CALM: A Framework for Continuous, Adaptive, and LLM-Mediated Anomaly Detection in Time-Series Streams

Ashok Devireddy*[1] and Shunping Huang[1]

[1]Dataflow ML Team , ashokrd@berkeley.edu, shunping@google.com

September 1, 2025

Abstract

The detection of anomalies in non-stationary time-series streams is a critical but challenging task across numerous industrial and scientific domains. Traditional models, trained offline, suffer significant performance degradation when faced with concept drift, where the underlying statistical properties of the data change over time. This paper introduces CALM (Continuous, Adaptive, and LLM-Mediated), a novel, end-to-end framework for real-time anomaly detection designed to address this challenge. CALM is built on the Apache Beam distributed processing framework and leverages the TimesFm foundation model for forecasting-based anomaly detection. The framework's novelty lies in two core contributions. First, it implements a closed-loop, continuous fine-tuning mechanism that allows the anomaly detection model to adapt to evolving data patterns in near real-time. Second, it introduces an "LLM-as-a-Judge" component—a Large Language Model that provides semantic, context-aware judgments on detected anomalies to curate a high-quality training dataset, deciding whether an anomaly represents transient noise or a meaningful pattern shift. We evaluate CALM on the comprehensive TSB-UAD benchmark. Our results demonstrate that the continuously fine-tuned model improves the ROC AUC score in most datasets compared to the static, pre-trained base model, validating the efficacy of our adaptive, LLM-guided approach to maintaining high-performance anomaly detection in dynamic streaming environments.

1 Introduction

Time-series anomaly detection is a fundamental task with profound implications across a wide array of domains, including financial markets, industrial manufacturing, and IT systems monitoring. The timely identification of anomalous events—observations that deviate significantly from expected behavior—can prevent substantial economic losses, preempt system failures, and ensure operational reliability. As data generation becomes increasingly continuous and voluminous, the analysis of these time-series streams has become paramount.

A primary challenge in real-world streaming environments is the phenomenon of *concept drift*, where the statistical properties of the data stream evolve over time. This non-stationarity invalidates the core assumption of many classical machine learning models, which are trained on static, historical datasets. A model that performs well at deployment time may see its accuracy degrade rapidly as new patterns emerge, a problem that necessitates a shift from static modeling to adaptive, online learning paradigms.

The recent advent of large-scale, pre-trained foundation models for time-series, such as TimesFm, represents a significant advancement. Trained on vast and diverse datasets comprising billions of data points, these models exhibit impressive zero-shot forecasting capabilities,

*The first author's contribution was performed during a summer internship at Google in 2025.

providing a powerful baseline for various downstream tasks. However, their generalist nature, while a strength for broad applicability, can be a limitation in specialized, dynamic domains. Research suggests that while these models can be adapted for anomaly detection, they may not outperform specialized approaches without further tuning, particularly when faced with domain-specific concept drift [Fu et al., 2024].

This paper proposes that the solution lies not in choosing between a generalist foundation model and a specialized, adaptive one, but in architecting a system that achieves the best of both. While significant research has advanced scalable stream processing [Akidau et al., 2015], the power of time-series foundation models [Liang et al., 2024], and the reasoning capabilities of LLMs [Zheng et al., 2023], these powerful paradigms have largely evolved in parallel. The critical gap, which this paper addresses, lies in their architectural synthesis. We argue that the next frontier in intelligent systems is not a better isolated algorithm, but a more integrated, self-correcting architecture. The true innovation is a novel architectural synthesis of three powerful and distinct technological paradigms: (1) scalable, stateful stream processing for robust data handling; (2) time-series foundation models for powerful, out-of-the-box forecasting; and (3) Large Language Models (LLMs) as sophisticated reasoning agents. By integrating these components into a cohesive, closed-loop system, we create a new class of intelligent infrastructure capable of autonomous adaptation.

To this end, we introduce CALM (Continuous, Adaptive, and LLM-Mediated), a holistic framework that leverages the zero-shot power of a foundation model as a starting point but integrates it into a fully automated, adaptive pipeline that specializes it over time. The CALM framework makes the following primary contributions:

1. **A Scalable Streaming Architecture:** An end-to-end anomaly detection system architected on Apache Beam, specifically designed to handle high-throughput, out-of-order data streams. It employs stateful processing primitives to ensure correctness and robustness in a distributed environment [Akidau et al., 2015, Apache Beam Community, 2024].

2. **The LLM-as-a-Judge Mechanism:** A novel component that functions as an online, semantic concept drift detector. Moving beyond the typical use of LLMs for offline evaluation [Patronus AI, 2025], our judge actively curates the fine-tuning dataset by providing reasoned judgments on anomalies, distinguishing transient noise from genuine pattern shifts to guide the model's adaptation [Feki, 2023].

3. **A Closed-Loop Continuous Fine-Tuning Pipeline:** A fully automated MLOps loop [Feki, 2023, Google Cloud, 2024b] that collects data deemed valuable by the LLM Judge, batches it for training, triggers a fine-tuning job, and enables the dynamic, in-flight deployment of the improved model. This allows the system to autonomously adapt to concept drift without human intervention.

We validate the CALM framework through a rigorous experimental evaluation on the TSB-UAD benchmark, demonstrating that our adaptive, LLM-guided approach yields significant performance improvements over a static, pre-trained model.

2 Related Work

2.1 Taxonomy of Time-Series Anomaly Detection

A time series is a sequence of data points indexed chronologically [Salehi et al., 2024]. Such series can be *univariate*, consisting of a single variable observed over time, or *multivariate*, comprising multiple variables recorded concurrently [Liang et al., 2024]. Anomalies within these series are broadly categorized into three types: *point anomalies*, which are single data points deviating from the norm; *contextual anomalies*, which are normal in a global sense but abnormal within

their local context; and *collective anomalies*, which are sequences of data points that, as a group, represent an anomalous pattern [Chandola et al., 2009, Hawkins, 1980].

The methodologies for detecting these anomalies are diverse, but a common taxonomy classifies them into four main categories: forecasting-based, reconstruction-based, representation-based, and hybrid methods [Salehi et al., 2024, Blázquez-García et al., 2021]. The CALM framework falls squarely within the **forecasting-based** paradigm. The fundamental principle of this approach is to train a model to predict future values of the time series based on its recent history. The deviation between the model's forecast and the actual observed value, often termed the prediction error or residual, serves as the anomaly score [Laptev et al., 2015, Malhotra et al., 2015]. A data point is flagged as an anomaly if this error surpasses a predefined threshold [Blázquez-García et al., 2021, Hundman et al., 2018]. Our work refines this by using the model's quantile forecasts to establish a dynamic and robust statistical threshold.

2.2 The Landscape of Time-Series Foundation Models

Inspired by the transformative success of LLMs in Natural Language Processing (NLP), a recent paradigm shift has occurred in time-series analysis towards the use of large, pre-trained foundation models, often called Time-Series Foundation Models (TSFMs) [Liang et al., 2024, Wen and Liang, 2024]. These models are trained on massive, heterogeneous time-series corpora, enabling them to learn a rich variety of temporal patterns and exhibit impressive zero-shot performance on downstream tasks [Azad and Garza, 2024, Arnab et al., 2024].

A key architectural innovation enabling TSFMs is **patching**, where the input time series is partitioned into contiguous "patches" that are treated as tokens [Nie et al., 2023]. This reduces the sequence length processed by the transformer, improving computational efficiency and allowing the model to learn local semantic meaning [Peixeiro, 2025]. Several prominent TSFMs have emerged, each with distinct architectural philosophies. **TimesFm**, the model used in our framework, is a decoder-only model trained to predict the next patch based on preceding ones, a design that naturally accommodates variable-length inputs and longer output horizons [Arnab et al., 2024, Google Cloud, 2024a]. Similarly, **Chronos** tokenizes time-series values via quantization into a fixed vocabulary, treating forecasting as a pure language modeling problem within a decoder-only framework [Ansari et al., 2024]. In contrast, **Moirai** uses a masked encoder-based architecture, akin to BERT, and introduces "any-variate attention" by flattening multivariate series into a single sequence [Peixeiro, 2025, Dataloop AI, 2024]. Other models like **PatchTST** focus on channel-independent patching for multivariate series [Nie et al., 2023], while commercial offerings like **TimeGPT** leverage transformer architectures with conformal prediction to provide robust uncertainty intervals [Nixtla, 2024, Olivares et al., 2023]. A brief comparison is provided in Table 1.

Table 1: Comparison of Time-Series Foundation Models

Model	**Core Architecture**	**Key Feature(s)**	**Primary Use Case**
TimesFm	Decoder-only Transformer	Patching; Long output patches	Zero-shot Forecasting
Chronos	Decoder-only Transformer	Value Quantization; Language Model	Probabilistic Forecasting
Moirai	Encoder-only Transformer	Any-variate Attention; Mixture Dist.	Multivariate Forecasting
PatchTST	Encoder-only Transformer	Channel-Independent Patching	Multivariate Forecasting
TimeGPT	Transformer	Conformal Prediction	Commercial Forecasting/AD

While these models demonstrate powerful zero-shot capabilities, their generalist nature can be a limitation in specialized domains with unique data characteristics and concept drift [Alnegheimish et al., 2024a]. Research indicates that for challenging tasks like anomaly detection, fine-tuning or linear probing on task-specific data is often necessary to achieve state-of-the-art performance [Goswami et al., 2024]. CALM is built on this premise: it leverages the zero-shot power of a TSFM as a strong starting point but integrates a continuous fine-tuning mechanism to specialize it for a dynamic target environment.

2.3 LLM-based Approaches to Anomaly Detection

The application of LLMs to time-series anomaly detection is a burgeoning field with several distinct methodologies emerging. These can be broadly categorized as follows:

1. **Direct Prompting:** These methods convert the numerical time-series into a textual representation and directly query an LLM to identify anomalous points. For example, the "Prompter" pipeline within the SigLLM framework serializes the time series and asks the LLM to output the indices of any anomalies [Alnegheimish et al., 2024a]. While conceptually simple, this approach has been found to be less effective than forecasting-based methods [Alnegheimish et al., 2024b].

2. **Forecasting-based Detection:** This paradigm, which CALM employs, uses an LLM as a forecasting engine. The LLM predicts future values, and anomalies are flagged based on a significant residual error between the forecast and the actual observations. The "Detector" pipeline in SigLLM is a prime example of this approach [Alnegheimish et al., 2024a].

3. **Representation and Distillation:** More architecturally integrated approaches use the LLM as a powerful feature extractor. **AnomalyLLM** [Yu et al., 2024] proposes a knowledge distillation framework where a smaller student network is trained to mimic the feature representations of a larger, frozen LLM-based teacher network. Anomalies are then detected by a large discrepancy between the student and teacher outputs. Similarly, **TriP-LLM** [Yu et al., 2025] uses a frozen, pre-trained LLM to process patch-wise tokens from three distinct branches (capturing local, selected, and global features) before a lightweight decoder reconstructs the input for anomaly scoring.

4. **Multimodal Approaches:** A cutting-edge direction involves converting time-series into images and leveraging Vision-Language Models (VLMs). The **VisualTimeAnomaly** benchmark [Luo et al., 2025] demonstrates that VLMs can be robust anomaly detectors, particularly for range-based anomalies, and are surprisingly resilient to missing data, showcasing the potential of treating anomaly detection as a visual reasoning task.

CALM utilizes the forecasting-based paradigm but innovates by embedding this capability within a fully automated, adaptive loop. Its primary novelty lies not in the forecasting method itself, but in the integration of an LLM-based semantic filter to guide the continuous adaptation of the forecasting model.

2.4 Online Learning and Adaptation to Concept Drift

A fundamental limitation of models trained offline is their inability to cope with **concept drift**, the phenomenon where the underlying data distribution of a stream changes over time [Gama and Žliobaitė, 2014]. This drift renders static models obsolete and necessitates **online learning** or continuous adaptation strategies to maintain performance [Lu et al., 2018, Feki, 2023]. Traditional approaches to drift detection can be categorized as statistical or performance-based. Statistical methods use tests like the Kolmogorov-Smirnov (KS) test to detect changes

in the input data distribution [Ditzler et al., 2015]. Performance-based methods monitor the model's output, such as its error rate, to detect degradation [Gama and Žliobaitė, 2014].

Modern frameworks often use a hybrid approach. For instance, **ADDAEIL** [Zhang et al., 2025] combines statistical distribution tests (KS and Mann-Whitney U tests) with an analysis of the internal structure of its ensemble model to detect drift and selectively retrain degraded components. However, these quantitative methods have inherent limitations. Statistical tests can be overly sensitive, triggering unnecessary retraining in response to harmless noise, while performance-based metrics may fail to detect significant data drift if it does not immediately degrade the chosen metric [Sobolewski et al., 2023].

The CALM framework's continuous fine-tuning loop is a direct and practical implementation of an adaptive strategy. Crucially, it replaces purely statistical drift detection with a *semantic* one. The LLM-as-a-Judge performs a qualitative assessment of whether a pattern change is meaningful, mimicking the cognitive process of a human analyst and addressing the brittleness of purely quantitative drift detectors. This aligns with established MLOps principles for Continuous Integration (CI), Continuous Delivery (CD), and Continuous Training (CT) of machine learning systems, which emphasize automation, monitoring, and iterative deployment to combat model decay [Feki, 2023, Google Cloud, 2024b, Huyen, 2025]. CALM operationalizes these principles in a fully automated, closed-loop system driven by the data stream itself.

2.5 Large Language Models as Reasoning Agents in ML Pipelines

The role of LLMs is rapidly expanding from standalone applications to integral components within larger computational systems. A significant body of research has explored the use of LLMs for data annotation and labeling, automating tasks that are traditionally manual, costly, and time-consuming [Yu et al., 2024, Tan et al., 2024]. LLMs can generate not only categorical labels but also rich auxiliary information like rationales and confidence scores [He et al., 2023].

More recently, the **"LLM-as-a-Judge"** paradigm has emerged, primarily for the task of *evaluating* the outputs of other AI models [Zheng et al., 2023, Amazon Web Services, 2024]. These LLM judges have demonstrated a remarkable ability to provide nuanced, context-aware assessments that align closely with human preferences and are more informative than traditional metrics [Patronus AI, 2025]. This is an active area of research, with frameworks like **MCTS-Judge** [Wang et al., 2025] exploring how to enhance judge reliability for complex, reasoning-intensive tasks by incorporating structured search algorithms like Monte Carlo Tree Search.

The CALM framework represents a significant evolution of this concept, shifting the role of the LLM from a passive evaluator to an active controller. Instead of using an LLM for offline evaluation or one-shot annotation, we deploy it as an **active, online, semantic filter** within the data processing pipeline itself. The LLM is not merely labeling data; it is executing a sophisticated reasoning task to guide the learning process of the downstream anomaly detection model. This moves beyond data annotation toward intelligent data curation for online adaptation.

This approach effectively automates the role of a human domain expert. In traditional human-in-the-loop systems, an analyst is often required to validate alerts, distinguish true anomalies from false positives, and ultimately decide if a model needs retraining in response to a pattern shift [Feki, 2023]. The LLM-as-a-Judge in CALM performs this exact cognitive function. By analyzing the data context before and after an outlier and deciding whether it signifies a "transient, one-off event" or a "sustained shift," the LLM is making a determination about concept drift. It acts as a cognitive automaton, automating a high-level reasoning task that is a critical bottleneck in conventional adaptive systems. This represents a step toward more autonomous, self-correcting ML systems.

3 The CALM System Architecture

3.1 High-Level Architectural Overview

The CALM framework is an end-to-end streaming pipeline designed for scalability, robustness, and continuous adaptation. The high-level architecture, depicted in Figure 1, illustrates the closed-loop flow of data and models. The process begins with the ingestion of a potentially unordered stream of univariate time-series data points. This data undergoes stateful windowing and cleaning to prepare it for inference. Anomaly detection is performed using a dynamically loaded TimesFm model. Detected anomalies are then passed to the LLM-as-a-Judge for semantic classification. Anomalies deemed significant are collected and used to trigger a fine-tuning job, which produces an updated model. This new model is then seamlessly deployed back into the detection stage, completing the adaptive loop.

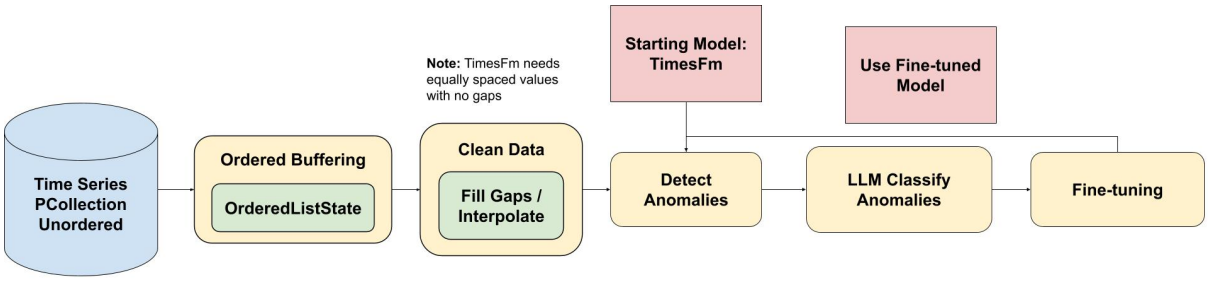


Figure 1: The high-level architecture of the CALM framework. Unordered data is processed through a series of stateful transformations, including ordered buffering and cleaning. Anomaly detection uses a dynamically swappable TimesFm model. The LLM Judge curates anomalies for the fine-tuning stage, which produces an improved model to be used in the next cycle.

3.2 Stateful Ingestion and Windowing on Apache Beam

To handle the challenges of real-world data streams, which are often high-volume and delivered with out-of-order timestamps, the framework is built on Apache Beam, a unified model for defining both batch and streaming data-parallel processing pipelines.

Handling Out-of-Order Data A naive windowing approach would fail on out-of-order data, leading to incomplete or incorrect windows. We implement a custom stateful sliding-window function to handle this. As shown in Figure 2, incoming elements are first buffered in state (per key) instead of being immediately emitted.

- **Stateful Ordered Buffer:** We use a persistent in-memory buffer that maintains events sorted by their timestamp (leveraging Beam's ordered state primitive). This ensures that even if elements arrive out of order (e.g., a timestamp 2 arrives after timestamp 3), they are placed in the correct order within the buffer.
- **Watermark Timer:** We employ a watermark-driven timer to determine when a window is ready to be emitted. The timer is set to fire at the end of each window interval, and it will only fire once the event-time watermark has passed the window's end (indicating no more late data is expected for that interval). When the timer fires, the buffered events for that window are collected and emitted downstream as a complete window. After emission, the state buffer is cleaned up (removing old data), and a new timer is set for the next sliding window.

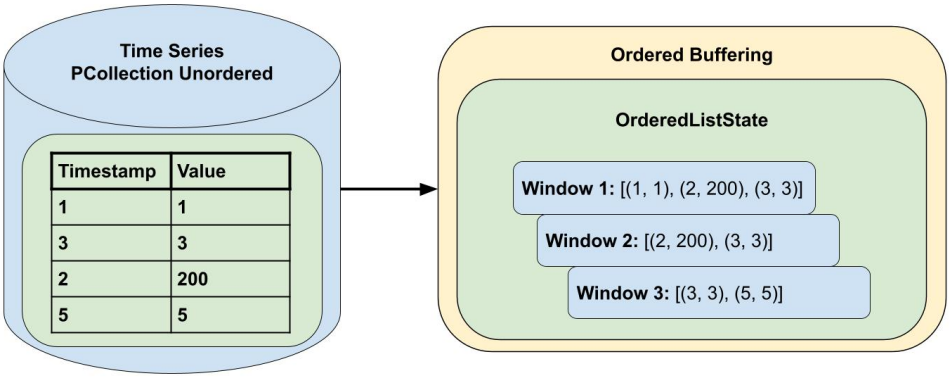


Figure 2: The process of ordered buffering. An unordered PCollection of time-series points is added to a stateful buffer ("OrderedListState"), which maintains the elements sorted by timestamp. This allows for the correct construction of overlapping sliding windows.

Data Cleaning and Interpolation Foundation models like TimesFm need clean, regularly spaced data. We include a data-cleaning step that checks each window for missing timestamps based on the expected interval. As illustrated in Figure 3, this gap-filling function inserts a placeholder value (e.g., "NaN") whenever a timestamp is missing, ensuring the window has a value at every expected time step. These placeholders can then be replaced using an appropriate interpolation strategy (e.g., linear or forward-fill) before the data is passed to the model.

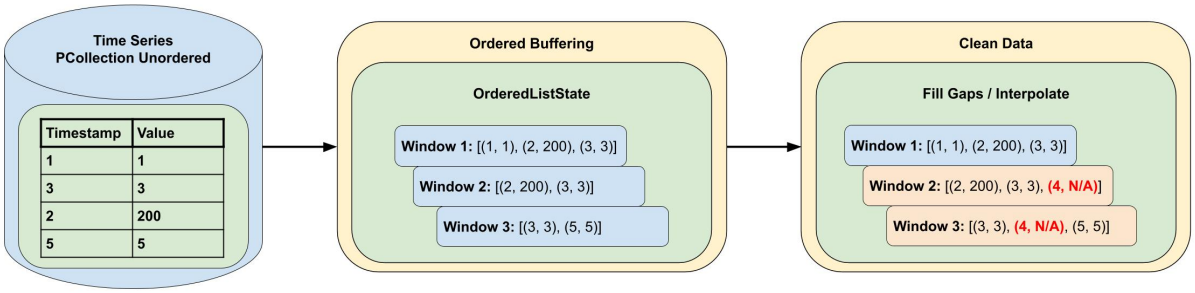


Figure 3: The data cleaning process. The gap-filling function takes a window of data and ensures it has a value for every expected timestamp. Missing values (e.g., at timestamp 4) are filled with a placeholder, preparing the data for the model, which requires regularly spaced inputs.

3.3 Dynamic Anomaly Detection with TimesFm

Forecasting-based Detection The core anomaly detection logic uses a forecasting model (TimesFm) on each window of data. We developed a custom inference component (integrated with Beam's "RunInference" transform) that processes each window by splitting it into a context portion and a prediction horizon. The model takes the context and produces a forecast for the horizon. Rather than using a single-point prediction, we leverage the model's quantile forecasts (e.g., q_{20}, q_{30}, q_{70}, q_{80}) to establish a robust threshold for anomalies. We compute the interquartile range (IQR) from these quantiles, and flag a data point as anomalous if the actual value lies outside the range $[Q1 - 1.5 \times IQR,\ Q3 + 1.5 \times IQR]$. This statistical approach is less sensitive to non-Gaussian error distributions than one based on a fixed standard deviation.

Dynamic Model Swapping A cornerstone of the CALM framework is its ability to adapt the model on the fly. We achieve this by using Apache Beam's side input mechanism – an

auxiliary, broadcast input that can provide configuration data to all workers. In our case, the side input continuously supplies the path of the latest fine-tuned model to the inference pipeline. The pipeline monitors a storage location (e.g., a local or remote bucket) for new model files. Whenever the fine-tuning stage produces a new model artifact and saves it to this location, the side input value is updated to the new model's path. The inference component is designed to detect this update and then load the new model weights via a callback in the model handler. This entire process happens in-flight, allowing the streaming pipeline to seamlessly swap in the updated model without downtime.

3.4 The LLM-as-a-Judge for Data Curation

Once an outlier is detected by the forecasting model, it is not immediately used for fine-tuning. Instead, it is first passed to an LLM-based classifier (the "LLM-as-a-Judge" component) for semantic analysis, as depicted in Figure 4.

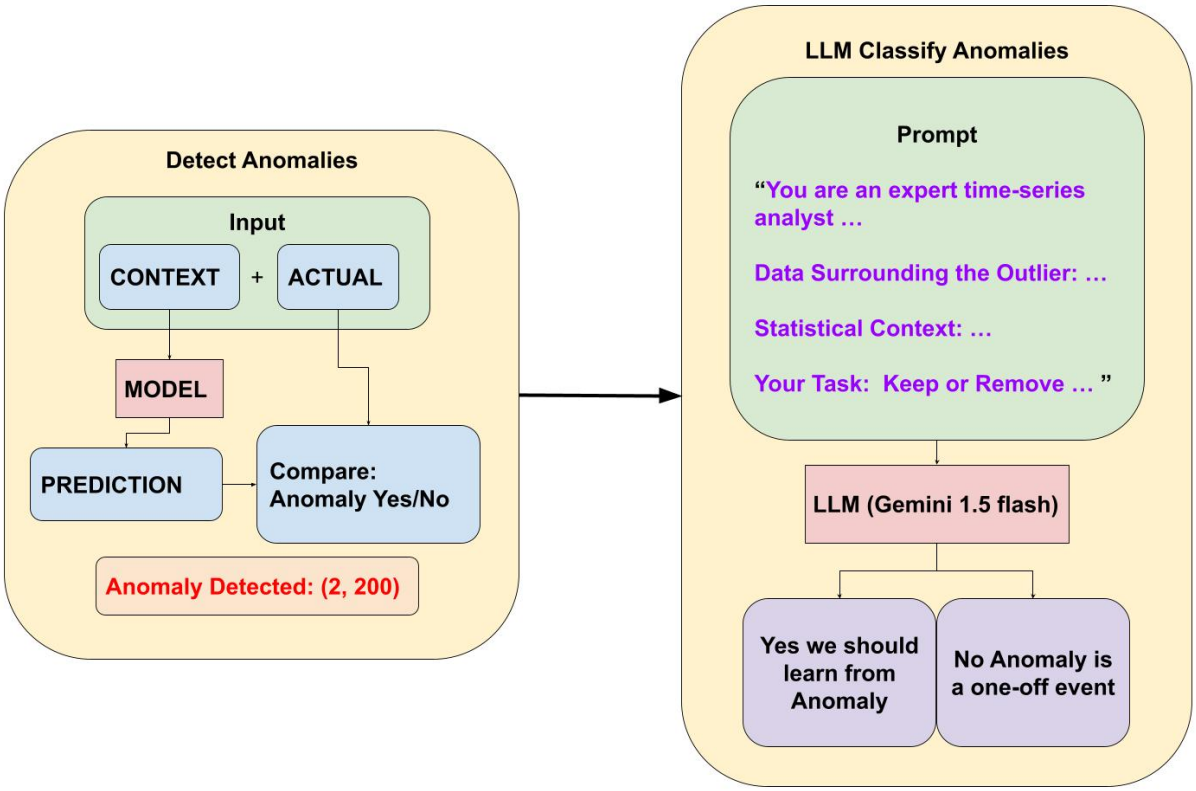


Figure 4: The LLM-as-a-Judge mechanism. A detected anomaly is enriched with surrounding data and statistical context. This is formatted into a detailed prompt and sent to an LLM (e.g., Gemini 1.5 Flash), which judges whether the anomaly represents a pattern to be learned ("KEEP") or a one-off event to be ignored ("REMOVE").

Structured Prompting for Contextual Reasoning The LLM-as-a-Judge constructs a detailed, structured prompt designed to provide the LLM with all necessary context to make an informed judgment. The prompt includes:

- **Persona Setting:** "You are an expert time-series analyst..." to prime the model for the task.
- **Outlier Details:** The precise timestamp, actual value, predicted value, and anomaly bounds.

- **Surrounding Data:** A configurable number of data points immediately preceding and following the anomaly.
- **Statistical Context:** The mean and standard deviation of the data before and after the anomaly.

The LLM is then tasked to decide whether to "KEEP" the anomaly (if it signifies a sustained shift in the pattern) or "REMOVE" it (if it's a transient, one-off event).

Delayed Judgment with Stateful Context A practical challenge arises if an anomaly occurs at the end of a window, leaving little "data after" for the LLM to consider. To handle this, our LLM-based judge uses a stateful approach to delay its decision until sufficient future context is available. In simpler terms, if an anomaly is detected but there are not enough subsequent points to provide context, the system will hold that anomaly aside temporarily. It buffers the anomaly and accumulates incoming data points from subsequent windows. A timer is set to defer the LLM evaluation for a short interval. Once enough new data has been collected to give a complete picture around the anomaly, the timer triggers and the LLM is invoked on the buffered data. This ensures the LLM always receives a full context (both before and after the outlier) before making its judgment.

Output and Routing The LLM returns its verdict in a structured JSON format (including a rationale and a confidence score). If the decision is "KEEP," the original data point (with its anomaly value) is forwarded to the fine-tuning pipeline. If the decision is "REMOVE," we treat that point as a spurious outlier and replace its value with the model's predicted value (effectively cleaning the anomaly from the data stream). In this way, only validated pattern shifts (labeled "KEEP") are used to update the model, while isolated anomalies ("REMOVE") are filtered out.

3.5 Qualitative Analysis of the LLM Judge

To provide insight into the reasoning capability of the LLM Judge, we present a qualitative example. Consider an anomaly detected in a dataset monitoring server CPU usage.

Input to LLM: The anomaly is a sudden spike to 95% CPU usage. The surrounding data shows usage typically hovering around 30%, but in the period immediately following the spike, usage stabilizes at a new, higher baseline of 60%.

Generated Prompt Snippet:

```
...
**2. Data Surrounding the Outlier:**
* Data Before (25 points): [31.2, 29.8,..., 30.5]
* Data After (25 points): [62.1, 59.8, 61.5,..., 60.7]

**3. Statistical Context:**
* Mean Before: 30.15
* Mean After: 60.88
...
**4. Your Task:**
... Classify the outlier.
* REMOVE: If it is a transient, one-off event...
* KEEP: If it signifies a sustained shift in the pattern...
```

LLM JSON Response:

```
{
```

```
  "reasoning_steps": "The data after the outlier does not revert
   to the previous mean of ~30. Instead, it establishes a new,
   stable pattern around a mean of ~61. This indicates the
   outlier was not a transient error but the beginning of a new
   system state. This is a significant and sustained shift.",
  "decision": "KEEP",
  "confidence_score": 0.95
}
```

This example demonstrates the LLM's ability to go beyond the statistical fact of the outlier and perform contextual reasoning. It correctly identifies the sustained shift in the mean as evidence of a meaningful change, justifying its decision to "KEEP" the data point for fine-tuning. This qualitative result highlights the value of the LLM as a semantic filter.

3.6 The Continuous Fine-Tuning Loop

Batching for Fine-Tuning The data points labeled "KEEP" by the LLM Judge are buffered and aggregated into batches for model training. We use a stateful batching mechanism that accumulates these points and only emits a batch when it has collected a full, continuous sequence of a predefined length. In other words, it waits until it can form a contiguous time-series segment (of size N, the batch size) before outputting it. This ensures that the data sent for fine-tuning constitutes a valid continuous window of time-series data, which is necessary for training the forecasting model.

Orchestrating the Fine-Tuning Job When a batch of new data is ready, the system launches a fine-tuning process for the model. The first step is to determine the starting model: if a fine-tuned model from an earlier cycle exists, we use that; otherwise, we fall back to the original pre-trained model. Next, the batch of "KEEP" data is split into training and validation sets (for example, using an 80/20 split) to create a proper fine-tuning dataset. We then initiate a fine-tuning run on this data (using the TimesFm training routine). Upon completion, the updated model weights are saved as a new model artifact in persistent storage (for example, writing a file like 'timesfm_finetuned_{timestamp}.pth' to a local or cloud storage bucket).

Closing the Loop After the model is saved, the pipeline's model-watching mechanism picks up the new model version. In our implementation, once the new model file appears in the storage location, it triggers an update to the side input discussed earlier. The anomaly detection stage then automatically loads the latest model for subsequent inferences. This closes the continuous learning loop: the system has now incorporated the newly fine-tuned model into production, and the pipeline returns to monitoring for the next anomalies and drift events.

4 Experimental Evaluation

4.1 Benchmark and Dataset Selection

To empirically validate the CALM framework, we conducted our evaluation on the **TSB-UAD benchmark** [Paparrizos et al., 2022], a comprehensive suite for unsupervised time-series anomaly detection. From the TSB-UAD v2 release, we selected a subset of **33 datasets** that met two specific criteria: (1) they contained multiple labeled anomaly groups, providing sufficient events for the model to potentially learn from, and (2) they had anomalous points present in the final 20% of the series. This second criterion was essential to ensure we could fairly evaluate the fine-tuned model's performance on a strictly held-out set of future anomalies.

4.2 Evaluation Methodology and Metric Justification

Our experiment followed a strict temporal split to simulate a real-world scenario. For each dataset:

1. The **first 80%** of the data served as the "live" stream for fine-tuning.
2. The **final 20%** was a strict hold-out test set, used only for evaluation.

The primary goal of CALM is not just to improve forecast accuracy but to enhance the model's fundamental ability to **distinguish between normal and anomalous states**. For this reason, we chose the **Receiver Operating Characteristic (ROC) Area Under the Curve (AUC)** score as our primary metric.

ROC AUC is ideal because it measures a model's discriminative power across all possible thresholds. This is crucial for validating our LLM-powered data curation. A key risk of naive fine-tuning is overfitting to noise; if a model learns to treat random, one-off spikes as normal, its forecast boundaries widen, and it fails to flag real anomalies. The LLM Judge is designed to prevent this by filtering out transient noise and retaining only anomalies that signify genuine pattern shifts. Therefore, a higher ROC AUC score is direct proof that the fine-tuned model has learned a more robust, generalizable understanding of "normalcy," making it a superior anomaly detector.

4.3 Results and Analysis

The experiments demonstrate a clear and positive impact from the LLM-judged fine-tuning process.

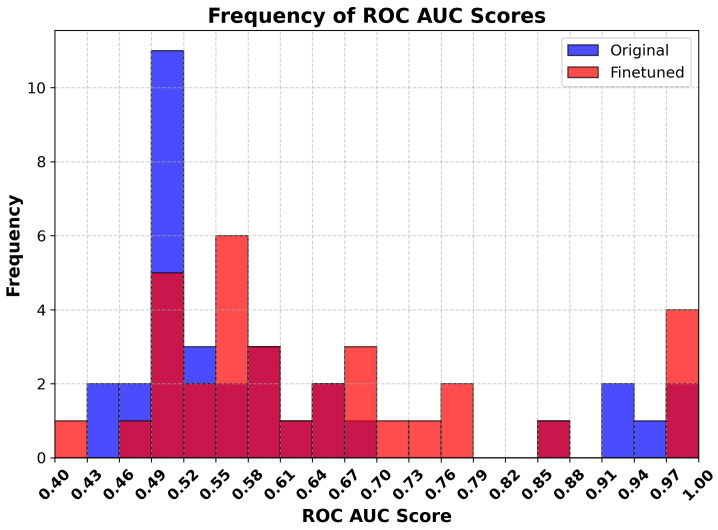


Figure 5: Distribution of ROC AUC scores across 45 TSB-UAD datasets. The panel shows overlaid histograms comparing the original and CALM-Tuned models. The visualization clearly illustrates that the LLM-guided fine-tuning process shifts the distribution of performance toward higher AUC values, with a more concentrated mass of datasets achieving scores between 0.6 and 1.0.

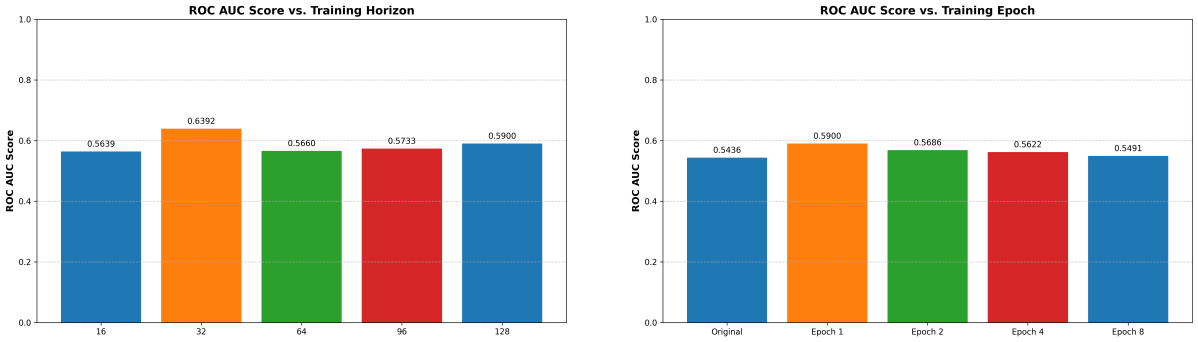
Table 2: Top and Bottom 10 Datasets by ROC AUC Improvement (Δ)

Top 10				Bottom 10			
Dataset	**Orig.**	**Tuned**	**Impr.**	**Dataset**	**Orig.**	**Tuned**	**Impr.**
9.3..74_col_3	0.499	0.851	+0.351	WSD_79	0.923	0.691	-0.232
NAB_data...._4	0.503	0.748	+0.245	stock...0.25_5	0.607	0.500	-0.107
proc...-3-10_col_6	0.727	0.934	+0.207	stock...0.15_2	0.879	0.788	-0.091
8.3..73_col_2	0.568	0.774	+0.207	SWaT...._col_58	0.512	0.430	-0.082
101-freeway-traffic	0.454	0.648	+0.194	KPI-e0747cad...	0.480	0.423	-0.057
6.1..65_col_5	0.439	0.600	+0.161	9.3..74_col_1	0.592	0.552	-0.040
WSD_37	0.565	0.674	+0.110	stb-6	0.500	0.463	-0.037
proc...-2-7_col_34	0.469	0.557	+0.088	WSD_183	0.587	0.558	-0.028
stb-5	0.504	0.583	+0.079	SWaT...._col_60	0.501	0.480	-0.021
stb-32	0.456	0.527	+0.071	SED_20000_0	0.457	0.437	-0.019

Table 2 Top 10 shows that the framework can yield substantial gains, especially on datasets where the original model performed poorly (AUC near 0.5). This highlights its ability to adapt a general model to specific, previously challenging patterns. Conversely, Table 2 Bottom 10 provides a balanced view, showing cases where performance decreased. This often occurred on datasets where the original model was already highly performant, suggesting the fine-tuning process may have slightly overfit to the limited anomalies in the training portion. This represents a clear area for future work, such as developing more intelligent triggers for the fine-tuning process or introducing noise before finetuning to prevent overfitting.

4.4 Hyperparameter Sensitivity Analysis

To better understand the behavior of the CALM framework, we conducted a sensitivity analysis on two key hyperparameters using the WSD_94 dataset: the prediction horizon length and the number of fine-tuning epochs. These experiments, shown in Figure 6, provide insight into the framework's robustness and help identify optimal settings for deployment.



(a) Impact of Prediction Horizon

(b) Impact of Training Epochs

Figure 6: ROC AUC score sensitivity to prediction horizon (a) and number of fine-tuning epochs (b) on the WSD_94 dataset.

Impact of Prediction Horizon We first investigated how the model's prediction horizon affects its anomaly detection performance (Figure 6a). We varied the horizon length, testing values of 16, 32, 64, 96, and the default of 128. The results indicate that the framework's performance is surprisingly robust to changes in this parameter. While a horizon of 32 yields the highest ROC AUC score of 0.6392, the scores for other horizons remain competitive. This

phenomenon can likely be attributed to the TimesFM architecture, which is pre-trained to output a fixed-length forecast (e.g., 128 steps) that is subsequently truncated if asked for a shorter horizon.

Impact of Training Epochs Next, we examined the effect of the number of fine-tuning epochs (Figure 6b). The most substantial performance gain is achieved within the very first epoch, where the ROC AUC score jumps from 0.5436 (original model) to 0.5900. This demonstrates the rapid effectiveness of the CALM framework's adaptation mechanism. However, subsequent epochs yield marginal improvements, suggesting that the model begins to overfit. This pattern indicates that a short fine-tuning cycle (e.g., one epoch) is both sufficient and optimal for adapting to concept drift without significant overfitting.

5 Discussion and Future Work

The CALM framework demonstrates a powerful architectural pattern for building next-generation, self-adapting monitoring systems. The fusion of scalable stream processing, pre-trained foundation models, and LLM-based reasoning agents provides a robust blueprint that could be extended beyond anomaly detection to other adaptive tasks like dynamic forecasting and online classification.

Limitations Despite its promising results, the framework has several limitations that warrant discussion. First, the inclusion of LLM API calls and periodic fine-tuning jobs introduces both monetary **cost and processing latency**. There is an inherent trade-off between the frequency of adaptation (and thus responsiveness to drift) and the operational cost of the system. Second, the framework's performance is partially dependent on the **reliability and consistency of the LLM judge**. While powerful, LLMs can be sensitive to prompt phrasing and may occasionally produce inconsistent outputs, which could impact the quality of the fine-tuning data. This contrasts with the formal, albeit potentially brittle, statistical guarantees of methods like ADDAEIL [Zhang et al., 2025]. Third, the current implementation is focused on **univariate time-series**, a limitation inherited from the base TimesFm 1.0 model [Arnab et al., 2024]. This is a significant constraint, as many real-world systems require detecting anomalies based on inter-variable correlations [Zhao et al., 2020, Liang et al., 2024].

Future Work These limitations point to several exciting avenues for future research:

- **Parameter-Efficient Fine-Tuning (PEFT):** To mitigate the cost and time of full fine-tuning, future versions could explore PEFT methods [Lialin et al., 2023, Kanerika Inc., 2024]. By training only a small subset of the model's parameters, PEFT could enable more frequent and cheaper adaptation cycles. A particularly promising technique is **Low-Rank Adaptation (LoRA)** [Hu et al., 2021], which injects small, trainable low-rank matrices into the model. Recent work has already demonstrated the successful application of LoRA to time-series foundation models, making it a viable path for optimizing the CALM framework [Bansal et al., 2024].
- **Extension to Multivariate Time-Series:** A significant extension would be to adapt the framework for multivariate anomaly detection. This would require replacing the univariate TimesFm with a multivariate-capable foundation model. Promising candidates include **Moirai**, which is designed for "any-variate" time series by flattening variables into a single sequence [Peixeiro, 2025], or adopting architectures like **DualLMAD**, which uses parallel GPT-2 backbones to explicitly model temporal and inter-metric dependencies [Zhang et al., 2024]. This extension would also necessitate developing more complex logic for the LLM Judge to reason about inter-variable relationships.

- **Advanced LLM Judging Strategies:** The judging component offers significant room for advancement. This includes upgrading the reasoning engine itself. While our work uses Gemini 1.5 Flash for efficiency, employing more powerful models could enhance reasoning capabilities. More importantly, the interaction logic could be improved. Inspired by frameworks like **MCTS-Judge** [Wang et al., 2025], which uses Monte Carlo Tree Search to guide an LLM to more reliable conclusions in complex domains like code evaluation, a future version of CALM could implement a multi-step verification or chain-of-thought reasoning process for the judge. Furthermore, using an ensemble of different LLM judges could improve robustness and reduce the impact of idiosyncratic errors from a single model.

- **Optimizing the Fine-Tuning Trigger:** The current trigger for fine-tuning is based on collecting a fixed-size batch of data. A more intelligent trigger could be developed, for example, based on the rate of "KEEP" decisions from the LLM. A high rate would signal that significant concept drift is occurring, justifying an immediate fine-tuning cycle, thereby making the system's adaptivity more responsive to the dynamics of the data stream.

- **Robustness through Noise Augmentation:** A critical issue observed in fine-tuning is that the model can overfit to new "normal" data, causing the predicted quantiles $[q_{0.1}, q_{0.9}]$ to shrink. To investigate a solution, we conducted a preliminary experiment with **noise augmentation**, a regularization technique also known as jittering [Wen et al., 2021].

  The `noise_level` is a critical hyperparameter that controls the intensity of the Gaussian noise, $N(0, \sigma^2_{\text{noise}})$, added to the training data. To ensure the noise is appropriately scaled to the data's magnitude and variance, we define the standard deviation of the noise (σ_{noise}) as a fraction of the standard deviation of the entire training time series (σ_{data}). This relationship is formalized by the equation:

  $$\sigma_{\text{noise}} = \sigma_{\text{data}} \times \texttt{noise\_level}$$

  Therefore, for our experiment with a `noise_level` of 0.01, the standard deviation of the injected noise is precisely 1% of the standard deviation of the training data. This proportional scaling makes the augmentation strategy robust, as it automatically adapts the noise intensity to the specific statistical characteristics of any given time series. We implemented a `NoisyTimeSeriesDataset` class that injects Gaussian noise into the training data on-the-fly. For each sample requested by the dataloader, noise is generated and added to the numpy arrays of the context and horizon windows. The standard deviation of this noise is proportional to the standard deviation of the entire training series, controlled by a `noise_level` hyperparameter, which we set to 0.01 for this experiment. This on-the-fly approach ensures that noise is applied only to the training set and that the model sees a different random perturbation in each epoch, which is an effective regularization strategy [Bishop, 1995].

  The results of this initial experiment are shown in Figure 7. Without noise (the solid line), the model's performance peaks at Epoch 1 and then degrades, a classic sign of overfitting. When noise is introduced (the dashed line), the performance curve becomes flatter. While the peak performance is slightly lower, the degradation in later epochs is mitigated. This suggests that noise augmentation is successfully acting as a regularizer, forcing the model to learn more general features instead of memorizing the training data.

  These findings are highly promising. A clear direction for future work is to perform a comprehensive hyperparameter search to identify the optimal `noise_level`. A well-tuned noise level could not only prevent overfitting but also lead to a higher overall performance by allowing the model to train for more epochs without performance degradation.

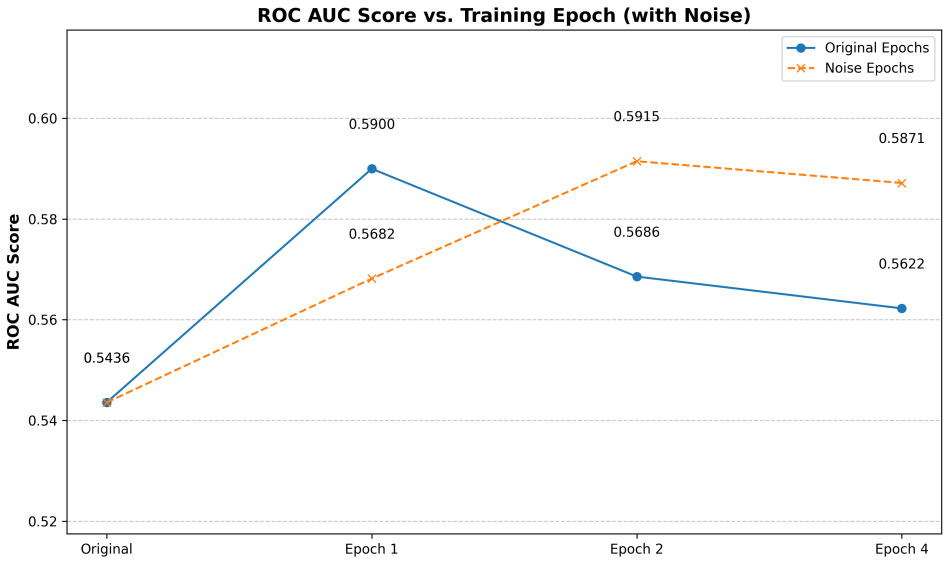


Figure 7: Comparison of ROC AUC scores over training epochs with and without noise augmentation (`noise_level=0.01`). Noise acts as a regularizer, mitigating the performance degradation from overfitting seen in later epochs.

6 Conclusion

This paper addressed the critical challenge of anomaly detection in non-stationary time-series streams. We introduced CALM, a novel framework that provides a robust and scalable solution by synthesizing three key technologies. It leverages Apache Beam for stateful stream processing, the TimesFm foundation model for powerful forecasting, and a Large Language Model as an intelligent, semantic judge to curate data for adaptation. The core of the framework is a fully automated, closed-loop continuous fine-tuning pipeline that allows the system to autonomously adapt to concept drift. Our empirical evaluation on the TSB-UAD benchmark confirmed that this adaptive, LLM-guided approach leads to significant performance gains over a static, pre-trained model. The CALM architecture serves as a blueprint for a new class of intelligent, self-adapting monitoring systems capable of operating effectively in dynamic, real-world environments.

Acknowledgments

The authors would like to express their gratitude to the Dataflow ML team at Google for their invaluable support and collaboration. Special thanks go to Xiangqian Hu for guidance, Claude van der Merwe for co-hosting and mentorship, and Danny McCormick for his insightful feedback. The first author also wishes to thank his co-author, Shunping Huang, for serving as his host during the internship and for his continual support throughout the project.

References

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803, 2015.

Sarah Alnegheimish, Linh Nguyen, Laure Berti-Equille, and Kalyan Veeramachaneni. Large language models can be zero-shot anomaly detectors for time series? *Accepted to IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2024a. Preprint available at https://dai.lids.mit.edu/large-language-models-can-be-zero-shot-anomaly-detectors-for-time-series/.

Sarah Alnegheimish, Linh Nguyen, Laure Berti-Equille, and Kalyan Veeramachaneni. Large language models can be zero-shot anomaly detectors for time series? https://consensus.app/papers/large-language-models-can-be-zeroshot-anomaly-detectors-veeramachaneni-berti-%C3%A9quille/2cb9b682fe8958eea002ac047e98f0c5/, 2024b. Accessed: August 16, 2025.

Amazon Web Services. Llm as a judge on amazon bedrock model evaluation. https://aws.amazon.com/blogs/machine-learning/llm-as-a-judge-on-amazon-bedrock-model-evaluation/, 2024. Accessed: August 16, 2025.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, and Yisong Yue. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024.

Apache Beam Community. Stateful processing - apache beam. https://beam.apache.org/documentation/programming-guide/#stateful-processing, 2024. Accessed: August 16, 2025.

Anurag Arnab, Viorica Pătrăucean, Mostafa Dehghani, Georgios Papamakarios, Carl-Johann Simon-Gabriel, Methma Liyanage, Alexey Gritsenko, Neil Houlsby, Basil Mustafa, Piotr Padlewski, Roman Ring, Jasper Uijlings, Olivier J. Hénaff, and Razvan Pascanu. A decoder-only foundation model for time-series forecasting. *arXiv preprint arXiv:2310.10688*, 2024.

MD. Mainul Azad and Anthony Garza. Timegpt vs snowflake - 50x faster forecasting with better accuracy. *Nixtla Blog*, 2024.

Harshavardhan Bansal, Abhishek Ananth, S. V. R. K. Chaitanya, and Manish Singh. Low-rank adaptation of time series foundational models for out-of-domain modality forecasting. *arXiv preprint arXiv:2405.10216*, 2024.

Christopher M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995. doi: 10.1162/neco.1995.7.1.108.

A. Blázquez-García, A. Conde-Clemente, D. Corral-Plaza, P. García-Bringas, and D. Gutiérrez-Avilés. A review on outlier/anomaly detection in time series data. *ACM Computing Surveys (CSUR)*, 54(8):1–33, 2021.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. In *ACM computing surveys (CSUR)*, volume 41, pages 1–58. ACM New York, NY, 2009.

Dataloop AI. Salesforce moirai 1.0 r small. https://dataloop.ai/library/model/salesforce_moirai-10-r-small/, 2024. Accessed: August 16, 2025.

Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.

Rihab Feki. Mlops #02: 7 things you need to learn about continuous training & continuous deployment. *Medium*, Feb 2023.

Xinghong Fu, Masanori Hirano, and Kentaro Imajo. Financial fine-tuning a large time series model. *arXiv preprint arXiv:2412.09880*, 2024. preprint.

João Gama and Žliobaitė. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

Google Cloud. Use the timesfm model. https://cloud.google.com/bigquery/docs/timesfm-model, 2024a. Accessed: August 16, 2025.

Google Cloud. Mlops: Continuous delivery and automation pipelines in machine learning. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning, 2024b. Accessed: August 16, 2025.

Manish Goswami, Kshitij Szafer, A. Choudhry, Y. Cai, S. Li, and A. Dubrawski. Moment: A family of open time-series foundation models. *arXiv preprint arXiv:2402.03885*, 2024.

Douglas M. Hawkins. Identification of outliers. 1980.

Jun-Yan He, Zikang Lin, Zihan Chen, Yifu Zhang, Wenhao Li, and Richong Huang. Large language models can be good annotators for content analysis: A preliminary study. *arXiv preprint arXiv:2307.03050*, 2023.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 387–395, 2018.

Chip Huyen. Mlops and llmops crash course—part 1. https://www.dailydoseofds.com/mlops-crash-course-part-1/, 2025.

Kanerika Inc. Parameter-efficient fine-tuning (peft): Optimizing llms. https://kanerika.com/blogs/parameter-efficient-fine-tuning/, 2024. Accessed: August 16, 2025.

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 and Data Mining*, pages 1939–1947, 2015.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*, 2023.

Yuxuan Liang, Haomin Wen, Yuqi Nie, Yushan Jiang, Ming Jin, Dongjin Song, Shirui Pan, and Qingsong Wen. Foundation models for time series analysis: A tutorial and survey. *arXiv preprint arXiv:2403.14735*, 2024.

Jie Lu, An Liu, Fang Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12): 2346–2363, 2018.

Yifan Luo, Yiming Zhang, Jian wei Liu, Zhen-Yu Zhang, Zhi-Gao Liu, Wei-Long Zheng, and Bao-Liang Lu. Can multimodal llms perform time series anomaly detection? *arXiv preprint arXiv:2502.17812*, 2025.

Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2015.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2023.

Nixtla. Timegpt-1: production ready pre-trained time series foundation model. https://github.com/Nixtla/nixtla, 2024. Accessed: August 16, 2025.

Kin G. Olivares, Cristian Challu, Magnus W. M. Kjaero, Anthony Garza, Max Mergenthaler-Canela, David Valdes, Federico Garza, Stefania Tel-Zur, and Artur Dubrawski. Timegpt-1. *arXiv preprint arXiv:2310.03589*, 2023.

John Paparrizos, Yixuan Kang, Paul Boniol, Ruey S. Tsay, Themis Palpanas, and Michael J. Franklin. Tsb-uad: An end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(8):1697–1711, 2022.

Patronus AI. Llm as a judge. https://www.patronus.ai/llm-testing/llm-as-a-judge, 2025. Accessed: August 16, 2025.

Marco Peixeiro. Hands-on with moirai: A foundation forecasting model by salesforce. *DataSciencewithMarco.com*, May 2025.

Mohammadreza Salehi, III Le-Heng Harris, Su-Yang Yu, Ryan A. Rossi, Rui-Jie Yew, II An-Kuei, II Sung-Chul, II Han-Yu, Hootan Nakhost, II Ali, II Jonathan, and II Giorgio. A survey on deep learning for time series anomaly detection. *arXiv preprint arXiv:2211.05244*, 2024.

Pawel Sobolewski, Przemyslaw Biecek, and Hubert Baniecki. On the evaluation of data-based drift detection approaches. *Scientific Reports*, 13(1):2846, 2023.

Zhen Tan, Alimohammad Beigi, Song-Chun Zhu, and Reza Haf. Large language models for data annotation: A survey. *arXiv preprint arXiv:2402.13446*, 2024.

Yutong Wang, Pengliang Ji, Chaoqun Yang, Kaixin Li, Ming Hu, Jiaoyang Li, and Guillaume Sartoretti. Mcts-judge: Test-time scaling in llm-as-a-judge for code correctness evaluation. *arXiv preprint arXiv:2502.12468*, 2025.

Haomin Wen and Yuxuan Liang. Foundation models for time series analysis. 2024. Accessed: August 16, 2025.

Qingsong Wen, Linchen Sun, Fan Yang, Xiaomin Song, Jing Gao, Xiaogang Wang, and Huiting Xu. Time series data augmentation for deep learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)*, pages 4673–4680, 2021. doi: 10.24963/ijcai.2021/637.

Yuan-Cheng Yu, Chun-An Lin, Yen-Chieh Ouyang, and Wen-Chih Peng. Large language model guided knowledge distillation for time series anomaly detection. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*, pages 2161–2169, 2024.

Yuan-Cheng Yu, Yen-Chieh Ouyang, and Chun-An Lin. Trip-llm: A tri-branch patch-wise large language model framework for time-series anomaly detection. *arXiv preprint arXiv:2508.00047*, 2025.

Yifan Zhang, Zhi-Gao Liu, Wei-Long Zheng, and Bao-Liang Lu. Addaeil: Anomaly detection with drift-aware ensemble-based incremental learning for non-stationary time series. *Mathematics*, 18(6):359, 2025.

Zhiyuan Zhang, Zhi-Qiang Liu, Shiyang Liu, Yifeng Gao, Zitao Liu, Wenyu Du, and Guandong Xu. Duallmad: A dual pre-trained language models based framework for multivariate time series anomaly detection. *Accepted to International Symposium on Software Reliability Engineering (ISSRE)*, 2024. Preprint available at https://nkcs.iops.ai/wp-content/uploads/2024/08/ISSRE24-DualLMAD.pdf.

Han Zhao, Yuxiang Wang, Juanyong Duan, Chen-Kuo Huang, Dongjie Cao, Yanjie Wang, and Leman Akoglu. Multivariate time-series anomaly detection via graph attention network. pages 841–850, 2020.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.