Liu, D., **Alnegheimish, S.**, Cuesta-Infante, A., & Veeramachaneni, K. (2020, December). *TadGAN: Time Series Anomaly Detection using Generative Adversarial Networks*. In IEEE International Conference on Big Data (Big Data).
Orion Library	Orion GitHub Repository, `https://github.com/sintel-dev/Orion`.

Table 1.1: Published work from the *Orion* project.

In writing this thesis, several chapters were adapted and extended from previously published work. Table 1.1 lists some of our published work and open-source libraries. I am grateful to all the co-authors and collaborators, this project would not have been possible without them.

Project *Orion* encompasses a `python` library called **Orion** [1] for unsupervised time series anomaly detection using machine learning. The library is open-source and currently has 595 stars and 110 forks, Table 1.2 shows the team's contribution to the library. In addition, it has a web-application named **MTV** [2] for multivariate time series visualization and annotation. Figure 1-1 depicts how Orion + MTV connect together serving the human-in-the-loop analysis of events that are produced by the anomaly detection algorithms. We will visit the algorithms and components of the framework in Chapter 5 and 6 respectively.

[1] `github.com/sintel-dev/Orion`
[2] `github.com/sintel-dev/MTV`

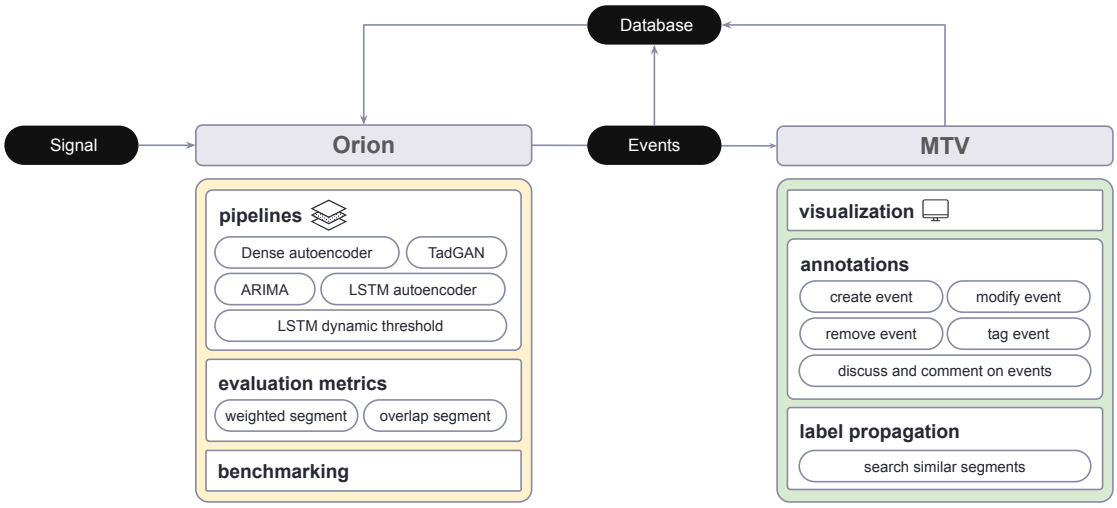


Figure 1-1: The framework includes the Orion library for detecting anomalies in time series data and MTV for visualizing and interacting with events.

				Code Contribution	
	Commits	Pull Requests	Issues Raised	++	−−
Overall	407	112	177	71,377	29,338
Myself	141	57	46	47,849 (65%)	13,019 (44%)

Table 1.2: Contributions to Orion library.

1.1 Summary of Contributions

Our main contributions are summarized as follows:

A framework for end-to-end time series anomaly detection. *Orion* provides a suite of anomaly detection pipelines executable through a user-friendly interface (Table 1.1 Sintel & Orion Library). Users kickstart the system by presenting a signal, which trains a model and returns the detected anomalies. The framework's modular nature facilitates the creation, exchange and reuse of primitives between different pipelines.

An ecosystem for user interaction and annotation-based learning. The framework includes a human-in-the-loop component, allowing domain experts to properly annotate and interact with detected anomalies (Table 1.1 MTV). We support this with a visualization tool that aids users in the inspection and investigation processes. The feedback component within the framework learns from human annotations to further improve detection performance.

A standardized benchmarking framework for time series anomaly detection pipelines. We designed a comprehensive benchmarking suite to compare multiple pipelines on a collection of different time series datasets (Table 1.1 Sintel, TadGAN & Orion Library). The benchmarking suite features intricate evaluation metrics designed specifically for anomaly detection. This feature enables users to run multiple experiments under the same conditions in order to obtain fair, empirical comparisons. The benchmarking suite currently has 6 pipelines (1 statistical and 5 deep learning models) and 2 evaluation mechanisms.

A comprehensive evaluation. We evaluate our framework by benchmarking all pipelines on a collection of 11 datasets from three reputable data sources – NASA, Yahoo, and Numenta – and reporting their quality and computational performance (Table 1.1 Sintel & Orion Library). Moreover, we conduct an experiment to assess the framework's ability to learn from experts' annotations. We compare the features of *Orion* against existing systems. Lastly, we conduct a user study with a leading satellite operation company to test *Orion* in a real-world setting.

1.2 Structure of Thesis

The remainder of this thesis is structured as follows. We highlight the motivations behind *Orion* in Chapter 2 and state the relevant challenges more concretely. Chapter 3 goes through some useful background information on time series and anomaly detection in general, followed by related work in Chapter 4. We conceptualize the

problem and define it more clearly in Chapter 5, where we go through anomaly detection and evaluation. Chapter 6 architects the system and its components from pipelines to benchmarks to interactions. Next, we evaluate the system and report its effectiveness in Chapter 7 and present our results from a real-world user study in Chapter 8. Lastly, we go through some interesting discussion points and some of the hurdles of time series anomaly detection in Chapter 9 before concluding in Chapter 10.

Chapter 2

Motivation

Anomaly Detection (AD) problems have existed for decades [17, 31, 32, 35], but solutions are still far off. To illustrate some of the most common challenges in time series anomaly detection problems, we walk through a real-world example.

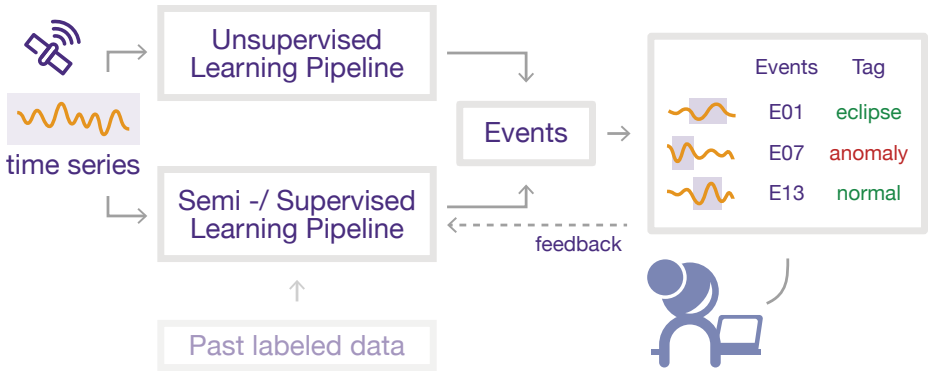


Figure 2-1: Anomaly detection workflow. Use an unsupervised pipeline to locate anomalies, which are then presented to the expert for annotation. The annotated events are provided to the semi-/supervised pipeline — which can be pre-trained with past labeled data — to learn from feedback and keep improving.

2.1 Real-World Scenario

To explain the motivation for *Orion* and illustrate the anomaly detection workflow (Figure 2-1), we describe a real-world scenario drawn from our three-year collabo-

ration with a world-leading communication satellite company. One major objective of the company's operations team is to detect unexpected behaviors (i.e., anomalies) in tens of thousands of signals. We collaborated with a spacecraft program manager and 5 senior satellite engineers, each of whom has considerable experience in telemetry data analysis (between 5 and 17 years), but relatively little machine learning experience (0 to 3 years).

The team works with multiple spacecrafts. Each spacecraft telemetry database contains around $37,000$ signals spanning 9 different subsystems. Each signal is a univariate time series collected at the microsecond level, and has been tracked for over 10 years. The team's conventional approach to anomaly detection is based around setting and adjusting thresholds in order to flag anomalous intervals. The team then reviews the suspicious intervals manually, often using simple `csv` files, and examines individual signals in a third-party platform such as `MATLAB`. Around 20 alarms are reported every day, most of which can be resolved within a few hours. For some reports that are identified as true alarms but aren't considered urgent, the experts gather further information over some time window to help explain the root cause and the way forward.

Over the course of this process, we identified challenges routinely faced by this team, as well as others we have worked with. We describe them as follows:

C1: Needle in a Haystack With the abundance of data nowadays, it is infeasible to manually inspect a large number of signals. Even with traditional methods, such as threshold-based methods, setting and adjusting the thresholds can be demanding and laborious, forcing teams to restrict their focus to a subset of a few hundred signals chosen based on domain knowledge and neglect the remaining thousands of signals. Moreover, teams want to identify contextual anomalies — anomalies that do not exceed a normal range, but are unusual compared to neighboring values. Classical methods often fail to find these anomalies. Teams would like to use machine learning models for this task, but this can be difficult when team members have limited machine learning experience.

C2: Lack of Labels In most time series settings, there are no built-in labels, and it may be difficult for users to label signals manually. A successful anomaly detector should be able to process a large number of signals and suggest potential anomalies. Unsupervised learning approaches achieve this by detecting anomalies in situations where data is not labeled and signals do not follow a particular pattern.

C3: The "Best" Pipeline There are many different approaches for anomaly detection, but there is no single pipeline that can solve everyone's problem. A variety of anomalies are possible across different signals, and existing approaches excel at detecting some while failing to find others. Furthermore, each pipeline is composed of a wide range of pre-/post-processing methods for time series (e.g., aggregation, normalization, etc.), giving a nearly infinite number of possible options. With the copious amounts of AD methods available, teams struggle to know which machine learning model to select for a particular dataset, making their jobs more difficult. Moreover, as machine learning is a continuously growing field, we ask ourselves, *how can we keep up with the latest developments?*

C4: Feedback Providing a set of potential anomalies is only the first step in an anomaly detection workflow. The teams we worked with found that machine learning models often flag unusual patterns even when these patterns do not necessarily indicate a problem. For example, a certain maneuver might cause patterns that are then flagged even though they are not troublesome. Teams are eager for their models to ignore these patterns, but struggle to teach them how to do this. Human-in-the-loop integration is essential for continuous learning.

C5: System Usability Many proposed anomaly detection methods require users to have substantial technical/programming knowledge, or even machine learning knowledge. This amount of overhead means that users struggle to apply these methods to their own use cases, or find themselves shifting to a programming language that they are comfortable with (e.g., `MATLAB`, `R`, or `python`), operating independently from the base code to overcome these barriers. The usability of a system is crucial,

as it allows users to detect, inspect, and annotate anomalies in time series data. The right abstractions and a concise set of application programming interfaces (APIs) can overcome these hurdles and help us serve three classes of population, which we define as follows:

1. *end users* who want to detect anomalies within their own data.
2. *system developers* who want to build their own anomaly detection systems.
3. *machine learning researchers* who want to create, access, and compare with some of the best anomaly detection methods.

We define these user types in more detail in Section 3.4.

In our work, we aim to address all of these challenges. *Orion* is an automated end-to-end framework that is able to identify anomalous events from tens of thousands of signals (**C1**). It integrates various state-of-the-art unsupervised anomaly detection pipelines, and provides simple user-friendly APIs to interact with the framework (**C2**, **C5**). With the benchmark suite, *Orion* provides results in an organized manner. This helps the team to easily pick a suitable, existing, and verified unsupervised pipeline from our collection, or even to create their own (**C3**). Through MTV, which is *Orion*'s visual interface, the team investigates flagged anomalies, annotates them, and incorporates the feedback back into the framework (**C4**).

Chapter 3

Background

3.1 Time Series Format

A time series is a collection of data points that are indexed by time. There are many forms in which time series can be stored. For the purpose of this thesis, 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.

Time series as we reference them occur in both univariate (Table 3.1) and multivariate (Table 3.2) forms.

timestamp	value
1222819200	215
1222840800	124
⋮	⋮
1334905600	15

Table 3.1: Univariate time series

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

Table 3.2: Multivariate time series

In a case where we have multiple entity references in the data, as shown in Table 3.3, we extract each entity on its own to conform to the representation we defined earlier, as shown in Table 3.4 and Table 3.5.

entity	timestamp	v_1	$\cdots$	v_k
E04	1222819200	13		12
E25	1222819200	7		34
E25	1222862400	9		53
E04	1222884000	62		21
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
E25	1334901600	3		23
E04	1334905600	31		52

Table 3.3: Multiple entity time series

timestamp	v_1	$\cdots$	v_k
1222819200	13		12
1222884000	62		21
$\vdots$	$\vdots$	$\cdots$	$\vdots$
1334905600	31		52

Table 3.4: E04 time series

timestamp	v_1	$\cdots$	v_k
1222819200	7		34
1222862400	9		53
$\vdots$	$\vdots$	$\cdots$	$\vdots$
1334901600	3		23

Table 3.5: E25 time series

3.2 Anomalies in Time Series

Depending on the source of data and its domain, there can be many types of anomalies within a dataset. It is useful to consider two broad categories:

- *Point anomalies* are single values that fall within low-density value regions (outliers). We identify point anomalies by a single timestamp. If we witness a series of consecutive point anomalies, we refer to them as *collective anomalies*, represented by timestamp intervals. These anomalies are often detected using statistical-based outlier detection methods [35].

- *Contextual anomalies* are values that are anomalous with regard to the context. In other words, it falls out of place when compared with neighboring values. We sometimes refer to them as in-signature anomalies. We identify such anomalies by an interval, denoted via a start timestamp and an end timestamp.

Figure 3-1 illustrates the difference between anomaly types, where contextual anomalies cannot be found by pure thresholds.

(a) Collective anomalies (b) Contextual anomalies

Figure 3-1: Illustration of (a) point/collective anomalies (b) contextual anomalies.

Not all anomalies are problematic; we expand on this notion in Chapter 9.

3.3 Anomaly Detection Methods

There are many anomaly detection methods. In this section, we categorize some of the most common approaches to this task.

3.3.1 Statistical Methods

People have long been coming up with ways to systematically identify anomalous sequences within a time series. Static thresholding is one of the simplest techniques. With this strategy, an alert is raised whenever a data point exceeds the expected range. However, this approach often fails to detect contextual anomalies. Various statistical methods have been proposed to improve upon thresholding, such as Statistical Process Control (SPC) [75], in which data points are identified as anomalies if they fail to pass statistical hypothesis testing. However, human expertise is still required to set prior assumptions.

3.3.2 Machine Learning Methods

With the recent proliferation of machine learning methods, further anomaly detection approaches have been proposed. One of the most interesting ideas involves using recurrent neural networks (RNNs) to recognize a pattern sequence and use an estimator to "forecast" the expected value. From there, we can locate any anomalies by pinpointing discrepancies between the forecasted signal and the real one [37]. RNNs have found great success in capturing temporal correlations, which makes them useful

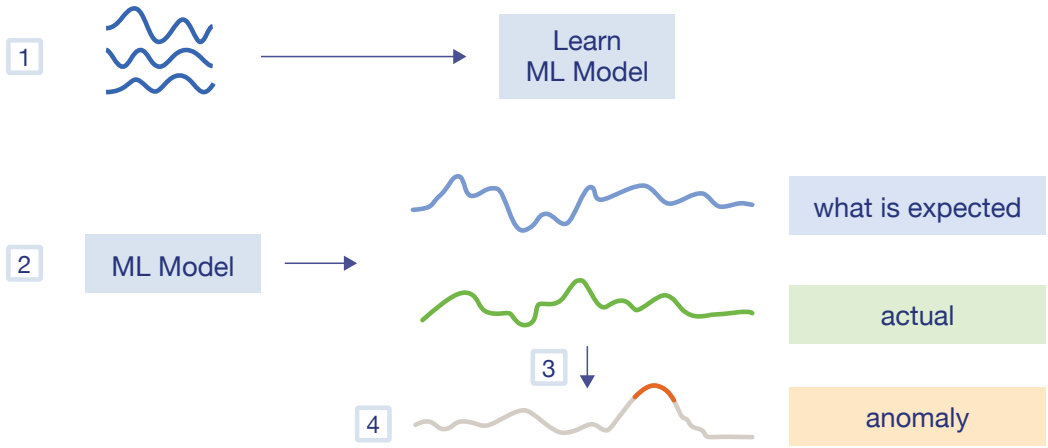


Figure 3-2: General principle of how deep learning models are used to find anomalies without labels. (1) Apply deep learning to learn the pattern of the data; (2) Use the learned model to generate another time series; (3) Compare what the model expects with the actual time series value; (4) Use this discrepancy to extract anomalies.

for time series.

Moreover, Deep Learning (DL) methods make judicious use of available data to learn the underlying structure of a time series, enabling them to perform complicated tasks such as anomaly detection. The general principle behind deep learning models (prediction- or reconstruction-based) is to generate an "expected" signal that represents the pattern of the original signal free from anomalies. Figure 3-2 demonstrates this process. In the end, this will result in a sequence of "errors" for each time point that measures the likelihood of that time point being an anomaly.

3.4 Personas

As suggested in Chapter 2, there is a wide range of users interested in solving time series anomaly detection tasks using machine learning. In this section, we expand on three user personas that *Orion*'s system aims to serve. Table 3.7 summarizes some of the interesting functionalities supported by *Orion* in order to accommodate each type of user's needs.

3.4.1 End User

End users are those who want to solve anomaly detection problems. These users can come from different backgrounds, from satellite operations to marketing and sales. It is important to create a system that appeals to users with diverse types of experience. There are different types of end users. We detail the differences in terms of **usage** and **skillset**. For usage, users can be: (1) *snapshot user*: These users are interested in trying out the system once to monitor signals and explore. They could also be curious about the library. First impressions are critical to draw this type of user into using the system more regularly. (2) *continuous user*: These users perform anomaly detection regularly. For example, a continuous user may work for a company and be tasked with monitoring time series and checking for anomalies.

We further break down these types of users based on skill levels. Table 3.6 presents the three types of users based on their skillset: expert end user, causal end user, and layman end user.

Skill Level	Description
EXPERT	Users who are competent in `python` can further improve the system by contributing to it. These users like to dive into the code and modify it as they see fit.
CASUAL	These users are familiar with the syntax and setup of python, but are not as experienced with machine learning or the task itself. In this case, users interact with high-level APIs to complete their tasks, and are not exposed to the intricacies of the system.
LAYMAN	These users do not know python, but are interested in solving a problem. To appeal to this category, we require a graphical interface to make interactions with the system as intuitive as possible.

Table 3.6: Description of the different end users based on skillset.

3.4.2 System Developer

A system developer is part of a team with a mandate to build an anomaly detection system. These systems are then provided to the end user (the first type of user in our categorization) who performs an anomaly detection task. These users emerge when companies need to create an in-house anomaly detection system to accommodate an end user, but cannot rely on commercialized tools and must instead build their own. In addition, system developers need to incorporate their own specific workflows to adapt to their domain and requirements.

3.4.3 Machine Learning Researcher

The Machine Learning researcher develops and tests ML models. These users are not particularly interested in the concept of a system; however, they would like to investigate how it compares to the current state of the art. One example of such a user might be a mathematician who wants to create an anomaly detection method that outperforms existing approaches. Most of the time, their objective is centered around publishing their work in conferences and journals. However, with the lack of infrastructure, it is hard to verify the claims of their model. More so, it is difficult to compare different methods; running into hiccups trying to benchmark models as soon as they become available. Having a system to support this exploration would make their job easier and more efficient.

End User	
Question	I want to detect anomalies within a signal: where do I find an anomaly detection method? do I need to learn a completely new API usage to use this method? how do I put my data into an acceptable format?
Challenge	(1) lack in background knowledge about programming and machine learning, (2) not invested in knowing the details about the best anomaly detection method
Functionality	(1) simplified API (similar to `scikit-learn` style) that relies on fit for training a pipeline and detect for detecting anomalies in a pipeline. Alternately, these two can be merged into `fit_detect`. (2) docker installation for easy offline setup. (3) thorough documentation of functionalities, database, and API.
System Developer	
Question	I want to create an anomaly detection system for my end user: where should I start? should I use `scikit-learn`? build a visualization? if there are multiple pipelines, how to get the best one?
Challenge	(1) typically requires a team effort to build, (2) they don't know what the best modeling technique is, (3) they might not know the proper workflow, (4) anomaly detection might not be their specialty.
Functionality	(1) thorough database schema. (2) simple `RESTful` API. (3) a suite of pipelines from the best methods available that have been validated and tested. (4) an complete system with extensiblity to accommodate special workflows.
ML Researcher	
Question	I want to create the best anomaly detection pipelines; what I need is to gather datasets, find computing resources, and discover which pipeline i should be comparing against.
Challenge	(1) they are keen to know the best ML pipeline. (2) ML folks who are interested in building anomaly detection methods might not have the right background and resources to build proper infrastructure.
Functionality	(1) an evaluation functionality. (2) a benchmarking functionality that allows for end-to-end model comparison. (3) a primitive profiling feature to monitor the execution performance of the pipeline.

Table 3.7: Description of user personas targeted by *Orion*. We visit their requirements through the questions they seek to answer the, what challenges they face, and how does the functionality in *Orion* satisfy their requirement.

Chapter 4

Related Work

Orion lies at the intersection of four domains: time series anomaly detection algorithms, time series anomaly detection systems, anomaly detection benchmark systems, and active incorporation of user feedback.

4.1 Time Series Anomaly Detection Algorithms

Many algorithms have been proposed to address time series anomaly detection [17, 29]. The most basic approaches simply flag regions where values exceed a certain threshold [53, 25]. While these methods are intuitive, they struggle to detect contextual anomalies. More advanced methods are based on statistical hypothesis testing [75], clustering [40, 41], and/or machine learning [69]. We summarize the categories of anomaly detection methods into *proximity-*, *prediction-*, and *reconstruction-based* methods.

4.1.1 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) [8] – 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) [12] and Clustering-Based Local Outlier Factor [33], 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 we need prior knowledge about the expected number and duration of anomalies. Moreover, most of these methods do not capture temporal correlations.

4.1.2 Prediction Methods

Prediction-based methods learn a predictive model to learn the patterns of the given time series data, and then use that model to predict 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 indicates where the anomalous regions are.

Some of the most classic techniques in time series anomaly detection include statistical methods used in a prediction-based approach. For example, ARIMA [55], Holt-Winters [55], and FDA [63] 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. [37] 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 aimed to prune detected anomalies that are very close in error scores to normal intervals.

4.1.3 Reconstruction Methods

Reconstruction-based methods learn a model to capture the latent structure (low-dimensional representations) of the given time series data and then create a synthetic

reconstruction of the signal. Reconstruction-based methods assume that anomalies lose information 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) [57], a dimensionality-reduction technique, can be used to reconstruct data, but it is limited to linear reconstruction and requires data to be highly correlated and to follow a Gaussian distribution [22]. With respect to deep learning, further reconstruction-based techniques have been investigated. These include the use of Auto-Encoder (AE) [52], Variational Auto-Encoder (VAE) [6], and Generative Adversarial Networks (GAN) [28].

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 — to adequately assist users in making decisions.

4.2 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. They address a variety of tasks such as classification [14], feature extraction [19], and anomaly detection [66, 46, 56, 27]. Table 4.1 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 a human-in-the-loop workflow. In contrast, *Orion* aims to provide an end-to-end development workflow to aid all types of users. Constructing new primitives and pipelines can be easily integrated using a `fit-predict` interface with minimal overhead [59]. Moreover, we provide a human-in-the-loop (HIL) workflow to analyze and annotate

the detected anomalies, which feeds back into the system for improvement over time. This subsystem leveraging user annotation is neglected by all existing systems.

4.3 Anomaly Detection Benchmarks

Training and optimizing deep learning models can be computationally expensive, making the process of selecting and comparing pipelines difficult. Benchmarking frameworks has now become a necessity to evaluate and compare the performance of models in a standardized end-to-end fashion [21]. With respect to anomaly detection, Lavin and Ahmad [47] 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. [42] introduced Exathlon – a benchmark framework for anomaly detection and explanation discovery. The framework features a systematically generated dataset from real-world data traces. In addition, it elicits some of the intricacies involving time series benchmarks, including evaluation metrics and performance monitoring. Currently, benchmarking frameworks have limited pipelines, and are not easily extendable.

4.4 Active Incorporation of User Feedback

A lot of work has focused on developing anomaly detection methods using supervised approaches [17, 29]. These methods are limited to the availability of labels. On the other hand, semi-supervised and unsupervised approaches require few to no labels at all [28]. In order to effectively detect anomalies, a system must continuously learn to adapt to the expert feedback.

Pelleg and Moore [54] propose an active learning approach that first generates the detected anomalies, selects the 35 strangest objects, prompts the expert to classify them, then repeats the process. Similarly, Veeramachaneni et al. [66] combines supervised and unsupervised output by selecting k entities for users to label, in which half are obtained from an unsupervised method and the other half is provided by the

supervised method. After all entities are labeled, they retrain the supervised model, select another k objects, and repeat the cycle. Das et al. [23] suggest an iterative process by training a supervised model (classifier) and selecting the points that are most likely to be outlier objects for expert review. Then they update the model and the weights of each feature accordingly. On the other hand, Chai et al. [15] aim to label the entire set of detected outlier objects by prompting users to answer questions. The questions are selected and optimized to cover as much information as possible about given objects. Based on found knowledge, the labels of the objects are adjusted. Once the labels are assigned, a supervised model can be trained on the given data. Although all of these methods are promising, they have been developed specifically for tabular data and require prior knowledge of data distribution.

		MS Azure [56]	ADTK [7]	Luminaire [16]	TODS [45]	Telemanom [37]	NAB [1]	EGADS [46]	Stumpy [48]	GluonTS [2]	*Orion*
Persona	End User	✓	✓	✓	✗	✗	✗	✗	✓	✗	✓
	System Builder	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
	ML Researcher	✗	✗	✗	✓	✓	✓	✓	✗	✓	✓
Engine	Preprocessing	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓
	Modeling	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
	Postprocessing	✗	✓	✓	✓	✗	✗	✗	✓	✗	✓
Modular		✗	✓	✓	✓	✗	✗	✗	✓	✓	✓
Comp.	Evaluation	✗	✓	✗	✗	✓	✗	✗	✗	✗	✓
	Benchmark	✗	✗	✗	✓	✗	✓	✗	✗	✓	✓
	Database	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
API	`lang. specific`	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓
	`RESTful`	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
HIL		✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Table 4.1: Comparison of anomaly detection software. 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 *databases* 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.

Chapter 5

Time Series Anomaly Detection

Prior to building a system, we look closely at the task of anomaly detection. In this chapter, we formalize the task and its objectives, detail several approaches to using deep learning for anomaly detection, and introduce evaluation mechanisms specific to the anomaly detection task.

5.1 Task Definition

In an anomaly detection task, we aim to detect if any intervals in the time series data are anomalous; i.e. deviate from "normal". Given $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, a multivariate signal with m channels where $\mathbf{x}_i \in \mathbb{R}^m$ is an entry at time i and has the representation $\mathbf{x}_i = \{x_i^1, x_i^2, \ldots, x_i^m\}$, and assuming there exists a set of *variable-length* anomalies $A = \{(t_s, t_e) \mid 1 \leq t_s < t_e \leq n\}$ that is <u>unknown</u> a priori, *Orion* aims to detect A using a combination of machine and human intelligence. Note that *Orion* addresses *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 a typical case in this problem.

5.1.1 Distinction from Classification

A time series classification task takes as input a univariate or multivariate signal X, and maps it to a probability distribution over the class variable values (*labels*) [38]. In an anomaly detection task, these class labels pertain to whether something is an anomaly or not. For example, Cao et al. [14] takes as input EEG data, divides the raw EEG data into segments of fixed duration, then produces a single output that assigns a label to that particular segment. Hu et al. [36] users time series classification to label ECG data into their respective physical activity label: normal-walking, walking-very-slow, descending-stairs, cycling, inactivity, etc.

(a) Detection task (b) Classification task

Figure 5-1: Illustration of (a) detection task where the objective is to find anomalies in time series data. (b) classification task where the objective is to assign a label to a time series or a segment.

We distinguish this classification problem from our anomaly detection task (Figure 5-1). In a detection task, the model works with the entire time series in its raw format and must predict whether the signal contains anomalies, as well as *localize* them by producing start and end time stamp intervals as shown in Figure 5-1(a).

5.1.2 Distinction from Supervised Anomaly Detection

The two paradigms of machine learning are learning with labels (supervised learning) and learning with unlabeled data (unsupervised learning) [43]. As described in Chapter 2, time series data typically suffers from lack of labels, which incentivizes the use of unsupervised methodologies. Hence, a system like *Orion* should work with only the time series data as input. Once the *unsupervised* model produces a set of candidate anomalies, they should be reviewed by the domain expert to verify its tag. After collecting a set of tags, *supervised* anomaly detection [49] can be leveraged in

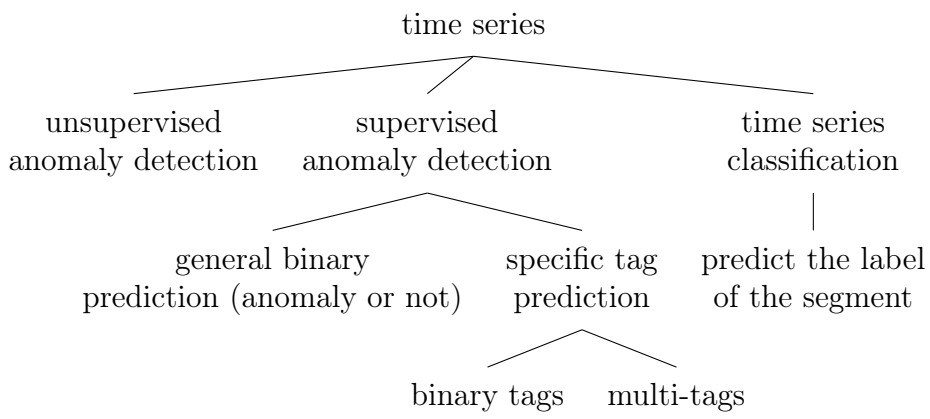


Figure 5-2: Paradigm of some time series tasks using machine learning.

the next iterations. Figure 5-2 illustrates the distinction between some of the several concepts pertaining to time series tasks. We note that there are many other tasks that are not addressed in this thesis such as time series forecasting, change point detection, etc.

5.2 Unsupervised Anomaly Detection Methods

In this section, we go through the technical details of three deep learning models: Long Short-Term Memory Networks, Auto-Encoders, and Generative Adversarial Networks. The objective of all these models is to generate an "expected" signal, as illustrated in 3. We continue by describing several approaches for calculating the error signal.

5.2.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory networks are a type of Recurrent Neural Network (RNN) and work well with sequential data [34]. As pointed out in [37], the natural properties of LSTMs make them capable of learning the relationships between past and current values, as well as correlations between different channels.

The model takes as input X which consists of prior telemetry values for all channels of window size l. The model outputs $\hat{y}$, a predicted value for a specific channel

m for a given horizon h, where h represents how many values will be predicted in the future. We use a mean squared error loss to train the model. This process is repeated for every entry in the time series.

Input: $X \in \mathbb{R}^m$, $y \in \mathbb{R}$ where y is the channel to predict (e.g. channel m)

Output: $\hat{y} \in \mathbb{R}$

Loss: $\mathcal{L}(y, \hat{y}) = \sum (y_i - \hat{y}_i)^2$

$$X = \begin{bmatrix} x^1_{t-l} & \dots & x^1_{t-1} & x^1_t \\ x^2_{t-l} & \dots & x^2_{t-1} & x^2_t \\ \vdots & \ddots & \vdots & \vdots \\ x^m_{t-l} & \dots & x^m_{t-1} & x^m_t \end{bmatrix}_{m \times l} \qquad y = \begin{bmatrix} x^m_{t+1} \\ x^m_{t+2} \\ \vdots \\ x^m_{t+h} \end{bmatrix}_{h \times 1} \qquad \hat{y} = \begin{bmatrix} \hat{x}^m_{t+1} \\ \hat{x}^m_{t+2} \\ \vdots \\ \hat{x}^m_{t+h} \end{bmatrix}_{h \times 1}$$

Notice if $h > 1$, then each value in the future will be predicted more than once. We aggregate these values (e.g. mean value) such that we end up with a one-dimensional vector.

5.2.2 Auto-Encoder (AE)

Auto-Encoder models are sequence-to-sequence based architectures that use an "encoder" network to map a time series into the latent space, and then a "decoder" network to map it back into the original feature domain [61]. They are relatively easy and quick to train. In addition, an AE model is flexible enough to use any type of network for the encoder and decoder, including Dense layers, LSTMs, GRUs, etc. Malhotra et al. [52], suggests that LSTMs are ideal for both encoder and decoder networks in a time series setting.

Similar to the prediction model, an AE takes as an input X, which consists of prior telemetry values for all channels of window size l. The model then outputs $\hat{X}$, which is a reconstruction of X for a specific channel m. In the matrix below, we highlight the channel that was reconstructed. The loss function is a mean squared error loss with respect to the selected channel to be reconstructed.

Input: $X \in \mathbb{R}^m$

Output: $\hat{X} \in \mathbb{R}$ where $\hat{X}$ is a reconstructed channel (e.g. channel m)

Loss: $\mathcal{L}(X, \hat{X}) = \sum (X_i^m - \hat{X}_i)^2$

$$X = \begin{bmatrix} x_{t-l}^1 & \dots & x_{t-1}^1 & x_t^1 \\ x_{t-l}^2 & \dots & x_{t-1}^2 & x_t^2 \\ \vdots & \ddots & \vdots & \vdots \\ x_{t-l}^m & \dots & x_{t-1}^m & x_t^m \end{bmatrix}_{m \times l} \qquad \hat{X} = \begin{bmatrix} x_{t-l}^m \\ \vdots \\ x_{t-1}^m \\ x_t^m \end{bmatrix}_{l \times 1}$$

5.2.3 Generative Adversarial Network (GAN)

A popular deep learning approach for reconstructing data is Generative Adversarial Networks (GAN) [30]. The idea behind a GAN model is that the generator $\mathcal{G} : Z \to X$ constructs a fake time series based on some random latent variable in order to fool the critic (commonly known as the discriminator) $\mathcal{C}_x$, which tries to differentiate "fake" examples from "real" ones in the original domain. In its generic form, it is defined as a min-max game:

$$\min_{\mathcal{G}} \max_{\mathcal{C}_x} V(\mathcal{G}, \mathcal{C}_x) = \mathbb{E}_{x \sim \mathbb{P}_x}[log\, \mathcal{C}_x(x)] + \mathbb{E}_{z \sim \mathbb{P}_z}[log\, (1 - \mathcal{C}_x(\mathcal{G}(z)))] \tag{5.1}$$

To account for temporal dynamics present in time series data, we introduce an encoder $\mathcal{E} : X \to Z$ to map features to the structure of the latent space. We train the encoder adversarially through another critic $\mathcal{C}_z$ to distinguish between random latent samples and encoded samples $V(\mathcal{E}, \mathcal{C}_z)$. A similar approach to learning an embedding network was introduced in [71].

In inference time, we use the GAN model in a similar fashion to AE: $x \to \mathcal{E}(x) \to \mathcal{G}(\mathcal{E}(x)) \approx \hat{x}$. To encourage mapping a sample from x to $\hat{x}$, we add a cycle consistency loss by minimizing the $L2$ norm of the difference between the original and the

reconstructed samples:

$$V_{L2}(\mathcal{E}, \mathcal{G}) = \mathbb{E}_{x \sim \mathbb{P}_x}[\|x - \mathcal{G}(\mathcal{E}(x))\|_2] \tag{5.2}$$

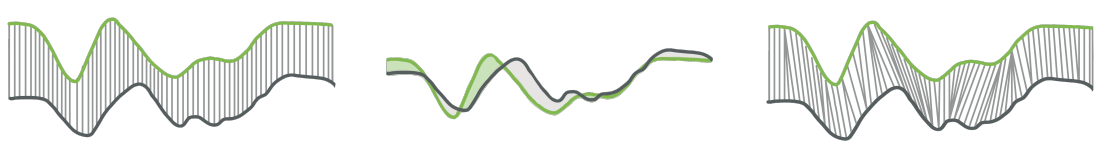
The overall objective then becomes a combination of all the aforementioned objectives:

$$\min_{\mathcal{G},\mathcal{E}} \max_{\mathcal{C}_x,\mathcal{C}_z} V(\mathcal{G}, \mathcal{C}_x) + V(\mathcal{E}, \mathcal{C}_z) + V_{L2}(\mathcal{E}, \mathcal{G}) \tag{5.3}$$

The **input** and **output** dimensions of a GAN network are identical to those of an AE model as it aims to reconstruct X. For both $\mathcal{G}$ and $\mathcal{E}$, Geiger et al. [28] implement a two layer LSTM architecture, and for the critics $\mathcal{C}_x$ and $\mathcal{C}_z$ they use a sequence of convolutional layers.

5.2.4 Calculating the Error Signal

All these models produce a one-dimensional vector, which we will refer to as $\hat{y}$, that aims to mimic the underlying distribution of a channel in $y \subseteq X$, which we will refer to as y. 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.



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

Figure 5-3: High-level depiction of (a) point error (b) area error (c) dynamic time warping.

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:

$$e^{(t)} = |\, y^{(t)} - \hat{y}^{(t)} \,| \tag{5.4}$$

All error values at each time step t are concatenated into a one-dimensional vector $\mathbf{e} = [e^{(1)}, e^{(2)}, \ldots, e^{(t)}, \ldots, e^{(n)}]$ where n is the length of the time series.

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:

$$e^{(t)} = \frac{1}{2l} \left| \int_{t-l}^{t+l} y^{(t)} - \hat{y}^{(t)} dy \right| \tag{5.5}$$

Similarly, we concatenate all values to form $\mathbf{e}$.

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. 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 kth element of that warp path mapping x_i to y_j. The optimal path is given by

$$\mathbf{e} = DTW(y, \hat{y}) = W^* = \min_W \left[\sqrt{\sum_{k=1}^{K} w_k} \right] \tag{5.6}$$

Given two time series sequences, we want to find an m-to-n mapping that illustrates which value from series A corresponds to which value in series B [9]. 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

$$\gamma(i, j) = d(x_i, y_j) + min\{\gamma(i-1, j), \gamma(i, j-1), \gamma(i-1, j-1)\}$$

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 $b - b' \geq 0$.

Critic Error

In the special case of GANs, we can incorporate the critic's judgement as part of the error signal. Recall that the critic $\mathcal{C}_x$ is trained to distinguish between real and fake examples. Following this logic, the critic can also indicate if it received any anomalous sequences.

We can use a weighted average between the critic error and any other distance-based error f, such as point, area, or DTW. We can formulate the general notation of an anomaly for some weight $\lambda \in [0, 1]$ as:

$$e^{(t)} = \lambda\, \mathcal{C}_x(\hat{y}^{(t)}) + (1 - \lambda)\, f(y^{(t)}, \hat{y}^{(t)}) \tag{5.7}$$

After that we can use a threshold on the error signal to segregate normal values from anomalous ones.

5.2.5 Discussion

In the previous description of anomaly detection methods, notice how *known ground truth* anomalies were not part of any step in the model's detection process. All methods work in a self-supervised manner where the model learns from the signal itself without manually collected labels [26]. All methods have the same objective of generating another signal to help locate anomalies.

5.3 Evaluation Metrics

What makes these anomaly detection models different from each other? Users might want to make an educated selection to best fit a certain time series and anomaly type. In order to choose between different pipelines, we need to measure the relative performances of different methods. 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 metrics include *precision*, *recall* and *F1-score*. However, as noted by Tatbul et al. [62], 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 $T = \{(t_s, t_e)\}_{i=1}^{m}$ and predicted anomalies $P = \{(t_s, t_e)\}_{i=1}^{n}$ where t_s and t_e represent the start and end timestamps of an anomaly respectively, we define specific methods to enable the fair computation of metrics without restrictions on the data: weighted segment and overlapping segment. Figure 5-4 shows an example of two difference sets for ground truth and detected anomalies. We will use this example in subsequent illustrations to showcase the mechanics of the two algorithms.

5.3.1 Weighted Segment

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

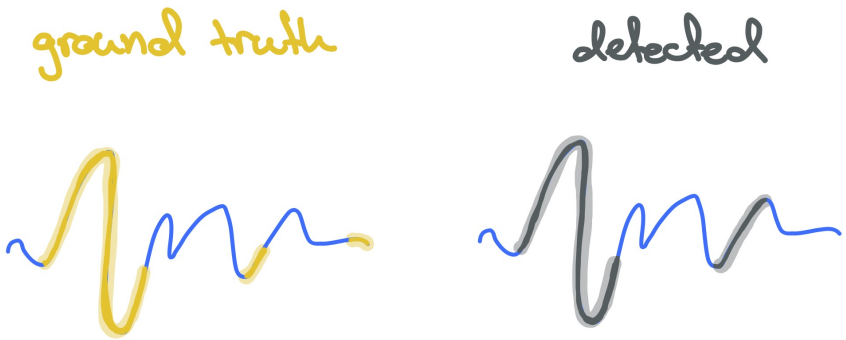


Figure 5-4: Example of known *ground truth* anomalies and *detected* anomalies via anomaly detection method.

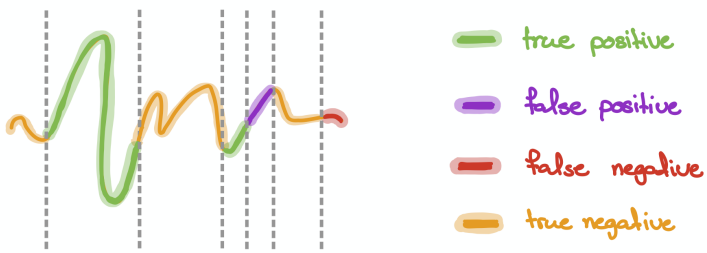


Figure 5-5: Weighted segment illustration. Each vertical line depicts a partition. Each partitioned segment will be evaluated into its respective true positive, false positive, false negative, and true negative values based on the comparison of ground truth and detected segments.

each edge, we record whether it was observed in the T set or P set and record it in $\tilde{T}$ and $\tilde{P}$ 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 weigh each segment by its time range. Figure 5-5 showcase how the weighted segment is applied to the example shown in Figure 5-4. This approach is valuable when the user aims to detect the exact segment of the anomaly. In case where the user's signal is inherently regularly sampled, this approach is equivalent to a sample-based evaluation.

```
Input: ground truth anomalies T, predicted anomalies P
Output: confusion matrix M
begin
    E ← T ∪ P                                   // all t_s and t_e timestamps
    T̃ ← ∅, P̃ ← ∅, W ← ∅
    E ← sort(E)                                 // sort timestamps from small to large
    s ← pop(E)                                  // the first timestamp
    while E ≠ ∅ do
        e ← pop(E)
        ti ← (t_s, t_e)                         // create a time interval (t_s, t_e)
        W ← W ∪ {t_e − t_s}
        T̃ ← T̃ ∪ {overlap(ti, T)}                // check if ti in ground truth
        P̃ ← P̃ ∪ {overlap(ti, P)}                // check if ti in predicted
        s ← e
    end
    M ← confusion_matrix (T̃, P̃, 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.

5.3.2 Overlapping Segment

Overlapping segment is a more lenient evaluation approach. It is inspired by the evaluation method of Hundman et al. [37], which rewards the model if it alerts the user to even 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.

Figure 5-6 showcase how the overlapping segment is applied to the example shown in Figure 5-4. Moreover, the overlapping segment approach does not account for true

```
Input: ground truth anomalies T, predicted anomalies P
Output: confusion matrix ⟨tp, fp, fn⟩
begin
    U ← ∅                                          // bookkeeping unmatched events
    tp ← 0
    while T ≠ ∅ do
        t ← pop(T)
        for p ∈ P do
            if overlap(t, p) then
                tp ← tp + 1                        // matched
            end
        end
        if unmatched(t) then
            U ← U ∪ {t}                            // add to unmatched
        end
    end
    tn ← |U|
    fp ← |P| − tp
    return ⟨tp, fp, fn⟩
```

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.

negatives (TN) and is invariate to time. Moreover, in a case where an entire time series is determined to be anomalous, this method will return high metric values. Therefore, it is important to regulate the length of the anomalies in this evaluation approach.

5.4 Known Anomalies

Once the first phase of anomaly detection using machine learning is complete, we require human expertise to validate and annotate the identified anomalies, which may or may not be based on ground truth. We provide a visualizing subsystem so that experts can view time series and their respective predicted anomalies (see Section 6.6). Users can interact with this system by *confirming, modifying, removing, searching,* and *discussing* events.

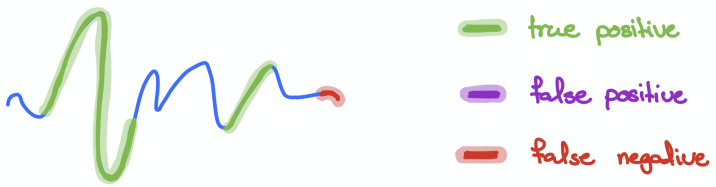


Figure 5-6: Overlapping segment illustration. Each detected anomaly will be assigned to either true positive or false positive, and missed ground truth anomalies will be assigned to false negative.

Why is the logic important? Based on these annotations, future predictions regarding the same or similar signals can be improved. These annotations effectively incorporate domain-specific knowledge into our anomaly detection framework. Then, we can start mixing supervised and unsupervised models with iterated learning. This reduces both false positives and false negatives and helps improve future predictions.

Chapter 6

System Design and Architecture

Many algorithms have been developed to address the task of time series anomaly detection, ranging from statistical methods to machine learning techniques [35, 17, 29, 32]. Despite the existence of these algorithms, the problem is only partially solved. There is a fundamental gap between scientists and engineers that precludes these methods from being used in a real world setting. To state the problems more specifically: *How do we enable an end user to use complex methods? How do we select an anomaly detection method from the array of available pipelines?* and *How do we maintain a constant knowledge base across a monitoring team?* Moreover, after detecting and annotating anomalies, *how do we harness the analysis and decisions of experts and feed them back into the system?* We aim to address these question in the *Orion* framework.

Orion is a system comprised of a series of components that can perform all the necessary detection steps, from composing pipelines to annotating anomalies. A user aiming to detect anomalies in their time series data can either select an existing, verified pipeline from our collection, or create their own. The system is backed by an integrated database that maintains the results of each detection [1]. We supplement *Orion* with an interactive visual interface, enabling users to intuitively investigate and annotate any detected anomalies.

[1]The database has now been migrated into its own repository under https:/github.com/sintel-dev/sintel

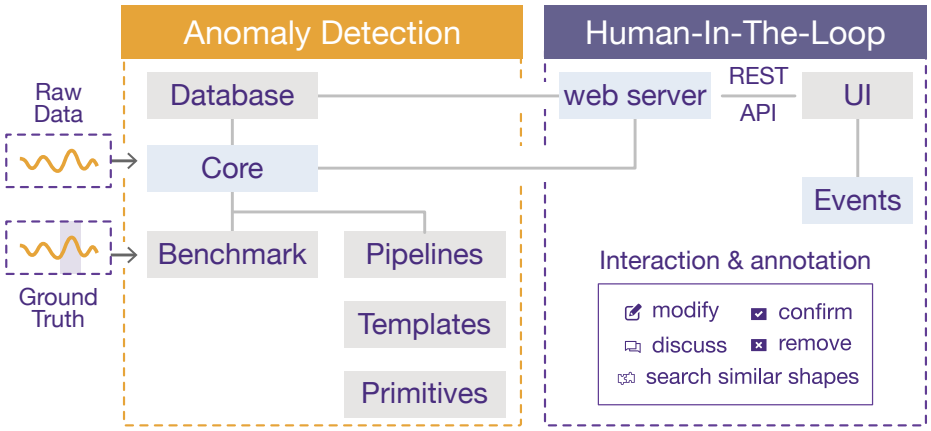


Figure 6-1: *Orion* consists of two subsystems. The anomaly detection subsystem detects anomalies through a `python` library, which are then annotated by users using the human-in-the-loop subsystem of MTV.

Orion's components are categorized into two main subsystems: anomaly detection and human-in-the-loop using **MTV** [2], with a communication channel between them (Figure 6-1). We first define the machine learning stack (Section 6.1), which contains primitives, templates, and pipelines. Then we introduce the core interaction (Section 6.2) which is the main entry point that allows users to select and train pipelines as well as to predict and store anomalies. This is followed by a description of the hyperparameter tuning component (Section 6.3). Due to the nature of anomaly detection pipelines, we include a benchmark utility (Section 6.4) to compare the quality and computational performance of different pipelines. We supplement the framework with a database (Section 6.5) to keep a persistent state of information. To incorporate human knowledge, we use the visualization subsystem to allow experts to annotate events and utilize human annotations through the feedback loop (Section 6.7).

6.1 Machine Learning Stack

Prior to diving into the components of the system, we need to convert these machine learning algorithms into standardized end-to-end programs, which we call *pipelines*. In this subsection, we define pipelines and their building blocks.

[2] `github.com/sintel-dev/MTV`

Anomaly detection pipelines take a univariate or a multivariate signal as input and use it to generate an array of intervals $A = [(t_s^1, t_e^1), \cdots, (t_s^k, t_e^k)]$ representing the anomalies discovered. In most cases, users lack labels for their data. Therefore, our pipelines service both unsupervised and supervised approaches, and are built end-to-end such that *Orion* is agnostic as to which pipeline is executed. To understand the composition of pipelines, we describe their basic building blocks, or *primitives*, below.

6.1.1 Primitives

Primitives are reusable software components [59]. 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 has associated metadata including annotations, such as the name of the primitive, the description and documentation link, and the engine category. As illustrated in Table 4.1, *Orion* covers 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 modeling it. 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 an *expected* signal (see Figure 3-2).

Post-processing

After generating an *expected* signal, we use discrepancies between the expected 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.

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.

6.1.2 Pipelines

Pipelines are end-to-end programs composed of primitives. Each pipeline is computed into its respective computational graph, similar to the examples shown in Figure 6-2. 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 [3, 59].

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 $T = \langle V, E, \Lambda \rangle$, 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 [59]. 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 create and manipulate pipelines easily, enabling their use 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.

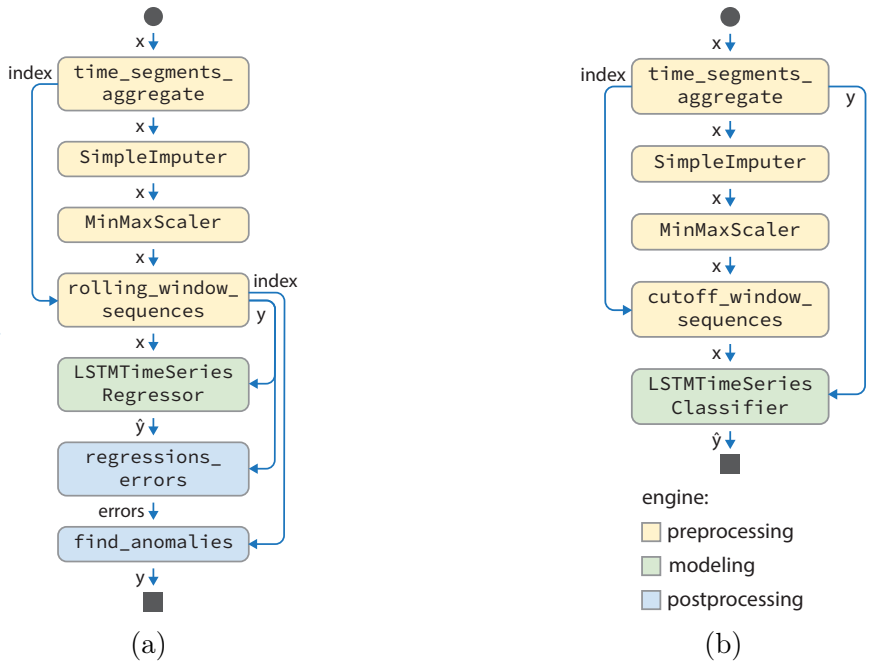


Figure 6-2: Graphic representations of (a) an unsupervised pipeline with a Long Short-Term Memory (LSTM) network. (b) a supervised counterpart of the LSTM network for time series classification.

Dissecting LSTM Pipeline Example The pipeline in Figure 6-2a uses a Long Short-Term Memory (LSTM) network to predict data values at future timestamps. We first take a raw signal $\mathbf{x}$ and feed it into the `time_segments_aggregate` to produce $\mathbf{x} = [x^1, x^2, \ldots, x^T]$ where the time intervals between x^{t-1} and x^t are equal. Then we scale the data $\mathbf{x} \in [-1, 1]$ and impute missing values using the mean value of the signal. After that, we create the training window sequences. The processed signal is now ready to train the double-stacked LSTM network. Once the network is trained, we generate the predicted signal and compute the discrepancies using `regression_errors`, which is an absolute point-wise difference $|\hat{\mathbf{x}} - \mathbf{x}|$. Lastly, we use a dynamic threshold on error values to find anomalous regions [37]. Customizing pipelines is fairly easy. Users can configure a primitive or even replace it with another. For example, to use z-score normalization, users can swap `SimpleImputer` with `StandardScaler`.

```python
from sintel import Sintel
from sintel.data import load_signal

train_data = load_signal('S-1-train')
sintel = Sintel(
    pipeline='lstm_dynamic_threshold'
)

# train the pipeline
sintel.fit(train_data)

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

# detect anomalies
anomalies = sintel.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 6-3: Usage with python SDK. (a) end-to-end anomaly detection pipeline. First the user loads the data either by using `load_signal` or externally. Then the user select the desired pipeline for detection. In this example, we use `lstm_dynamic_threshold`. 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.

AD Pipeline Hub *Orion* stores a collection of end-to-end anomaly detection pipelines that work with state-of-the-art methods. As of the time of writing, we have incorporated `ARIMA` [55], `LSTM DT` [37], `LSTM AE` [52], and `TadGAN` [28]. Moreover, we provide "pipelines" that can connect to existing anomaly detection services, such as MS Azure [56]. This set of pipelines is easily extendable and can be expanded further.

6.2 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 APIs, allowing users to execute end-to-end processes. Given a signal X, we want to obtain a set of detected anomalies. With *Orion*, this is straightforward. First, the user loads a signal which 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(data)`. To detect anomalies, the user then executes `orion.detect(data)` to produce a set of possible anomalies. Users can also use the evaluation mechanisms defined in Section 5.3 to view the performance of a pipeline if the ground truth anomalies are present. Figure 6-3 shows an example code for this process.

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 [13].

6.3 Hyperparameter Tuner

Hyperparameter tuning is instrumental to ML systems [10, 11]. To tune pipelines automatically (Figure 6-4), we integrate `BTB` [3], an open-source and extensible framework with black box Bayesian Optimization [59]. 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 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 the search until our budget runs out, or we have reached the optimal value. Algorithm 3 depicts the general flow of optimization in *Orion*.

Orion's tuners are customized for use in two different settings, as shown in Figure 6-5. In an *unsupervised* setting, a user only tunes the sub-pipeline that attempts to generate the signal closest to the original signal. To achieve this, users specify the pipeline template and evaluation metrics, such as `MAPE`, `MAE`, etc. for their objective

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

```python
from sintel import Sintel

# initialize data and ground_truth
# ...
sintel = Sintel(
    pipeline="lstm_dynamic_threshold"
)

# supervised
sintel.tune(
    data=data,
    anomalies=ground_truth,
    scorer='f1'
)

# unsupervised
sintel.tune(data=data, scorer='mse')
```

Figure 6-4: 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.

function. In a *supervised* setting, we provide objective functions that evaluate the efficacy of the pipeline in detecting known anomalies, such as `F1`. Note that in the supervised setting, a ground truth set of anomalies must be defined. Tuning can be used to fine-tune a pipeline with expert annotations. Depending on the tuning setting, some engines may not need to be tuned. Recall in Section 6.1, our primitives are annotated, enabling *Orion* to automatically pull hyperparameters with respect to the set of primitives needed.

6.4 Benchmark Framework

The availability of benchmarking is one of the key advantages of our framework. As shown in Table 4.1, many existing frameworks develop algorithms for AD, but lack an out-of-the-box benchmark. With *Orion*'s evaluation metrics and pipeline hub, we are able to thoroughly compare these methods on multiple datasets, through a single command — `benchmark`. Figure 6-6 illustrates our benchmark API. The benchmark is designed to measure two main aspects: quality and computational performance.

```
Input: template T, dataset D, scorer function f, budget B.
Output: best hyperparameter λ*
begin
    init Tuner                                          // Bayesian Tuner from BTB
    s* ← +∞, λ* ← ∅
    while B > 0 ∧ s* ≠ f* do
        λ ← Tuner.propose(T)                            // propose a set of hyperparameters
        P ← (T, λ)
        s ← cross_validate(f, P, D)
        Tuner.record(λ, s)                              // update tuner
        if s < s* then
            s* ← s
            λ* ← λ
        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.

6.4.1 Quality performance

We define quality evaluation as an assessment of how well the pipeline has detected ground truth anomalies. We extend beyond sample-based evaluation metrics such as precision and recall as defined in `scikit-learn`, and introduce our pipeline evaluation metrics as detailed in Section 5.3. The benchmark is extensible, and users can easily add new metrics and evaluation criteria.

6.4.2 Computational performance

Deep learning is notorious for its expensive computational needs. In addition to quality performance, it is important to compare different pipelines' computational costs. To accomplish this, we measure the total time necessary for each pipeline's execution, including how much time is required to train the pipeline (*training time*) and how much time the pipeline takes to turn an input into an output (*pipeline latency*). Additionally, we record how much time each individual primitive needs for both phases. We also measure memory consumption for pipelines and primitives. Moreover, deep

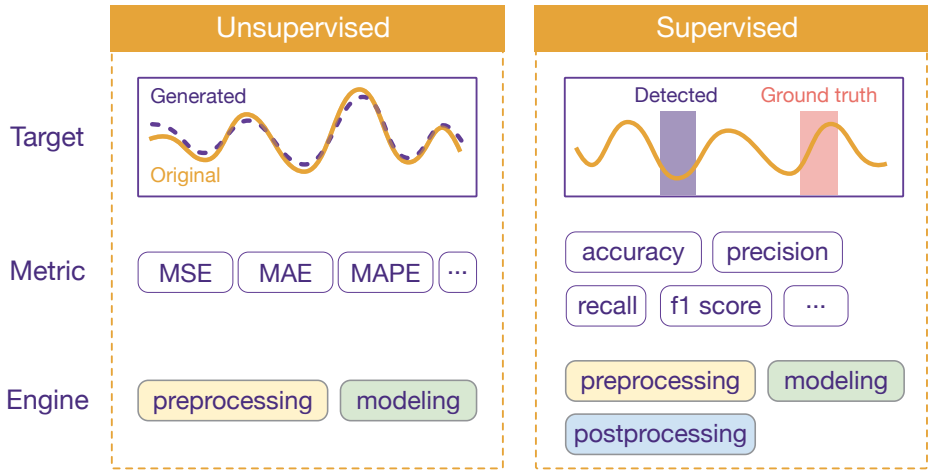


Figure 6-5: Hyperparameter tuning in two conditions: (1) unsupervised, where the goal is to optimize the signal generated by the ML model, and (2) supervised, where our goal is to produce anomalies that best match the ground truth set.

```
from sintel import benchmark

pipelines = ['arima', 'lstm_dynamic_threshold', '...']
datasets = ['NAB', 'NASA', '...']
metrics = ['f1', 'accuracy', '...']

benchmark(pipelines=pipelines,
          datasets=datasets,
          metrics=metrics,
          rank='f1')

# >>>
#                  pipeline  rank  accuracy  elapsed     f1         precision   recall
# 0  lstm_dynamic_threshold     1  0.986993  915.958132  0.846868   0.879518     0.816555
# 1                    arima     2  0.962160  319.968949  0.382637   0.680000     0.266219
```

Figure 6-6: Benchmarking usage with python SDK. `benchmark` allows users to compare the performance of many pipelines via one command.

learning libraries are constantly optimizing the performance of their framework which consequently directly improves the performance of modeling primitives. Keeping this in mind, we are more interested in inspecting which in-house pre-processing and post-processing primitives are incurring additional cost. The inclusion of computational monitoring allows us to identify which primitives cause bottlenecks within the pipeline and to improve them further.

6.5 Persistent Knowledge Base

In real-world settings, it is often necessary to continuously record data, including expected anomalies. It is also important to document events so that information is not lost, which can lead to repetitive and unnecessary investigations. Proper logging of signals and their corresponding anomalies will greatly improve users' understanding of where these anomalies came from, and tracing back decisions will become easier as well. In *Orion*, we use a `mongoDB` database with an extensive schema to fill this gap. We chose `mongoDB` due to: (1) its flexibility over application domains; (2) its interoperability with *Orion*'s visualization tool — a web app developed using JavaScript; (3) its extensiblility through community add-ons.
Incorporation of the database enables users to do the following:

1. use anomalies that exist within the database to annotate new signals, in order to avoid rerunning the same model and wasting computational time.

2. document the history of anomalies as users become more experienced.

3. maintain a growing store of information as multiple users annotate signals.

A high-level depiction of the entities and relationships within the database is shown in Figure 6-7. Figure A-2 provides a detailed UML view. Through the database, we can retrieve information easily, constantly trace what is happening, and create a valuable knowledge base for both new and experienced users.

6.6 Visualization and Anomaly Annotation

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 an interactive visualization tool [50], presented in Figure 6-8. This tool allows subject matter experts to undertake an investigation process, and to annotate and discuss events.

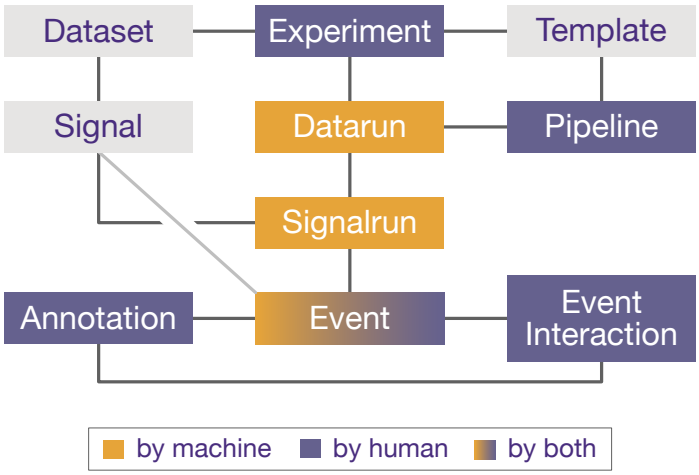


Figure 6-7: High-level database schema.

The visualization subsystem 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 allows 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.

6.6.1 Shape-Matching

Annotating events can overwhelm a user due to the sheer number of events that need expert review. To enhance the annotation experience and make users more confident in their decisions, we propose a shape-matching algorithm to compare events in time

Figure 6-8: Snapshot of MTV — the visualization component of *Orion*. Multiple signals are displayed at the top as an overview, and the detailed view of one selected signal is shown at the bottom. The right panel displays how users assign tags and comment on the signal of interest.

series data.

Use Cases.

To enhance the annotation process for large-scale tasks such as annotating events in time series data, we can use shape matching for three different cases:

Tag Propagation. Once an expert confirms that an annotated event is indeed an anomaly, they are inclined to search for other segments that are similar to the confirmed event, but have not yet been flagged. These similar segments are likely to receive the same annotations as the confirmed one. Using shape-matching, users can propagate the tag (i.e. anomalous status) to these segments.

Mitigating False Alarms. After events have been properly annotated, we are able to mitigate false alarms (false positives) by leveraging the shape-matching algorithm. Consider an event that experts may think is unworthy of exploration – tagged as

"normal" for example. We ask the system to prune out events that are similar to it from the final output.

Decision support. Sometimes the status of an anomaly is ambiguous, and choosing a proper tag is difficult. In this case, an expert can use shape-matching to find similar shapes (non-anomalous segments) to this event. By comparing the differences between them, experts can quickly clarify why the anomaly detection algorithm identified this segment as anomalous.

Algorithm.

The algorithm is performed in a single signal based manner and outputs a set of candidate shapes in a univariate signal $\mathbf{x}$ that are similar to a particular sequence $\mathbf{s}$, typically $\mathbf{s} \subseteq \mathbf{x}$. The algorithm is designed to factor in the constraint that returned segments do not overlap in time with each other or any existing events. This constraint is important as without it, annotators will feel confused by overlapping events. We adopt a sliding window approach to generate sub-sequences of $\mathbf{x}$, then compare them to s through a similarity measure $f(\cdot, \cdot)$. We chose f to be the total cost of finding the optimal mapping between two sequences. Similarity measurement can be any function that determines the distance between two time series sequences, such as Euclidean Distance, Dynamic Time Warping (DTW) [9, 44], etc. Euclidean distance tends to find exactly matched shapes and is very sensitive to time, while DTW can tolerate a certain level of shifting.

The full algorithmic process is described in the following pseudo-code (algorithm 4). At each checkpoint t we attempt to include the current most similar shape $\mathbf{c}$ in the candidate subset. Prior to that, we check whether or not $\mathbf{c}$ overlaps with a preexisting candidate shape; if so, we keep the most similar between the two. This procedure will return a set of non-overlapping candidate shapes S that are most similar to $\mathbf{s}$.

Complexity Analysis.

In theory, the time complexity of Algorithm 4 is $O(nk)$, where n represents a length of $\mathbf{x}$ which denotes the total time it takes to iterate over the time series, and k is the

```
Input: signal x = {x_1, x_2, ..., x_n}
       sequence s = {s_1, s_2, ..., s_m} and m < n
Output: candidate sub-sequences S
begin
    c = ∅                                          // placeholder
    t = m                                          // checkpoint of length of s
    for i = 1 to n − m do
        x̃ = {x_i, x_{i+1}, ..., x_{i+t}}           // sub-sequence of x
        if f(x̃, s) < f(c, s) then
            c ← x̃                                  // update
        end
        if i > t then
            if overlap(S, c) then
                v ← arg overlap(S, c)              // overlapping sequence
                c ← argmin [f(c, s), f(v, s)]      // keep most similar
            end
            S = S ∪ {c}                            // add candidate shape
            c = ∅                                  // reset
            t = t + m
        end
    end
return S
```

Algorithm 4: Shape Matching Algorithm. We find a collection of non-overlapping sub-sequences in $\mathbf{x}$ that are similar to $\mathbf{s}$.

time needed to calculate the cost $f(\cdot, \cdot)$ between two sub-sequences of length m. In the case of Euclidean distance k is linear $O(m)$, while in the case of vanilla DTW, it runs in quadratic time $O(m^2)$. We consider enabling faster variations of DTW, such as the Sakoe Chiba band and the Itakura parallelogram, which make the time complexity dependant on the band width and slope of the parallelogram respectively [58, 39]. In real-world scenarios, Euclidean distance is sufficient and we expect $m \ll n$; thus, the algorithm runs in close to linear time. In Section 7.9, we experiment with the runtime of Algorithm 4.

6.7 Feedback

ML models detect anomalies that generally show up as unexpected temporal patterns within certain time periods. In order to allow experts to annotate all detected anomalies, the number of anomalies recommended by a model should be controlled. In our example scenario, the satellite operations team has to review all unexpected behaviors, which cause around 20 alarms per day. A failure to respond to a potential hazard may result in satellite traffic jams or even a crash, leading to financial loss and negative brand impact.

The experts' knowledge is fed into a semi-/supervised pipeline to calibrate the output of automated detection. After developing a careful understanding of what users needed, we based the refreshment process of the semi-/supervised pipeline on application-specific batch processing of annotations. There is a high variability between domains in terms of frequency of anomalies. Based on the satellite company's particular needs, we concluded that a weekly update is sufficient. The schedule is a parameter in the environment and can be modified to suit users.

Chapter 7

Evaluation

In the following chapter, we introduce several experiments that demonstrate the use, performance, and effectiveness of *Orion*. We experimentally evaluate its benchmarking suite, its tuning system, and its ability to close the loop of an anomaly detection process. Moreover, we test the shape-matching algorithm and report the runtimes under various conditions.

7.1 Datasets

Our experiments utilize three publicly-accessible time series datasets with known anomalies. First, we use two sets of spacecraft telemetry signals — MSL and SMAP — provided by **NASA**,[1] which contain a total of 80 signals and over 100 known anomalies. Second, we use **Yahoo S5**,[2] a dataset dealing with production traffic in Yahoo computing systems. Most subsets within this dataset have been synthetically created. Overall, this dataset contains 367 signals and 2,152 anomalies. Finally, we use the Numenta Anomaly Benchmark (**NAB**) dataset,[3] which has 45 signals with 94 anomalies. Table 7.1 summarizes each of the datasets used. In total, there are 492 signals with 2349 anomalies.

[1]NASA: https://github.com/khundman/telemanom
[2]YAHOO S5 https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70
[3]NAB: https://github.com/numenta/NAB

Dataset	# Signals	# Anomalies	Avg. Signal Length
NAB	45	94	6088
NASA	80	103	8686
YAHOO	367	2152	1561

Table 7.1: Dataset Summary: 492 signals and 2349 anomalies.

7.2 Experimental Setup

We first run the benchmark end-to-end on all available pipelines, using the described datasets. We use a total of 6 pipelines – LSTM DT [37], ARIMA [55], TadGAN [28], LSTM AE [52], and Dense AE – and a pipeline that uses the MS Azure anomaly detection service [56]. We evaluate the quality and computational performances of the pipelines according to the evaluation metric described in Section 5.3. To run the benchmark, we use a private HPC cluster with 192GB of memory and two 24-core Intel Xeon CPUs.

7.3 Quality Performance

Table 7.2 shows the quality performances of currently available pipelines. The scores are calculated according to the *overlapping segment* approach described in Section 5.3. As the table shows, no single pipeline outperforms all other pipelines – each dataset has its own properties that make particular pipelines well- or ill-suited for it. For example, MS Azure [56] manages to locate anomalies in all datasets, but at the expense of introducing many false positives. This number of false alarms could be prohibitively time-consuming for an expert monitoring team to investigate.

	NAB			NASA			YAHOO		
	F1	precision	recall	F1	precision	recall	F1	precision	recall
LSTM DT	0.555 ± 0.12	0.452 ± 0.09	0.734 ± 0.22	0.559 ± 0.15	0.472 ± 0.18	0.700 ± 0.08	**0.772 ± 0.14**	**0.880 ± 0.14**	0.716 ± 0.21
Dense AE	0.599 ± 0.12	**0.666 ± 0.14**	0.547 ± 0.10	**0.636 ± 0.10**	**0.707 ± 0.09**	0.578 ± 0.11	0.431 ± 0.40	0.793 ± 0.19	0.383 ± 0.40
LSTM AE	**0.640 ± 0.09**	0.632 ± 0.08	0.654 ± 0.11	0.583 ± 0.11	0.568 ± 0.09	0.601 ± 0.14	0.524 ± 0.26	0.760 ± 0.14	0.469 ± 0.33
TadGAN	0.606 ± 0.10	0.536 ± 0.10	0.721 ± 0.15	0.556 ± 0.12	0.473 ± 0.10	0.673 ± 0.16	0.568 ± 0.19	0.698 ± 0.12	0.507 ± 0.26
ARIMA	0.514 ± 0.12	0.476 ± 0.14	0.566 ± 0.11	0.381 ± 0.07	0.383 ± 0.10	0.380 ± 0.05	0.757 ± 0.05	0.852 ± 0.14	0.714 ± 0.15
MS Azure	0.149 ± 0.11	0.086 ± 0.07	**0.892 ± 0.11**	0.041 ± 0.02	0.021 ± 0.01	**0.873 ± 0.09**	0.494 ± 0.21	0.352 ± 0.18	**0.912 ± 0.10**

Table 7.2: Unsupervised anomaly detection results (F1 score, precision, and recall) per pipeline on each dataset.

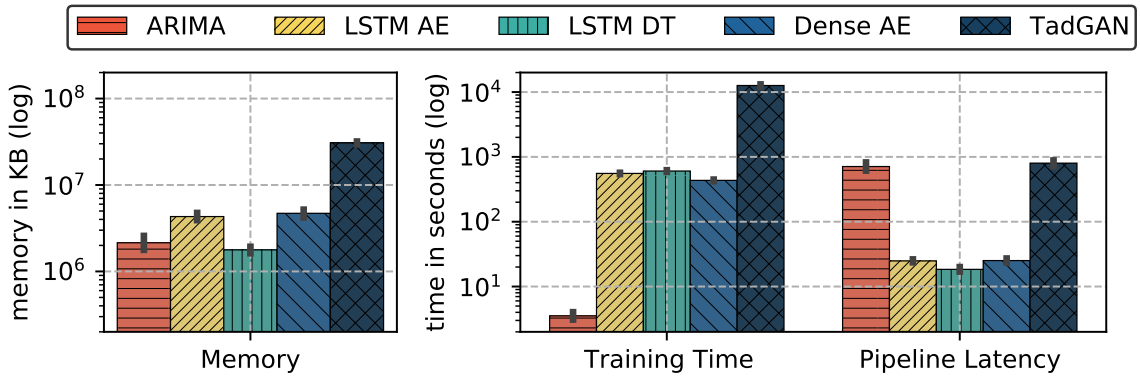


Figure 7-1: Pipelines' computational performance. First, we show the memory consumption by each pipeline. Next, we record the total time it takes to a train a pipeline end-to-end using `orion.fit`. In addition, we record the pipeline latency, which is analogous to how long the pipeline takes to produce an output using `orion.detect`.

7.4 Computational Performance

Figure 7-1 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 all 462 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 simultaneously. ARIMA — a popular statistical model — requires a similar amount of time as deep learning pipelines once both training time and pipeline latency are factored in. Users may be better served by different pipelines depending on their resources. Providing an assessment of the computational needs of each pipeline is necessary to help users determine which methods are the most appropriate for their particular case, especially when tackling deep learning models. This important evaluative feature is missing from other current systems.

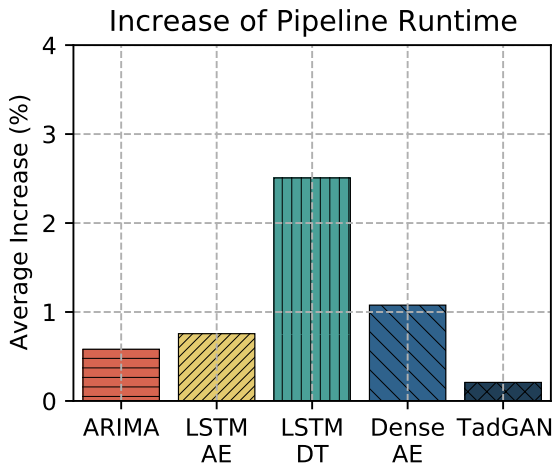


Figure 7-2: Difference in recorded runtime between stand-alone primitives and end-to-end pipelines.

7.5 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 it to the time needed to run a pipeline from beginning to end. We determine the runtime of each model on the entire dataset (462 signals). 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 7-2 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 a higher variability in runtimes.

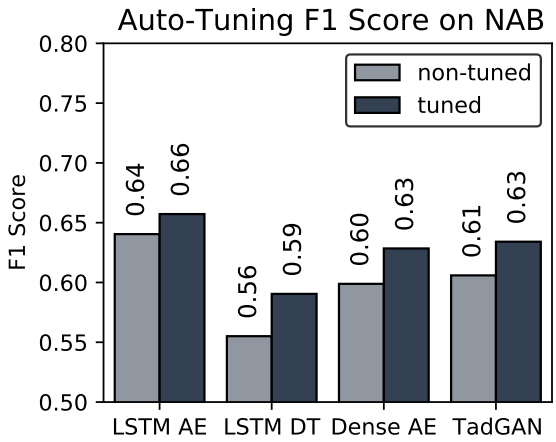


Figure 7-3: F1 Scores prior to and after tuning pipelines on the NAB dataset using a ground truth set of anomalies.

7.6 AutoML Performance

Orion improves pipelines using the hyperparameter optimization component introduced in Section 3.3. To test the efficacy of the platform's AutoML, we measure the F1 score per signal on the NAB dataset in a *supervised* manner. Pipelines improve 6.6% on average. Figure 7-3 shows the improvement in performance for each deep learning pipeline. Overall, 15% of hyperparameter changes were in the postprocessing engine, specifically in the `find_anomalies` primitive (see Figure 6-2a). This demonstrates the level of effectiveness of automated hyperparameter optimization that the user may expect to obtain.

7.7 Stability Testing

Continuous Integration (CI) tests have greatly increased the reliability of open-source code. When it comes to the evolution of a library, CI has alleviated some of the barriers of development, dependency, and release testing. However, it's difficult to see the direct effect on the anomaly detection task itself. We use benchmarking as a utility to test how *stable* these pipelines are and whether they are *reproducible*. When observing changes in the performance of pipelines, we can trace what caused it and what dependencies were involved.

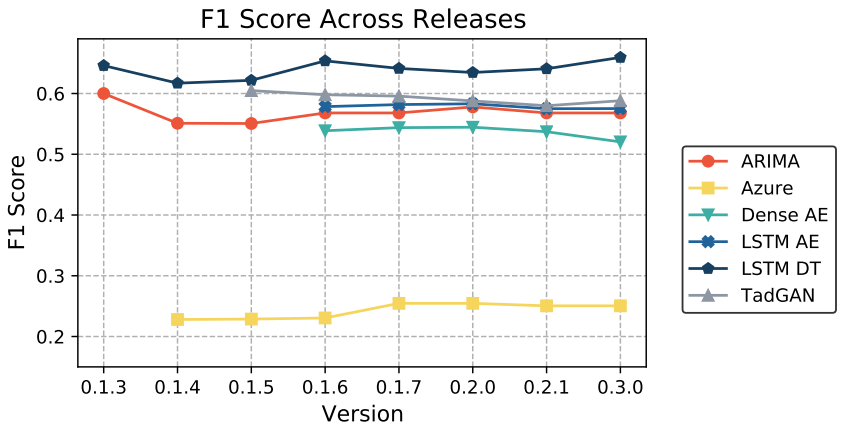


Figure 7-4: Average F1 Score of Orion release history across all datasets.

Figure 7-4 depicts the average F1 Score for each pipeline across all three datasets (NASA, Yahoo, and Numenta) that are currently available in our framework. Some pipelines were added at a later stage of the framework (e.g. Dense AE), indicating how extensible the framework is. Looking closely at Figure 7-4, we can see a drop in performance between version 0.1.3 and 0.1.4 tracing back to our change in aggregating the overall scores. More interestingly, we can see how MS Azure's service improved after version 0.1.6. In general, as deep learning methods vary in performance from one run to another, having recurring benchmarks over time helps us determine how stable a pipeline is. A history of F1 Scores for previous release versions is available in Appendix B.

7.8 Feedback Evaluation

To validate the impact of the user feedback loop, we assess its performance at integrating user annotations by simulating human actions. Here we assume that a user has the capacity to annotate $k = 2$ events at a single iteration and is capable of performing two types of annotations: adding or removing an event. The simulation stops when no events are left.

Our pipeline is a semi-supervised LSTM pipeline trained on sequences that were verified to be either anomalous or normal by the annotator. We warm-start the

simulation process with multiple initializations (all unsupervised pipelines). We use a 70/30 data split on the NAB dataset for training and testing. The training data encompasses 70 events, while the test data has 32 events. Results are depicted in Figure 7-5, where we observe that the performance of a semi-supervised pipeline surpasses the best-performing unsupervised pipeline once sufficient annotations have been obtained.

One drawback to depending solely on a semi-supervised pipeline is that before the pipeline becomes capable of identifying anomalies, its F1 Score is inferior to that of an unsupervised pipeline. Thus, we require a combination of unsupervised and semi-supervised pipelines to work synchronously. In addition, observing several flat segments in Figure 7-5, we note that some annotations may not help to improve detection. Given that all events must be annotated, it would be valuable to decide when to retrain the pipeline by estimating the benefit gain ahead of time, so as not to incur unnecessary costs.

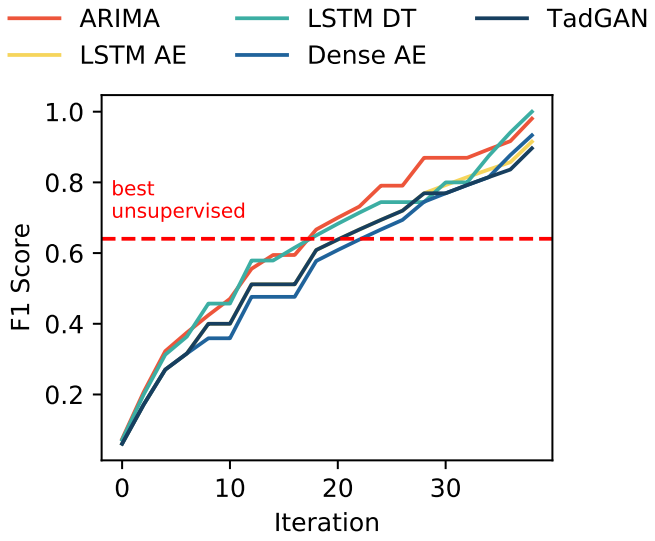
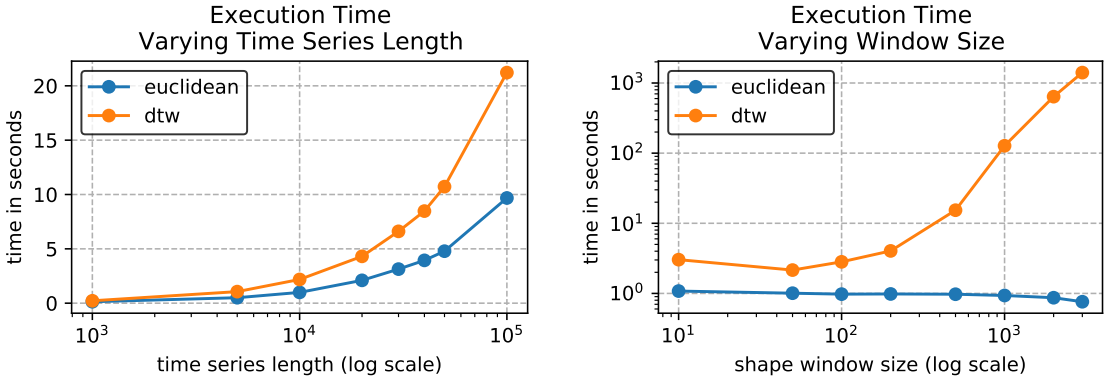


Figure 7-5: (Semi-supervised pipeline performance on NAB through simulating annotations from different starting points.

7.9 Shape Matching Analysis

To analyze the usability of Algorithm 4 in real applications, we record the runtime by varying the signal $\mathbf{x}$, the sequence $\mathbf{s}$, and the cost/distance function $f(\cdot, \cdot)$. Figures 7-6 report our experimental results on synthetic data when different distance metrics are applied. We generated signal $\mathbf{x}$ of length n of sine waves with added noise sampled from a Gaussian distribution $\mathcal{N}(0, 1)$. Sequence $\mathbf{s}$ size denotes the number of data points of the queried shape (m), and time series length indicates the total length of the signal (n).

In terms of evaluating the complexity of the algorithm, we measure the execution time (runtime) taken by each approach in seconds. Figure 7-6(a) showcases the time it takes to run the algorithm, fixing the window size at 100 and varying the time series length. We note that the time cost is less than 5 seconds when the length of the time series is less than $10,000$. Figure 7-6(b) illustrates the time it takes to run the shape matching algorithm by varying the window size and keeping the time series length fixed at 5000. As the window size increases, it becomes more computationally expensive to run (especially for DTW), which aligns with our analysis in section 6.6.1. In practice, we assume the anomalous window will be small (a few to dozens of data points). Hence, our algorithm is acceptable for real-time interactions in most scenarios.



(a) (b)

Figure 7-6: Time performance of shape-matching (a) with fixed window size 100 and varying time series length (log scale). (b) with fixed time series length 5000 and varying window size (log scale).

Chapter 8

Satellite User Study

We experimented using the framework with a satellite company's operations team. We report the results of our study here.

We demonstrate the usability of our framework through the real-world use case introduced in Section 2.1. We presented *Orion* to domain experts in the aerospace field, where detecting and annotating anomalies is their day-to-day operation. *Orion* was used by the satellite operations team to monitor thousands of signals and identify anomalies. We selected 16 real signals spanning a period of over 5 years from our collaborator's spacecraft telemetry database. These signals came from various subsystems and tracked metrics like electrical power, thermal temperature, etc. We recorded the usage and activity of 6 experts and conducted interviews to gain their qualitative feedback. In general, the experts greatly appreciated the system and valued its potential to enhance their efficiency for time series anomaly analysis.

	ML identified	ML missed
Normal	56	2
Problematic	11	6
Investigate	16	19
Total	83	27

Table 8.1: Collected tags from satellite use case.

We summarize the annotations collected from the study in Figure 8.1. A sample of 110 events tagged by humans were traced back a posteriori to determine whether or not the framework had also identified them as anomalous. The table depicts the events' detailed tags. The first column refers to the events identified by the ML model and then presented to domain experts for verification. The second column refers to the events that were missed by the ML model, but that experts marked as worth detecting. Among 110 events total, the team deemed 52.7% to be normal, confirmed 11 anomalies, manually added 6 events, and marked the rest as in need of further investigation. This illustrates the importance of incorporating human knowledge. Overall, experts valued our framework and believed in its ability to effectively identify and analyze anomalies.

Chapter 9

Discussion

In this chapter, we highlight some of the salient aspects of *Orion*, its limitations, and our attempts to create a framework that can be improved and evolve over time. We also touch on some interesting questions regarding the time series anomaly detection problem.

9.1 *Orion* Framework

9.1.1 Why do we need humans in the loop?

Unsupervised models are not perfect, particularly when past labels do not exist. Humans are necessary to iteratively guide even state-of-the-art models. In Table 8.1, we note that the ML missed $^{27}/_{110}$ events. When investigating why, we discovered that some events, such as lunar eclipses, have a normal shape, but should still be marked by experts for future reference. Meanwhile, some events, such as maneuvers, are actually considered normal by domain experts even though they have peculiar shapes. These issues are domain-specific, and it is difficult to find and understand them without a human annotator.

9.1.2 Addressing distribution shifts

Any pipeline's performance relies on *preprocessing*. For example, as domain experts pointed out, the ML detected events that were artifacts of aggregation. Also, in our experiments, we witnessed a drop in F1 scores using unsupervised pipelines when detecting anomalies in Yahoo's A4 subset. On investigation, we discovered that 86% of the signals in A4 contain a change point, which indicates a significant change in the data distribution. This could be overcome by preprocessing the signal using shift-elimination techniques such as decomposition [20, 24] as well as by segmenting signals using change point detection techniques [5, 64, 65]. Since *Orion*'s pipelines are modular, it is possible to add and integrate new preprocessing primitives.

9.1.3 Mixing supervised and unsupervised

In Figure 7-5, we observed that the semi-supervised models initially performed worse than the unsupervised models. To mitigate this problem, we can couple the models together – curating annotations as we collect them, so that we then have labeled data with which to train supervised models (pipelines). In subsequent iterations, we can run both supervised and unsupervised pipelines simultaneously as proposed in [66]. As we saw in Figure 7-5, the model did not always benefit from new annotations, necessitating such curation. We also note that pipelines will need to be updated when we observe drifts in the streaming data [68, 67].

9.1.4 Going beyond satellite operations

Although *Orion* has been designed and implemented to address the needs of a satellite company operations team, it can be adapted for other applications with different data volumes and efficacy needs. In fact, we collaborated with one of the world's largest electric utility companies to inform our design. They have subsequently put *Orion* to use, using application-specific pipelines to predict component failures in wind turbines. Most pipelines here are supervised (see Figure 6-2b as an example) due to the availability of labels.

9.1.5 What is the impact of *Orion* on future research?

Orion provides a base to explore more avenues of research; for example, developing new methods for incorporating user feedback by leveraging supervised and unsupervised models together. It provides a benchmarking framework and a pipeline hub that will aid researchers in developing new ML models and comparing them to existing pipelines. *Orion* provides a way to collect and incorporate human feedback continuously, leading to new innovations made possible by human-in-the loop systems. For example, collecting human annotations can result in the design of new preprocessing primitives.

9.2 Are We There Yet?

9.2.1 Can we use the framework in real applications?

Anomaly detection has existed for a long time, yet the industry still struggles to adopt anomaly detection algorithms. This is mainly because real data is a lot more complex than the pristine, curated academic datasets that have been studied for decades. When members of industry tried to apply these anomaly detection methods, they often fell short of the promised performance. With *Orion*, putting real datasets to the test is easily done using the benchmark utility. Our continuous evaluation of the system makes it more reliable. In addition, the framework we developed works with data in a close-to-raw form, helping users to try out the system with a simple plug-and-play.

9.2.2 Are benchmarks fair?

One of the hardest tasks in evaluating machine learning algorithm is fairness. With the massive space of hyperparameter settings, it is difficult to discover the right configuration of all pipelines. Even so, some researchers "slip" information about ground truth anomalies in order to tweak the pipelines and select hyperparameters that make the model perform better. While this process yields an improvement to

performance, it is critical to distinguish that the pipeline is no longer *unsupervised* — it is now supervised. Generally, pipelines struggle to stand the test of time and to perform as well on new datasets — one of the main reasons why we benchmark pipelines on a regular basis.

9.2.3 Do we even need machine learning?

Machine learning has garnered wide general interest, and many people are now developing new ML methods for a variety of tasks. However, ML should not be adopted because it is what is currently "hot" in the field, but rather because it is effective in a particular situation. We ran into an interesting example of a case where machine learning was not necessary. Referring back to Figure 3-1(b), we acquired a periodic sinusoidal time series data where the objective was to find noisy regions that were buried within the signal. Through experimentation, we found that the most effective way to locate these regions was through signal processing. Namely, we find the fundamental frequency of the signal and remove it, making the abnormal regions standout more prominently. *Orion* alone is not enough to solve the problem. To address these challenges, we built another framework for processing signals (SigPro) which is part of a larger ecosystem for time series analysis (Sintel) [4].

9.2.4 Are anomalies problems?

It is tempting to consider all anomalies to be problems, but this is not true in practice. As indicated by the domain experts, maneuvers are events that cause unusual behaviors in the signal, but are actually considered normal. Similarly, statistical outliers can occur as well, but in an expected scenario. Through this real-world collaboration with the satellite team, we have concluded that not just any anomaly counts as a problem — rather, problems are dependent on domain experts and how they define regular operations. Recently, several studies focus on quantifying uncertainty in unsupervised anomaly detection [70], increasing the interpretability of model predictions and creating safe trustworthy machine learning systems. With Orion, we need to find

a way to leverage human interventions in order to discriminate between legitimate anomalies versus exception cases.

Chapter 10

Conclusion

We have introduced *Orion*, a new end-to-end interactive anomaly detection framework for domain-specific time series. We first described the underaddressed problems that our system tries to solve, and provided details about the currently available set of anomaly detection methods for time series data. We then discussed our framework's different components — several state-of-the-art machine learning pipelines as well as a feedback integration mechanism — and how they interact with each other. We demonstrated how effective our framework can be for practical tasks, and presented a use case with a real-world application. Overall, we have shown how *Orion* can bridge the gap between domain experts and machine learning engineers, thus contributing heavily to the field of interactive machine learning.

Orion is currently in use by a large community that provides feedback, improvements, and contributions. This wide adoption has made *Orion* more robust and reliable to use. Continuous feedback has helped us create new features that users deem beneficial, and has increased their satisfaction. It is open-source and available at `https://github.com/sintel-dev/Orion`.

10.1 Future Perspective

Working on this thesis, it became clear that *Orion* alone will not cover the nuances in time series anomaly detection. Prior to using machine learning, we discovered that transforming signals between domains can be useful to pinpoint anomalous regions, which instantiated our endeavor for SigPro [1]. SigPro is a `python` library covering multiple signal processing techniques to convert raw time series into feature time series that encode the knowledge of domain experts for solving tasks using machine learning. Furthermore, to shift from the unsupervised setting of anomaly detection to supervised setting (after collecting annotations), we rely on Draco [2], a `python` library for supervised time series prediction tasks.

Overtime, we have closely observed the requirements of domain experts to understand the general workflow of their operations and to design the right system to solve their needs. To this end, we have created – *Sintel* [3] – an ecosystem for the general purpose of extracting insights from time series data. It encompasses data preparation and processing systems, three machine learning tasks, including: time series anomaly detection; time series classification; and time series forecasting, a generalized benchmarking framwork, RESTful APIs, and visualization & annotation tools. We are trekking our path towards a broader vision of Sintel, where we hope to solve the practical challenges faced by domain experts.

[1] https://github.com/sintel-dev/SigPro
[2] https://github.com/sintel-dev/Draco
[3] https://github.com/sintel-dev/

Appendix A

Database

The database stores all necessary information for the system. This includes the found anomalies and any outputs from the anomaly detection pipelines. Furthermore, we store any annotations that the user of the system might provide through the visual interface.

Figure A-1 demonstrates the use of *Orion*'s explorer, which connects directly to the database and records the results of the experiment. Figure A-2 shows the database collections and how they are related. The database is implemented in `mongoDB` and allows flexibility and scalability. We store a total of seven collections.

Signal Signal is the collection where different time series signals are stored. Note that we do not store any raw data in the database, but rather a link to the data location. We also store a reference to the Dataset collection, as well as metadata about the signal such as name, start and end time and in which columns the timestamps and values are located.

Dataset Within the Dataset collection we only have a name field for the dataset. However, as we store the reference to the Dataset collection within Signal, a Dataset can be understood as a set of signals. This way we can group signals to datasets and run pipelines on these sets.

PipelineTemplate The PipelineTemplate collection contains all the templates from which pipelines that will later be used to run experiments are generated. The template includes all the default hyperparameter values, as well as the tunable hyperparameter ranges.

Pipeline The Pipeline collection stores the pipeline JSON file, as required by ML-Primitives. Each pipeline is related to one PipelineTemplate and defines the specific hyperparameters that should be used.

Experiment As defined previously, an experiment is a the application of a pipeline on a Dataset. Therefore an experiment is linked to the Dataset collection and the PipelineTemplate collection. Furthermore, we add a project name to the experiment such that we are able to group experiments based on project names. Optionally a signalset can be specified, which can be a subset of the associated Dataset.

Datarun The Datarun objects represent single executions of an Experiment and contain all the information about the environment and context where this execution took place, which potentially allows later reproduction of the results in a new environment. It also contains information about whether the execution was successful or not, when it started and ended, and the number of events found in the experiment.

Signalrun Each Signalrun object represents a single execution of a Pipeline on a Signal within a Datarun. It contains information about whether the execution was successful or not, when it started and ended, the number of events that were found by the Pipeline, and where the model and metrics are stored.

Event All details regarding found anomalies are stored in the Event collection. Events have a reference to the Signalrun and contain a start-time, end-time and score field.

Event Interaction The Event Interaction collection records all the interaction history related to events.

Annotation Each Event can have multiple Annotations from one or more users. Annotations are inserted by domain experts after the Datarun has finished and they analyze the results.

```python
from orion.db import OrionDBExplorer
from orion.runner import start_datarun

# connect to db
orex = OrionDBExplorer(user='username', database='db')

dataset = orex.add_dataset(
    name='dataset',
)

# add signals to the dataset
orex.add_signals(
    dataset=dataset,
    signals_path='orion-data'
)

template = orex.add_template(
    name='lstmdt',
    template='lstm_dynamic_threshold',
)

# create an experiment
experiment = orex.add_experiment(
    name='experiment',
    template=template,
    dataset=dataset,
)

# configure template's hyperparameters
new_hyperparameters = {
   'keras.Sequential.LSTMTimeSeriesRegressor#1': {
       'epochs': 5,
       'verbose': True
   }
}

pipeline = orex.add_pipeline(
   name='lstmdt_5_epoch',
   template=template,
   hyperparameters=new_hyperparameters,
)

# run pipeline
start_datarun(orex, experiment, pipeline)
```

Figure A-1: Using python SDK. `OrionDBExplorer` allows users to run experiments and store them in the database as well as query and retrieve information from the database.

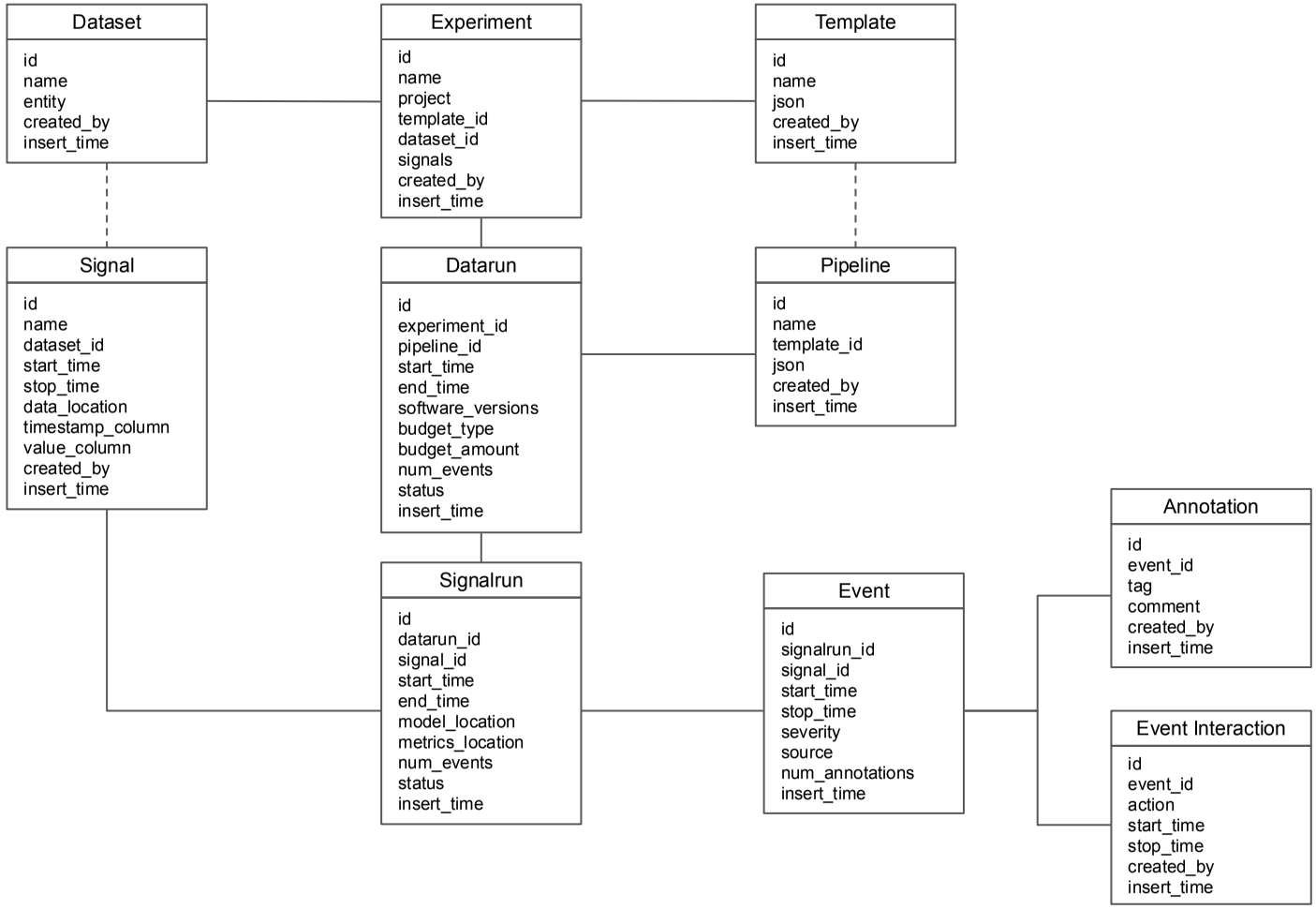


Figure A-2: Database schema

Appendix B

Reproducibility

This section contains more information about some implementation details of *Orion* and the extended results produced in this thesis.

B.1 Primitives

Type	Count
Pre-processing	8
Modeling	6
Post-processing	4

Table B.1: Primitives in the curated catalog of the source library categorized by type.

As of the writing of this thesis, there are a total of 8 pre-processing primitives, 6 modeling primitives, and 4 post-processing primitives. Using these primitives we assembled 6 different pipelines: ARIMA, LSTM DT, TadGAN, LSTM AE, Dense AE, and MS Azure. In the next section we expand on these pipelines.

B.2 Pipelines

As mentioned in section 6.1.2 the current implementation of *Orion* contains six distinct pipelines. Some are machine learning pipelines, others are statistical based

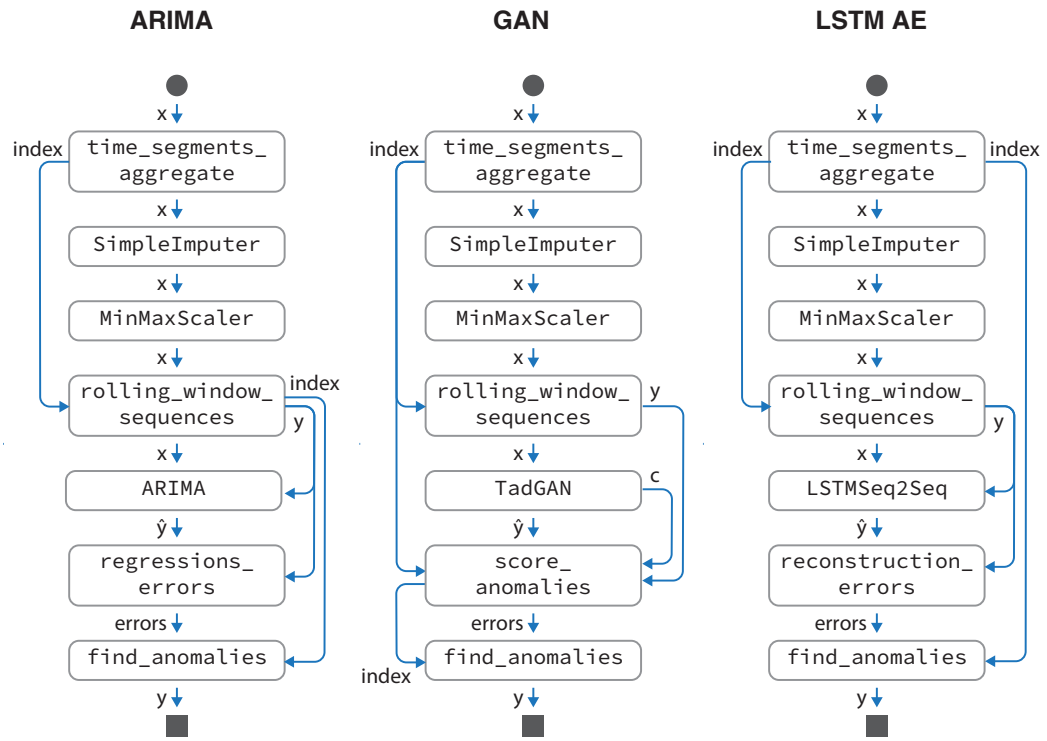


Figure B-1: Overview of the different anomaly detection pipelines

pipelines, and Azure [56] is a blackbox API integration. All pipelines consist of multiple steps. In addition to Figure 6-2, Figure B-1 showcases the remaining steps in the pipeline.

B.2.1 ARIMA

Prediction-based. An autoregressive integrated moving average (ARIMA) model is a popular statistical analysis model that learns autocorrelations in the time series for future value prediction. We use point-wise prediction errors as the anomaly scores to detect anomalies.

B.2.2 LSTM Dynamic Threshold (LSTM DT)

Prediction-based. A double stacked neural network consisting of two LSTM layers with 80 units each, and a subsequent dense layer with one unit that predicts the value at the next time step. This implementation is similar to the one proposed by Hundman et al. [37]. Point-wise prediction errors are used for anomaly detection.

B.2.3 Time Series Anomaly Detection using GAN (TadGAN)

Reconstruction-based. A Generative Adversarial Networks (GAN) containing multiple neural networks, including: an encoder, generator, critic x, and critic z with a cycle consistency loss. This model assumes that anomalous sequences would not be constructed as well as "normal" instances, making it easy to detect them. More details about the TadGAN implementation are available in our paper [28].

B.2.4 LSTM based Autoencoder (LSTM AE)

Reconstruction-based. Standard sequence-to-sequence model with LSTM layers. The LSTM autoencoder contains two LSTM layers, each with 60 units. A point-wise reconstruction error is used to detect anomalies.

B.2.5 Dense based Autoencoder (Dense AE)

Reconstruction-based. Similar to the LSTM AE model, where we have a sequence-to-sequence model with Dense layers. The dense autoencoder consists of three dense layers with 60, 20 and 60 units respectively. We also use a point-wise reconstruction error to detect anomalies.

B.2.6 MS Azure

AD Service. Microsoft Azure provides an anomaly detection service that uses Spectral Residual Convolutional Neural Networks (SR-CNN) in which the models are applied serially [56]. The SR model is responsible for saliency detection, and the CNN is responsible for learning a discriminating threshold. The output of the model is a sequence of binary labels (0 corresponding to "normal" and 1 to "anomalous") attributed to each timestamp.

		NASA		Yahoo S5				NAB				
Variation		MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
LSTM DT	P	0.375	0.680	0.690	0.966	0.991	0.893	0.333	0.391	0.625	0.542	0.538
	R	0.667	0.761	0.815	0.985	0.589	0.510	0.500	0.600	0.909	0.929	0.636
	F1	0.480	0.718	0.747	0.975	0.739	0.649	0.400	0.474	0.741	0.684	0.583
TadGAN	P	0.451	0.573	0.592	0.792	0.764	0.575	0.625	0.625	0.625	0.391	0.559
	R	0.639	0.761	0.522	0.855	0.358	0.280	0.833	0.667	0.909	0.643	0.576
	F1	0.529	0.654	0.555	0.822	0.487	0.377	0.714	0.645	0.741	0.486	0.567
LSTM AE	P	0.450	0.635	0.629	0.833	0.931	0.711	0.667	0.840	0.533	0.412	0.640
	R	0.500	0.701	0.562	0.900	0.286	0.171	0.667	0.700	0.727	0.500	0.485
	F1	0.474	0.667	0.593	0.865	0.438	0.276	0.667	0.764	0.615	0.452	0.552
ARIMA	P	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
	R	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
	F1	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
Dense AE	P	0.567	0.712	0.719	0.955	0.975	0.586	0.600	0.846	0.667	0.529	0.600
	R	0.472	0.627	0.590	0.845	0.042	0.049	0.500	0.733	0.545	0.643	0.455
	F1	0.515	0.667	0.648	0.897	0.080	0.091	0.545	0.786	0.600	0.581	0.517
MS Azure	P	0.032	0.011	0.166	0.484	0.542	0.216	0.027	0.036	0.169	0.035	0.161
	R	0.806	0.940	0.815	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818
	F1	0.061	0.021	0.276	0.653	0.702	0.344	0.053	0.068	0.286	0.068	0.269

Table B.2: Precision, Recall and F1-Scores of pipelines

	NASA		Yahoo S5				NAB						
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	**mean**	**std**
ARIMA	0.489	0.424	0.753	0.856	0.783	0.693	0.429	0.576	0.538	0.545	0.513	0.600	0.148
LSTM DT	0.495	0.750	0.757	0.987	0.756	0.643	0.400	0.531	0.452	0.718	0.620	0.646	0.172

Table B.3: F1 Scores Version 0.1.3

	NASA		Yahoo S5				NAB						
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	**mean**	**std**
ARIMA	0.344	0.307	0.744	0.816	0.782	0.684	0.429	0.472	0.538	0.429	0.513	0.551	0.179
LSTM DT	0.468	0.708	0.738	0.978	0.728	0.634	0.400	0.468	0.438	0.667	0.564	0.617	0.172
MS Azure	0.061	0.021	0.277	0.653	0.692	0.333	0.053	0.068	0.019	0.068	0.269	0.228	0.246

Table B.4: F1 Scores Version 0.1.4

	NASA		Yahoo S5				NAB						
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	**mean**	**std**
ARIMA	0.344	0.307	0.744	0.816	0.782	0.684	0.429	0.472	0.538	0.429	0.513	0.551	0.179
LSTM DT	0.532	0.704	0.735	0.980	0.743	0.653	0.400	0.462	0.467	0.615	0.548	0.622	0.166
TadGAN	0.575	0.644	0.626	0.700	0.494	0.381	0.714	0.677	0.800	0.450	0.592	0.605	0.124
MS Azure	0.061	0.021	0.271	0.653	0.697	0.337	0.053	0.068	0.019	0.068	0.269	0.229	0.247

Table B.5: F1 Scores Version 0.1.5

	NASA		Yahoo S5				NAB						
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	**mean**	**std**
ARIMA	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.727	0.472	0.429	0.513	0.568	0.186
LSTM DT	0.480	0.718	0.747	0.975	0.739	0.649	0.400	0.741	0.474	0.684	0.583	0.654	0.162
TadGAN	0.529	0.654	0.555	0.822	0.487	0.377	0.714	0.741	0.645	0.486	0.567	0.598	0.131
LSTM AE	0.474	0.667	0.593	0.865	0.438	0.276	0.667	0.615	0.764	0.452	0.552	0.578	0.165
Dense AE	0.515	0.667	0.648	0.897	0.080	0.091	0.545	0.600	0.786	0.581	0.517	0.539	0.252
MS Azure	0.061	0.021	0.276	0.653	0.702	0.344	0.053	0.019	0.070	0.068	0.269	0.231	0.249

Table B.6: F1 Scores Version 0.1.6

	NASA		Yahoo S5				NAB						
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	**mean**	**std**
ARIMA	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513	0.568	0.177
LSTM DT	0.466	0.689	0.739	0.975	0.746	0.645	0.400	0.500	0.759	0.585	0.551	0.641	0.156
TadGAN	0.517	0.633	0.551	0.841	0.484	0.376	0.571	0.689	0.769	0.562	0.559	0.596	0.125
LSTM AE	0.507	0.667	0.601	0.880	0.445	0.231	0.667	0.712	0.667	0.516	0.508	0.582	0.161
Dense AE	0.507	0.693	0.665	0.904	0.078	0.090	0.545	0.786	0.600	0.581	0.533	0.544	0.244
MS Azure	0.061	0.021	0.276	0.652	0.702	0.344	0.053	0.068	0.286	0.069	0.269	0.255	0.228

Table B.7: F1 Scores Version 0.1.7

	NASA		Yahoo S5				NAB						
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	**mean**	**std**
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	0.166
LSTM DT	0.447	0.671	0.724	0.975	0.751	0.637	0.400	0.488	0.733	0.619	0.535	0.635	0.157
TadGAN	0.465	0.646	0.549	0.843	0.492	0.389	0.667	0.623	0.741	0.476	0.523	0.583	0.128
LSTM AE	0.500	0.667	0.593	0.859	0.422	0.223	0.667	0.724	0.720	0.552	0.540	0.588	0.163
Dense AE	0.562	0.710	0.656	0.889	0.084	0.094	0.545	0.800	0.632	0.500	0.517	0.544	0.243
MS Azure	0.061	0.021	0.276	0.652	0.702	0.344	0.053	0.068	0.286	0.069	0.269	0.255	0.228

Table B.8: F1 Scores Version 0.2.0

	NASA		Yahoo S5				NAB						
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	**mean**	**std**
ARIMA	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513	0.568	0.177
LSTM DT	0.460	0.703	0.752	0.980	0.733	0.643	0.400	0.481	0.643	0.684	0.568	0.641	0.155
TadGAN	0.558	0.650	0.559	0.890	0.412	0.374	0.500	0.677	0.692	0.500	0.567	0.580	0.138
LSTM AE	0.480	0.690	0.600	0.870	0.436	0.243	0.545	0.750	0.615	0.571	0.525	0.575	0.158
Dense AE	0.529	0.661	0.652	0.899	0.087	0.092	0.545	0.741	0.632	0.552	0.517	0.537	0.236
MS Azure	0.050	0.020	0.279	0.652	0.702	0.344	0.053	0.068	0.250	0.068	0.269	0.251	0.229

Table B.9: F1 Scores Version 0.2.1

	NASA		Yahoo S5				NAB						
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	**mean**	**std**
ARIMA	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513	0.568	0.177
LSTM DT	0.476	0.741	0.739	0.990	0.753	0.644	0.400	0.537	0.714	0.703	0.556	0.659	0.154
TadGAN	0.575	0.659	0.564	0.853	0.439	0.369	0.615	0.656	0.640	0.540	0.559	0.588	0.119
LSTM AE	0.500	0.658	0.584	0.877	0.444	0.262	0.667	0.724	0.667	0.471	0.475	0.575	0.159
Dense AE	0.523	0.661	0.665	0.891	0.072	0.109	0.400	0.764	0.571	0.552	0.517	0.520	0.239
MS Azure	0.050	0.020	0.279	0.652	0.702	0.344	0.053	0.068	0.250	0.068	0.269	0.251	0.229

Table B.10: F1 Scores Version 0.3.0

Bibliography

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

[2] 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.

[3] 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.

[4] Sarah Alnegheimish, Dongyu Liu, Carles Sala, Laure Berti-Equille, and Kalyan Veeramachaneni. Sintel: A machine learning framework to extract insights from signals. *arXiv preprint arXiv:2204.09108*, 2022.

[5] Samaneh Aminikhanghahi and Diane J Cook. A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367, 2017.

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

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

[8] 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.

[9] Donald J Bemdt and James Clifford. Using Dynamic Time Warping to Find-Patterns in Time Series. In *AAAI-94 Workshop on Knowledge Discovery in Databases*, Seattle, Washington, 1994.

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

[11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[12] 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.

[13] 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.

[14] 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.

[15] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. Human-in-the-loop outlier detection. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 19–33, 2020.

[16] 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.

[17] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[18] 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.

[19] 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.

[20] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition. *Journal of Official Statistics*, 6(1):3–73, 1990.

[21] 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.

[22] 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.

[23] Shubhomoy Das, Weng-Keen Wong, Thomas Dietterich, Alan Fern, and Andrew Emmott. Incorporating expert feedback into active anomaly discovery. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 853–858. IEEE, 2016.

[24] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American statistical association*, 106(496):1513–1527, 2011.

[25] Dennis DeCoste. Automated learning and monitoring of limit functions. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 1997.

[26] Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060, 2017.

[27] 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.

[28] 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.

[29] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE*, 11(4):1–31, 2016.

[30] 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.

[31] 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.

[32] 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.

[33] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[35] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.

[36] 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.

[37] 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*, 2018.

[38] 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.

[39] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, 1975.

[40] David Iverson. Inductive system health monitoring. In *International Conference on Artificial Intelligence*, 2004.

[41] 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.

[42] Vincent Jacob, Fei Song, Arnaud Stiegler, Bijan Rad, Yanlei Diao, and Nesime Tatbul. Exathlon: A benchmark for explainable anomaly detection over time series. *arXiv preprint arXiv:2010.05073*, 2021.

[43] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[44] Eamonn J Keogh and Michael J Pazzani. Scaling up Dynamic Time Warping for Datamining Applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289. ACM, 2000.

[45] 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.

[46] 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.

[47] 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.

[48] Sean M. Law. STUMPY: A Powerful and Scalable Python Library for Time Series Data Mining. *Journal of Open Source Software*, 4(39):1504, 2019.

[49] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proceedings of the 2015 internet measurement conference*, pages 211–224, 2015.

[50] Dongyu Liu, Sarah Alnegheimish, Alexandra Zytek, and Kalyan Veeramachaneni. Mtv: Visual analytics for detecting, investigating, and annotating anomalies in multivariate time series. *arXiv preprint arXiv:2112.05734*, 2021.

[51] 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.

[52] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[53] José-Antonio Martínez-Heras and Alessandro Donati. Enhanced Telemetry Monitoring with Novelty Detection. *AI Magazine*, 35(4):37, 2014. ISSN 0738-4602.

[54] Dan Pelleg and Andrew Moore. Active learning for anomaly and rare-category detection. *Advances in neural information processing systems*, 17:1073–1080, 2004.

[55] 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.

[56] 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 & Data Mining*, pages 3009–3017, 2019.

[57] 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.

[58] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

[59] 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.

[60] 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.

[61] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[62] Nesime Tatbul, Tae Jun Lee, Stan Zdonik, Mejbah Alam, and Justin Gottschlich. Precision and recall for time series. *Advances in neural information processing systems*, 31, 2018.

[63] 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.

[64] Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167, 2020.

[65] Gerrit JJ van den Burg and Christopher KI Williams. An evaluation of change point detection algorithms. *arXiv preprint arXiv:2003.06222*, 2020.

[66] 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.

[67] Heng Wang and Zubin Abraham. Concept drift detection for streaming data. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2015.

[68] Geoffrey I Webb, Loong Kuan Lee, François Petitjean, and Bart Goethals. Understanding concept drift. *arXiv preprint arXiv:1704.00362*, 2017.

[69] 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.

[70] Bang Xiang Yong and Alexandra Brintrup. Bayesian autoencoders with uncertainty quantification: Towards trustworthy anomaly detection. *arXiv preprint arXiv:2202.12653*, 2022.

[71] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Scshaar. Time-series generative adversarial networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[72] 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.

[73] 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.

[74] 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.

[75] 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.