Sintel: A Machine Learning Framework to Extract Insights from Signals

Sarah Alnegheimish
smish@mit.edu
MIT

Dongyu Liu
dongyu@mit.edu
MIT

Carles Sala
csala@csail.mit.edu
MIT

Laure Berti-Equille
laure.berti@ird.fr
IRD

Kalyan Veeramachaneni
kalyanv@mit.edu
MIT

ABSTRACT

The detection of anomalies in time series data is a critical task with many monitoring applications. 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 ML experts. In this paper, we introduce *Sintel*, a machine learning framework for end-to-end time series tasks such as anomaly detection. The framework uses state-of-the-art approaches to support all steps of the anomaly detection process. *Sintel* logs the entire anomaly detection journey, providing detailed documentation of anomalies over time. It enables users to analyze signals, compare methods, and investigate anomalies through an interactive visualization tool, where they can annotate, modify, create, and remove events. Using these annotations, the framework leverages human knowledge to improve the anomaly detection pipeline. We demonstrate the usability, efficiency, and effectiveness of *Sintel* through a series of experiments on three public time series datasets, as well as one real-world use case involving spacecraft experts tasked with anomaly analysis tasks. *Sintel*'s framework, code, and datasets are open-sourced at https://github.com/sintel-dev/.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Information systems** → *Information systems applications.*

KEYWORDS

Machine Learning Framework, Anomaly Detection, Human-In-the-Loop AI, Time Series Data, Data Science Pipeline

ACM Reference Format:
Sarah Alnegheimish, Dongyu Liu, Carles Sala, Laure Berti-Equille, and Kalyan Veeramachaneni. 2022. Sintel: A Machine Learning Framework to Extract Insights from Signals. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3514221.3517910

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06.
https://doi.org/10.1145/3514221.3517910

1 INTRODUCTION

Many industries have set up processes and workflows for analyzing time series to monitor, control, and optimize the behaviours of the products and services they offer. These range from monitoring applications that we use on a daily basis (Zoom, Slack, and several others) to managing a fleet of heavy equipment like turbines and satellites. 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. In this paper, we pick one amongst those workflows – time series anomaly detection – and present the challenges and solutions.

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 [23]. As data has become larger, more complex, and increasingly multidimensional, traditional distance- [13], density- [50] or isolation-based [33] approaches have begun to perform less competitively in real-world scenarios.

Since then, ML and deep learning based methods have garnered increased attention [27, 40, 48, 49]. With the wide diversity of available methods, choosing pipelines and evaluating which one is most suitable can be difficult. Depending on the application, it may be insufficient to rely on data benchmarks [30] 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 of interest. Services and libraries, such as Microsoft Azure [38], and Luminaire [11], for AD have been recently proposed. However, their users may still run into the following problems.

HIL analysis. Services often neglect the Human-In-the-Loop (HIL) 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 could result in loss of information.

Contextual knowledge. 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.

Customizing workflows. 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 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 *Sintel*. 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.

The key contributions of this paper are summarized as follows:

- **An efficient end-to-end time series anomaly detection framework.** *Sintel* provides a suite of anomaly detection pipelines executable through a user-friendly interface. 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 anomaly 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. 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 the 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. 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. As of writing the paper, the benchmarking suite 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. Moreover, we conduct an experiment to assess the framework's ability to learn from experts' annotations. We compare the features of *Sintel* against existing systems (presented in Table 1). Lastly, we deploy the *Sintel* in a real-world setting with a leading satellite operation company.
- **Open-source.** Our framework, code and datasets are available at: Orion, Draco, MTV, and sintel API. Scripts to reproduce the results in the paper are available in [3].

2 SINTEL

Throughout this section, we instantiate the core idea for time series anomaly detection. Given $X = \{x_1, x_2, x_3, \ldots, x_n\}$, a multivariate signal with m channels where $x_i \in \mathbb{R}^m$, and assuming there exists a set of *variable-length* anomalies $A = \{(t_s, t_e) \,|\, 1 \leq t_s < t_e \leq n\}$ that is unknown a priori, *Sintel* aims to detect A using a combination of machine and human intelligence. Note that *Sintel* 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. The system comprises a series of components that can perform all the necessary detection steps, from composing pipelines to annotating anomalies. We describe the details in this section.

2.1 Real-World Scenario

To explain the motivation for *Sintel* and illustrate the anomaly detection workflow (Figure 1), we describe a real-world scenario drawn from our three-year collaboration with a world-leading communication satellite company. One major objective of the company's operative 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 ML experience (0 to 3 years).

The team works with multiple spacecrafts. Each spacecraft telemetry database contains around 37K signals spanning 9 different subsystems. Each signal is a univariate time series collected at 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 that are identified as true alarms but non-urgent, the experts gather further information over some time window to help explain the root cause and the way forward.

We experienced similar challenges with other teams as well. Over the course of this process, teams face the following challenges: (**C1**) Setting and adjusting threshold methods can be user demanding and laborious, forcing teams to restrict their focus to a subset of a few hundred signals chosen based on domain knowledge; (**C2**) Teams want to use ML models to identify contextual anomalies — anomalies that do not exceed a normal range, but are unusual compared to local values. With limited ML experience, it can be difficult to adopt such methods; (**C3**) With the abundance of AD methods, teams struggle to know which ML model to select for a particular dataset; (**C4**) Teams found that ML models often flag unusual patterns, even when these patterns do not necessarily indicate a problem. For example, a 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 integrate this feedback; (**C5**) Teams frequently discuss anomalous events of interest. However, they are not currently collecting these events into a knowledge base to learn from.

Sintel addresses all of these challenges. *Sintel* is an automated end-to-end framework that is able to identify anomalous events

		MS Azure [38]	ADTK [6]	Luminaire [11]	TODS [28]	Telemanom [24]	NAB [1]	EGADS [29]	Stumpy [31]	GluonTS [2]	*Sintel*
Users	End User	✓	✓	✓	✗	✗	✗	✗	✓	✗	✓
	System Builder	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
	ML Researcher	✗	✗	✗	✓	✓	✓	✓	✗	✓	✓
Engine	Preprocessing	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓
	Modeling	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
	Postprocessing	✗	✓	✓	✓	✗	✗	✗	✓	✗	✓
Modular		✗	✓	✓	✓	✗	✗	✗	✓	✓	✓
Comp.	Evaluation	✗	✓	✗	✗	✓	✗	✗	✗	✗	✓
	Benchmark	✗	✗	✗	✓	✗	✓	✗	✗	✓	✓
	Database	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
API	`lang. specific`	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓
	`RESTful`	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
HIL		✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Table 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.

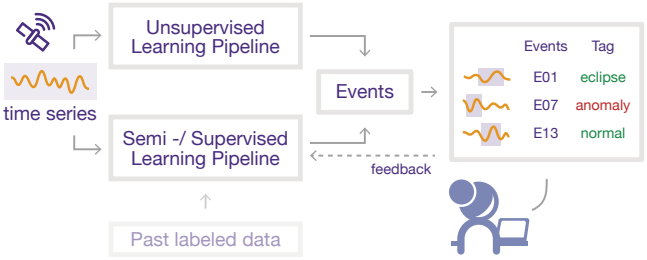


Figure 1: AD workflow. Use an unsupervised pipeline to locate anomalies, which are then presented to the expert for annotation. The annotated events are provided to the semi-/supervised pipeline — which can be pre-trained with past labeled data — to learn from feedback and keep improving.

from tens of thousands of signals (**C1**). It integrates various state-of-the-art anomaly detection pipelines, and provides simple user-friendly APIs to interact with the framework (**C2**). With the benchmark suite, *Sintel* 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 *Sintel*'s visual interface, the team investigates flagged anomalies, annotates them, and incorporates the feedback back into the framework (**C4**). The feedback is stored in an integrated database that maintains the results of each detected event and annotations (**C5**).

2.2 Anomaly Detection Pipelines

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 *Sintel* is agnostic as to which pipeline is executed. To understand the composition of pipelines, we describe their basic building blocks, or *primitives*, below.

Primitives are reusable software components [39]. 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 1, *Sintel* 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, such as imputing missing values.

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.

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 subsequences alongside their likelihood probability of being anomalous.

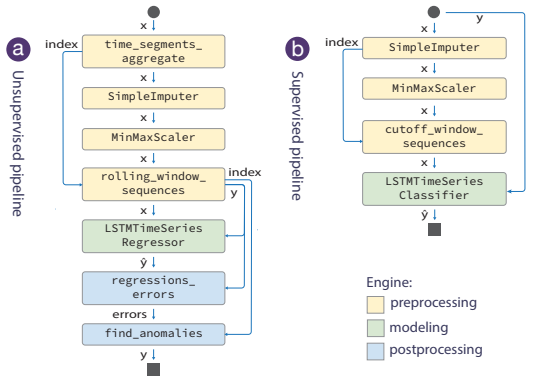


Figure 2: Graphic representations of (a) an unsupervised pipeline with LSTM network. (b) its supervised counterpart.

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 *Sintel* without modifying an entire pipeline.

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 2. In this paper, the term "pipeline" always refers to a program tasked to identify anomalies in time series data. Primitive and pipeline structures have been successfully adopted in many applications, including healthcare [4, 39].

Dissecting LSTM Pipeline. The pipeline in Figure 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 dynamic threshold on error values to find anomalous regions [24]. 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`.

2.3 Pipeline Evaluation Metrics

In order to measure the relative performances of different pipelines, we need a way to compare detected anomalies with ground truth labels according to a well-defined metric. In classification, the most widely-used metrics include *precision*, *recall* and *F1-score*. However, as noted by Tatbul et al. [41], 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. Each approach allows for a different assessment.

Input: ground truth anomalies T, predicted anomalies P
Output: confusion matrix
begin
  $E \leftarrow T \cup P$ // all t_s and t_e timestamps
  $\tilde{T} \leftarrow \varnothing, \tilde{P} \leftarrow \varnothing, W \leftarrow \varnothing$
  $E \leftarrow \text{sort}(E)$ // sort timestamps from small to large
  $s \leftarrow \text{pop}(E)$ // the first timestamp
  while $E \neq \varnothing$ **do**
    $e \leftarrow \text{pop}(E)$
    $ti \leftarrow (t_s, t_e)$ // create a time interval (t_s, t_e)
    $W \leftarrow W \cup \{t_e - t_s\}$
    $\tilde{T} \leftarrow \tilde{T} \cup \{\text{overlap}(ti, T)\}$ // check if ti in ground truth
    $\tilde{P} \leftarrow \tilde{P} \cup \{\text{overlap}(ti, P)\}$ // check if ti in predicted
    $s \leftarrow e$
  end
  return `confusion_matrix` $(\tilde{T}, \tilde{P}, W)$

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.

Input: ground truth anomalies T, predicted anomalies P
Output: confusion matrix $\langle \text{tp}, \text{fp}, \text{fn} \rangle$
begin
  $U \leftarrow \varnothing$ // bookkeeping unmatched events
  $\text{tp} \leftarrow 0$
  while $T \neq \varnothing$ **do**
    $t \leftarrow \text{pop}(T)$
    for $p \in P$ **do**
      if $\text{overlap}(t, p)$ **then**
        $\text{tp} \leftarrow \text{tp} + 1$ // matched
      end
    end
    if $\text{unmatched}(t)$ **then**
      $U \leftarrow U \cup \{t\}$ // add to unmatched
    end
  end
  $\text{tn} \leftarrow |U|$
  $\text{fp} \leftarrow |P| - \text{tp}$
  return $\langle \text{tp}, \text{fp}, \text{fn} \rangle$

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

2.3.1 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 each segment, we compute its confusion matrix and weight it by its time range. This approach is valuable when the user aims to detect the exact segment of the anomaly. In cases where the user's signal is inherently regularly sampled, this approach is equivalent to a sample-based evaluation.

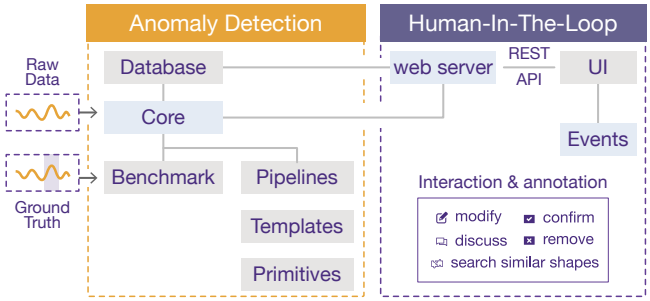


Figure 3: *Sintel* consists of two subsystems. The anomaly detection subsystem detects anomalies, which are then annotated by users using the human-in-the-loop subsystem.

2.3.2 Overlapping Segment. Overlapping segment is a more lenient evaluation approach. It is inspired by the evaluation method of Hundman et al. [24], 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. Algorithm 2 illustrates our approach to counting true positives, false positives, and false negatives.

2.4 Human Annotations

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 3.6). Users can interact with this system by *confirming, modifying, removing, searching,* and *discussing* events.

Why is the logic important? These annotations effectively incorporate domain-specific knowledge into our anomaly detection framework. This reduces both false positives and false negatives and helps improve future predictions.

3 SYSTEM DESIGN AND ARCHITECTURE

Sintel's components are categorized into two main subsystems: anomaly detection and human-in-the-loop, with a communication channel between them (Figure 3). The core (Section 3.1) is the main entry point, which allows users to train and benchmark pipelines and to predict and store anomalies. The machine learning stack (Section 3.2) contains primitives, templates, and pipelines, followed by the description of the hyperparameter tuning component (Section 3.3). Due to the nature of anomaly detection pipelines we include a benchmark utility (Section 3.4) to compare the quality and computational performance of different pipelines. We supplement the framework with a database (Section 3.5) 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 3.6).

3.1 Core

Sintel's core provides a set of coherent APIs, allowing users to execute end-to-end processes (**C1, C2**). Given a signal X, we want to obtain a set of detected anomalies. With *Sintel*, this is straightforward. First, the user loads a signal which follows the input standard

(a) End-to-end anomaly detection

```
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)
```

(b) Tuning pipelines

```
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')
```

(c) Benchmarking

```
from sintel.benchmark import benchmark

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

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

Figure 4: Example code for (a) end-to-end anomaly detection, where users load the data, select a pipeline (e.g. `lstm_dynamic_threshold`), train a pipeline, and detect anomalies; (b) pipeline tuning; and (c) benchmarking.

– (timestamp, values). Next, the user selects the pipeline of interest from a suite of available options. Once a pipeline is selected, it is trained on the signal using `sintel.fit(data)`. To detect anomalies, the user then executes `sintel.detect(data)` to produces a set of possible anomalies. Figure 4a shows 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 [9].

3.2 Machine Learning Stack

As described in Section 2.2, a pipeline is a set of primitives that executes an operation. Primitives can be reused between pipelines, which makes *Sintel* code-efficient. 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 [39]. 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.

AD Pipeline Hub. Sintel 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 `LSTM DT` [24], `LSTM AE` [34], and `TadGAN` [21]. Moreover, we provide "pipelines" that can connect to existing anomaly detection services, such as MS Azure [38]. This set of pipelines is easily extendable and can be expanded further (**C2**).

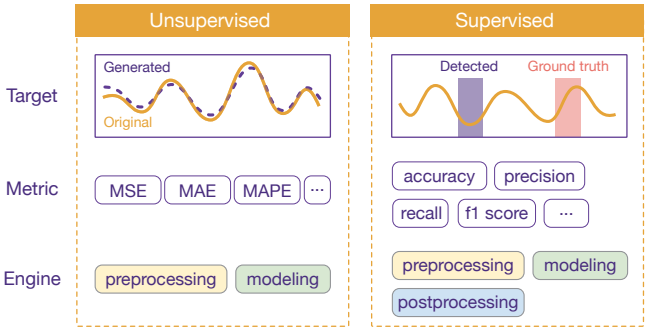


Figure 5: Hyperparameter tuning in two conditions: (1) unsupervised, where the goal is to optimize the signal generated by the ML model, and (2) supervised, where our goal is to produce anomalies that best match the ground truth set.

3.3 Tuning Hyperparameters

Hyperparameter tuning is instrumental to ML systems [7, 8]. To tune pipelines automatically (Figure 4b), we integrate `BTB`, an open-source and extensible framework with black box Bayesian Optimization [39]. 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.

Sintel's tuners are customized for use in two different settings, as shown in Figure 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 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 (**C4**). Depending on the tuning setting, some engines may not need to be tuned. Recall in Section 2.2, our primitives are annotated, enabling *Sintel* to automatically pull hyperparameters with respect to the set of primitives needed.

3.4 Benchmarking Framework

The availability of benchmarking is one of the key advantages of our framework (**C3**). As shown in Table 1, many existing frameworks develop algorithms for AD, but lack an out-of-the-box benchmark. With *Sintel*'s evaluation metrics and pipeline hub, we are able to thoroughly compare these methods on multiple datasets, through a single command — `sintel.benchmark`. Figure 4c illustrates our benchmark API. The benchmark is designed to measure two main aspects: quality and computational performance.

3.4.1 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 2.3. The benchmark is extensible, and users can easily add new metrics and evaluation criteria.

3.4.2 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. The inclusion of computational monitoring allows us to identify which primitives cause bottlenecks within the pipeline and to improve them further.

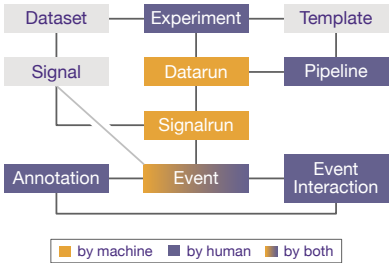


Figure 6: High-level database schema.

3.5 Persistent Knowledge Base

In real-world settings, it is often necessary to continuously record data, including expected anomalies (**C5**). 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 *Sintel*, 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 *Sintel*'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; and (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. 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.

3.6 Anomaly Annotation and Feedback

Another phase of our framework's workflow is anomaly analysis. Our goal is to enable the user to annotate flagged events and inspect detected anomalies. To achieve this, we introduce an interactive visualization tool [32]. This tool allows experts to undertake an investigation process, and to annotate and discuss events.

The visualization subsystem supports standard operations such as multi-signal viewing. In addition, it allows for a multi-aggregation view, which enables 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 can help to form the canonical logic behind a decision, the steps themselves are often quickly forgotten. Within our framework, this information is specifically collected and stored in a database, so that users can trace back the decision-making process.

The experts' knowledge is fed into a semi-/supervised pipeline to calibrate the output of automated detection (**C4**). After developing a careful understanding of what the users needed, we based the refreshment process of the semi-/supervised pipeline on application-specific batch processing of annotations. We justify our design based on the high variability in anomaly frequency among domains. Based on the satellite company's particular needs, we concluded that a weekly update is sufficient.

4 EVALUATION

In the following section, we introduce several experiments that demonstrate the use, performance, and effectiveness of *Sintel*. We experimentally evaluate its ability to complete anomaly detection in unsupervised and semi-supervised settings. In addition, we report the results of our study with the satellite company's operation team.

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 2 summarizes each of the datasets used.

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 [24], ARIMA [37], TadGAN [21], LSTM AE [34], and Dense AE – and a pipeline that uses the MS Azure anomaly detection service [38]. We evaluate the performances of the pipelines according to the evaluation metric described in Section 2.3. To run the benchmark, we use a private HPC cluster with 192GB of memory and two 24-core Intel Xeon CPUs.

Quality Performance. Table 3 shows the quality performances of currently available pipelines. The scores are calculated according to the *overlapping segment* approach described in Section 2.3. 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 [38] 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.

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

Table 2: Dataset Summary: 492 signals and 2349 anomalies.

Computational Performance. Figure 7a 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.

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 7b 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.

AutoML performance. *Sintel* 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 7c 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 2a). This demonstrates the level of effectiveness of automated hyperparameter optimization that the user may expect to obtain.

[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

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

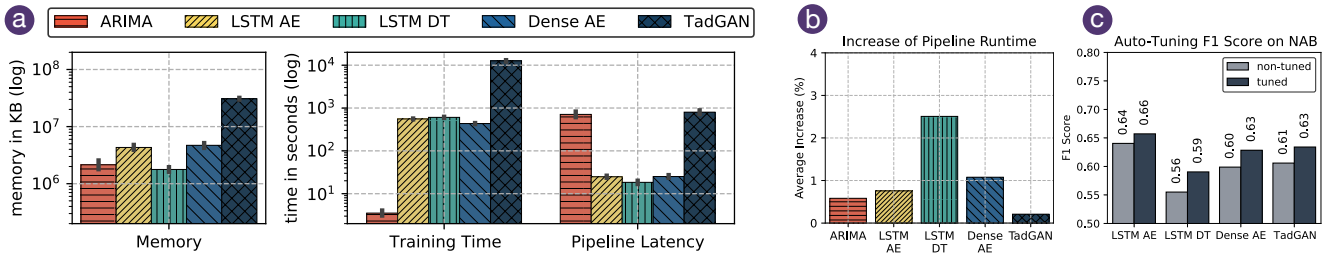
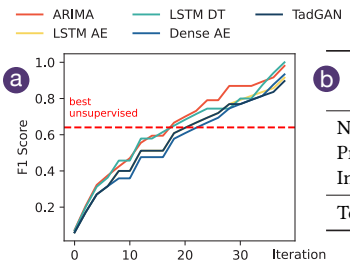


Figure 7: (a) Pipeline computational performance. (b) Difference in runtime between stand-alone primitives and end-to-end pipelines. (c) F1 Scores prior to and after tuning pipelines on the NAB dataset using a ground truth set of anomalies.

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 8a, 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 8a, 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 retrain the pipeline by estimating the benefit gain ahead of time, so as not to incur unnecessary costs.



b

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

Figure 8: (a) Semi-supervised pipeline performance on NAB through simulating annotations from different starting points. (b) Collected tags from real-world use case.

Real-world use case. We demonstrate the usability of our framework through the real-world use case introduced in Section 2.1. 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 track metrics like electrical power, thermal temperature, etc. We recorded the usage and activity of 6 experts and conducted interviews to gain their qualitative feedback.

We summarize the annotations collected from the study in Figure 8b. 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 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.

5 DISCUSSION

In this section, we highlight some of the salient aspects of *Sintel*, its limitations, and our attempts to create a framework that could be improved and evolve over time.

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 Figure 8b, we note that the ML missed $^{27}/_{110}$ events. When investigating why, we discovered that some events, such as a lunar eclipse, 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.

Addressing distribution shifts. Any pipeline's performance relies on the *preprocessing engine*. For example, domain experts pointed out that 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. Upon 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 feature shift-elimination techniques such as decomposition [15, 18] as well as by segmenting signals using change point detection techniques [5, 42, 43]. Since *Sintel*'s pipelines are modular and composed of primitives, it is possible to add new preprocessing primitives and to integrate them into *Sintel*.

Mixing supervised and unsupervised. In Figure 8a, 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 [44]. As we saw in Figure 8a, the model did not always benefit from new annotations, necessitating the need for such curation. We also note that pipelines will need to be updated when we observe drifts in the streaming data [45, 46].

Going beyond satellite operations. Although *Sintel* 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 *Sintel* to use, predicting component failures in wind turbines using application-specific pipelines. Most pipelines here are supervised (see Figure 2b as an example) due to the availability of labels. You can find our wind energy repository available at https://github.com/sintel-dev/Draco.

How will *Sintel* enable future research? *Sintel* provides a base to explore more avenues of research; for example, developing new methods of 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. *Sintel* 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.

6 RELATED WORK

Time series AD algorithms. Many algorithms have been proposed to address time series anomaly detection [12, 22]. The most basic approaches simply flag regions where values exceed a certain threshold [19, 35]. More advanced methods are based on statistical hypothesis testing [51], clustering [25, 26], and/or machine learning [47]. Recent advancements in deep neural networks have led to the emergence of deep learning-based anomaly detection approaches [21, 24, 34]. However, in order to be effectively used by domain experts, these algorithms cannot stand alone and must become part of an end-to-end workflow with relevant APIs.

Time series AD systems. A wide range of systems made specifically for time series data have been proposed. These address a variety of tasks including classification [10], feature extraction [14], and anomaly detection [20, 29, 38, 44]. Table 1 summarizes the features present in some of the 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.

Benchmarks. Benchmarking frameworks are necessary to evaluate and compare different anomaly detection models [16]. Lavin and Ahmad [30] introduced the first open-source benchmark library for time series anomaly detection on 58 signals from various sources. However, it is difficult to integrate new algorithms into the library. Jacob et al. [27] introduced Exathlon — a benchmark framework for anomaly detection and explanation discovery. This work addresses intricacies regarding the establishment of time series benchmarks, including evaluation metrics and performance monitoring. Currently, benchmarking frameworks have limited pipelines, and are not easily extendable.

Active incorporation of user feedback. After automatically detecting anomalies, we need to continuously recalibrate the detection model to adapt to anomalies identified by human experts. Pelleg and Moore [36] propose an active learning approach that generates the detected anomalies and prompts the expert to classify them. AI^2 [44] uses a similar strategy with an additional layer of combining supervised and unsupervised output. Das et al. [17] suggest an iterative process that involves training a supervised model, surfacing the most outlying points for expert review, and updating the model and feature weights accordingly. Although all of these methods are promising, they have been developed specifically for tabular data and require prior knowledge of data distribution.

7 CONCLUSION

In this paper, we have introduced 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. We then discussed its 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 Sintel can bridge the gap between domain experts and machine learning engineers, thus contributing heavily to the field of interactive machine learning.

Sintel is a larger ecosystem that can perform many tasks, including time series classification, regression, forecasting, and anomaly detection. All libraries are currently publicly available, with a large community providing feedback, improvements, and contributions.

ACKNOWLEDGMENTS

We thank SES S.A. for their invaluable insights, feedback, and their financial support to the project. We thank Arash Akhgari for his assistance in editing the graphics. Furthermore, the authors would like to acknowledge the contributions of Ihssan Tinawi, Manuel Alvarez Campo, Hector Dominguez, Alex Geiger, and Plamen Valentinov Kolev. Author Sarah Alnegheimish is supported by a scholarship from King Abdulaziz City for Science and Technology.

REFERENCES

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

[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. 2020. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *Journal of Machine Learning Research* 21, 116 (2020), 1–6.

[3] Sarah Alnegheimish. 2022. *Reproducibility of Sintel: A Machine Learning Framework to Extract Insights from Signals.* https://github.com/sarahmish/sintel-paper

[4] Sarah Alnegheimish, Najat Alrashed, Faisal Aleissa, Shahad Althobaiti, Dongyu Liu, Mansour Alsaleh, and Kalyan Veeramachaneni. 2020. Cardea: An Open Automated Machine Learning Framework for Electronic Health Records. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 536–545.

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

[6] Arundo Analytics. 2020. *Anomaly Detection Toolkit.* https://github.com/arundo/adtk

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

[8] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-parameter Optimization. *Journal of Machine Learning Research* 13, 2 (2012), 281–305.

[9] 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. 2013. API design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238* (2013).

[10] 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. 2019. Smile: A System to Support Machine Learning on EEG Data at Scale. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2230–2241.

[11] Sayan Chakraborty, Smit Shah, Kiumars Soltani, Anna Swigart, Luyao Yang, and Kyle Buckingham. 2020. Building an Automated and Self-Aware Anomaly Detection System. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 1465–1475.

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

[13] Wanpracha Art Chaovalitwongse, Ya-Ju Fan, and Rajesh C. Sachdeo. 2007. 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 (2007), 1005–1016.

[14] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. 2018. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing* 307 (2018), 72–77.

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

[16] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2017. Dawnbench: An end-to-end deep learning benchmark and competition. *Training* 100, 101 (2017), 102.

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

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

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

[20] Jingkun Gao, Xiaomin Song, Qingsong Wen, Pichao Wang, Liang Sun, and Huan Xu. 2020. RobustTAD: Robust Time Series Anomaly Detection via Decomposition and Convolutional Neural Networks. *arXiv preprint arXiv:2002.09545* (2020).

[21] Alexander Geiger, Dongyu Liu, Sarah Alnegheimish, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2020. TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 33–43.

[22] Markus Goldstein and Seiichi Uchida. 2016. A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data. *PLOS ONE* 11, 4 (2016), 1–31.

[23] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. 2013. Outlier Detection for Temporal Data: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 26, 9 (2013), 2250–2267.

[24] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. 2018. Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[25] David Iverson. 2004. Inductive System Health Monitoring. In *International Conference on Artificial Intelligence*.

[26] David Iverson. 2008. Data Mining Applications for Space Mission Operations System Health Monitoring. In *SpaceOps 2008 Conference*. American Institute of Aeronautics and Astronautics, 3212.

[27] Vincent Jacob, Fei Song, Arnaud Stiegler, Bijan Rad, Yanlei Diao, and Nesime Tatbul. 2021. Exathlon: A Benchmark for Explainable Anomaly Detection over Time Series. *arXiv preprint arXiv:2010.05073* (2021).

[28] Kwei-Harng Lai, Daochen Zha, Guanchu Wang, Junjie Xu, Yue Zhao, Devesh Kumar, Yile Chen, Purav Zumkhawaka, Minyang Wan, Diego Martinez, and Xia Hu. 2020. TODS: An Automated Time Series Outlier Detection System. *arXiv preprint arXiv:2009.09822* (2020).

[29] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. 2015. Generic and Scalable Framework for Automated Time-series Anomaly Detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1939–1947.

[30] Alexander Lavin and Subutai Ahmad. 2015. Evaluating real-time anomaly detection algorithms–the Numenta anomaly benchmark. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 38–44.

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

[32] Dongyu Liu, Sarah Alnegheimish, Alexandra Zytek, and Kalyan Veeramachaneni. 2021. MTV: Visual Analytics for Detecting, Investigating, and Annotating Anomalies in Multivariate Time Series. *arXiv preprint arXiv:2112.05734* (2021).

[33] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICMD)*. 413–422.

[34] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. 2016. LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. *arXiv preprint arXiv:1607.00148* (2016).

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

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

[37] Eduardo HM Pena, Marcos VO de Assis, and Mario Lemes Proença. 2013. Anomaly Detection Using Forecasting Methods ARIMA and HWDS. In *2013 32nd International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 63–66.

[38] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-Series Anomaly Detection Service at Microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3009–3017.

[39] Micah J Smith, Carles Sala, James Max Kanter, and Kalyan Veeramachaneni. 2020. 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*. 785–800.

[40] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. 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*. 2828–2837.

[41] Nesime Tatbul, Tae Jun Lee, Stan Zdonik, Mejbah Alam, and Justin Gottschlich. 2018. *Advances in neural information processing systems* 31 (2018).

[42] Charles Truong, Laurent Oudre, and Nicolas Vayatis. 2020. Selective Review of Offline Change Point Detection Methods. *Signal Processing* 167 (2020).

[43] Gerrit JJ van den Burg and Christopher KI Williams. 2020. An Evaluation of Change Point Detection Algorithms. *arXiv preprint arXiv:2003.06222* (2020).

[44] Kalyan Veeramachaneni, Ignacio Arnaldo, Vamsi Korrapati, Constantinos Bassias, and Ke Li. 2016. AI^2: 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)*. IEEE, 49–54.

[45] Heng Wang and Zubin Abraham. 2015. Concept Drift Detection for Streaming Data. In *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–9.

[46] Geoffrey I Webb, Loong Kuan Lee, François Petitjean, and Bart Goethals. 2017. Understanding Concept Drift. *arXiv preprint arXiv:1704.00362* (2017).

[47] Takehisa Yairi, Yoshinobu Kawahara, Ryohei Fujimaki, Yuichi Sato, and Kazuo Machida. 2006. 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)*. IEEE, 466–476.

[48] Ye Yuan, Guangxu Xun, Fenglong Ma, Yaqing Wang, Nan Du, Kebin Jia, Lu Su, and Aidong Zhang. 2018. MuVAN: A Multi-View Attention Network for Multivariate Temporal Data. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 717–726.

[49] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V Chawla. 2019. 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)*, Vol. 33. 1409–1416.

[50] Rui Zhang, Shaoyan Zhang, Sethuraman Muthuraman, and Jianmin Jiang. 2007. 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*. 31–37.

[51] Dequan Zheng, Fenghuan Li, and Tiejun Zhao. 2016. Self-Adaptive Statistical Process Control for Anomaly Detection in Time Series. *Expert Systems with Applications* 57 (2016), 324–336.