Large language models can be zero-shot anomaly detectors for time series?

Sarah Alnegheimish[1] , **Linh Nguyen**[1] , **Laure Berti-Equille**[2] and **Kalyan Veeramachaneni**[1]
[1]MIT
[2]IRD ESPACE-DEV
{smish, linhnk, kalyanv}@mit.edu, laure.berti@ird.fr,

Abstract

Recent studies have shown the ability of large language models to perform a variety of tasks, including time series forecasting. The flexible nature of these models allows them to be used for many applications. In this paper, we present a novel study of large language models used for the challenging task of time series anomaly detection. This problem entails two aspects novel for LLMs: the need for the model to identify part of the input sequence (or multiple parts) as anomalous; and the need for it to work with time series data rather than the traditional text input. We introduce SIGLLM, a framework for time series anomaly detection using large language models. Our framework includes a time-series-to-text conversion module, as well as end-to-end pipelines that prompt language models to perform time series anomaly detection. We investigate two paradigms for testing the abilities of large language models to perform the detection task. First, we present a prompt-based detection method that directly asks a language model to indicate which elements of the input are anomalies. Second, we leverage the forecasting capability of a large language model to guide the anomaly detection process. We evaluated our framework on 11 datasets spanning various sources and 10 pipelines. We show that the forecasting method significantly outperformed the prompting method in all 11 datasets with respect to the F1 score. Moreover, while large language models are capable of finding anomalies, state-of-the-art deep learning models are still superior in performance, achieving results 30% better than large language models.

1 Introduction

Large language models (LLMs) have demonstrated an outstanding ability to learn natural language tasks implicitly, whether performing reading comprehension, text summarization, translation, or related tasks. Radford et al. [2019]; Brown et al. [2020]; Sanh et al. [2022]; Chowdhery et al. [2023]; Wei et al. [2022]. Moreover, LLMs have shown tremendous promise for formal language generation, including code generation and synthesis Xu et al. [2022a]; Austin et al. [2021]; Chen et al. [2021], and in production beyond textual output, such as generating images and videos from natural language descriptions Saharia et al. [2022]; Koh et al. [2023]. Testing these models on new tasks and data modalities allows us to push the boundaries of LLMs and discover their value. In this paper, we present a thorough study of LLMs used for the challenging task of anomaly detection from time series data, asking the question *Can LLMs become anomaly detectors for time series data?* Here, LLMs are exposed to a new data type – time series – and are tasked with a detection task – different from the classification tasks in which they are known to excel at Howard and Ruder [2018].

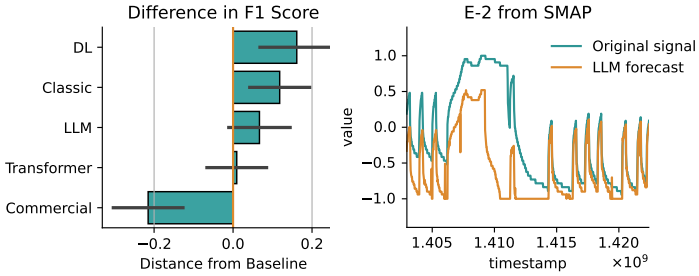

Figure 1: (left) F1 Score performances of different model types, compared to a moving average baseline. Each category represents a collection of models that fall under that group. For *classic* models, we consider ARIMA and Matrix Profiling; for *Deep Learning (DL)*, we utilize AER and LSTM DT; for *transformer* anomaly detection models, we look at Anomaly Transformer; lastly, for the *commercial* category, we compare to MS Azure. (right) Illustration of MISTRAL forecasts on E-2 signal from the SMAP dataset. The deviation between the signals can help identify anomalous regions.

Recently, Gruver et al. [2023] posited that large language models have an inherent auto-regressive feature which allows them to be effective forecasters. In the study, the authors fed a string representation of a time series sequence to a pre-trained LLM. The LLM then generated the next expected values, treating time series forecasting as a next-token prediction task. A follow-up question ari QWERT]śes: Does LLMs' auto-regressive nature allow them to take on more complex tasks, such as anomaly detection? State-of-the-art time series anomaly detection models using deep learning typically include a forecasting model as one of the steps in their process Hundman et al. [2018].

Time series anomaly detection is a regular part of day-to-day industry operations. Identifying unusual patterns can be a cumbersome and difficult task, especially when massive amounts of signal must be analyzed. If LLMs are genuine anomaly detectors, and can be employed directly in *zero-shot* (without any additional training), they could serve as off-the-shelf anomaly detectors for users, lifting a considerable amount of this burden. Considering that training deep learning models is time-consuming, skipping this phase could make anomaly detection more efficent overall.

In this paper, we present SIGLLM, a framework for using LLMs to detect anomalies in time series data, with current interaction support for models hosted by OpenAI [1] and HuggingFace [2]. Our framework includes a signal-to-text representation component to convert time series data into LLM-ready input. Moreover, we present two distinct approaches to investigating our main question. First, PROMPTER is a simple and direct prompting method, which elicits LLMs to identify the parts of a sequence it thinks are anomalous. Second, DETECTOR leverages LLMs' ability to forecast time series to find anomalies, by using the residual between the original signal and the forecasted one.

Our findings, captured in Figure 1(left), show that LLMs improve on a simple moving average baseline. Moreover, they outperform transformer-based models such as Anomaly Transformer Xu et al. [2022b]. However, there is still a gap between classic and deep learning approaches and LLMs. Furthermore, between our two approaches, DETECTOR is superior to PROMPTER, with an improvement of 135% in F1 Score, as the latter suffers from false positives. We highlight the potential of the DETECTOR approach in Figure 1(right), which showcases an example of an LLM forecast. We can clearly see that the LLM forecast is substantially different from the original signal; this difference is attributed to the presence of anomalies.

We summarize our contributions as follows:

- **Propose a new application for LLMs–anomaly detection–and study their efficacy and efficiency for this task.** We formalize a new task to present to LLMs; namely, time series anomaly detection in *zero-shot*.
- **Present the SIGLLM framework with a time-series-to-text representation module and two novel methodologies for solving this task.** We present SIGLLM with a module to convert time series data into language-model-ready input through a series of reversible transformations. Moreover, we propose two distinct approaches for solving the problem: the PROMPTER pipeline and the DETECTOR pipeline. As of this writing, our framework integrates propriety models such as GPT by OpenAI and open models provided on the HuggingFace `transformers` package.
- **Provide a comprehensive and thorough evaluation of LLM performance on this task.** We conduct our experiments on two prominent LLM models – GPT-3.5-turbo and MISTRAL-7B-Instruct-v0.2 – and 11 datasets. We show that LLMs are able to find anomalies with an average F1 score of 0.525. Moreover, we compare SIGLLM methods to 10 other existing methods including state-of-the-art models such as AER Wong et al. [2022].
- **Publish an open source software.** Our code and datasets are publicly available on github: https://github.com/sintel-dev/sigllm.

[1] https://platform.openai.com/docs/models

[2] https://huggingface.co/models

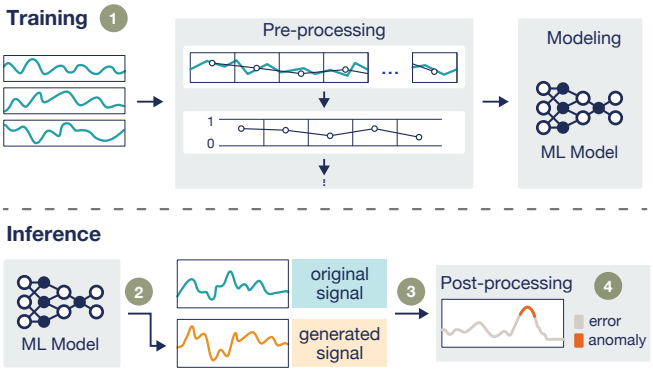

Figure 2: General principle of how machine learning models find anomalies in an unsupervised setting. Step 1: Apply a sequence of preprocessing operations and train a machine learning model to learn the pattern of the data. This is the most time-consuming step; Step 2: Use the trained model to generate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

2 Background and Related Work

Anomaly Detection Pipelines. Unsupervised machine learning-based anomaly detection pipelines generally follow the same sequence of steps, which roughly consist of pre-processing, modeling, and post-processing abstractions as presented in Figure 2 Alnegheimish et al. [2022]. Pre-processing operations include scaling the time series into a specific range, while post-processing includes computing discrepancies between two sequences. A wide variety of models can be trained to learn the features of input data, including Long-Short-Term-Memory models (LSTMs) Hundman et al. [2018], AutoEncoders (AE) Malhotra et al. [2016], Variational AutoEncoders (VAE) Park et al. [2018], Generative Adversarial Networks (GANs) Geiger et al. [2020], and Transformers Xu et al. [2022b]; Tuli et al. [2022]. These models perform well on existing benchmarks such as Alnegheimish et al. [2024], surpassing the performance of statistical approaches such as ARIMA Box and Pierce [1970]; Pena et al. [2013].

Transformers for Time Series. Transformers can be used directly to reconstruct time series Tuli et al. [2022]. Moreover, the attention mechanism can be leveraged to find anomalous sequences Xu et al. [2022b]. Recently, more work has emerged adopting transformer-based models for forecasting purposes by pretraining large transformer models on a large corpus of time series data. FORECASTPFN Dooley et al. [2023] pre-trains a basic encoder-decoder transformer with one multi-head attention layer and two feedforward layers on a synthetically generated time series dataset. Similarly,

TIMEGPT was pretrained on a large collection of publicly available time series datasets. LAG-LLAMA Rasul et al. [2023] is a decoder-only LLAMA-2 model pretrained on a large corpus of real time series data from diverse domains. Moreover, CHRONOS Ansari et al. [2024] adopts a T5 architecture, and parses time series data into text to pretrain their model. Most of these models were developed with the objective of creating a time series foundation model for time series forecasting.

LLMs for Time Series. The past several months have seen considerable efforts toward LLM utilization for time series data. Given the parallels between predicting the next word in a sentence and predicting the next value in a time series, most of these efforts have focused on time series forecasting. One notable effort is LLMTIME where Gruver et al. [2023] employ GPT Brown et al. [2020], and LLAMA-2 Touvron et al. [2023] models to forecast time series data. PROMPTCAST Xue and Salim [2023] is a related work that translates a forecasting problem into a prompt, transforming forecasting into a question-answering task.

Our Work. In this paper, we work strictly with LLMs that have been pre-trained on text, particularly a proprietary model using GPT-3.5 Brown et al. [2020] and an open source model using MISTRAL Jiang et al. [2023]. Our main objective is to determine whether LLMs have the ability to directly uncover anomalies in time series data. Referring back to Figure 2, our methodology focuses on Step 2 onwards – primarily the inference phase. To our knowledge, there is no other work that utilizes large language models as zero-shot anomaly detectors for time series data. We explore two avenues for accomplishing this task: (a) Through the paradigm of prompt engineering; (b) By leveraging LLMs' ability to forecast time series in *zero-shot* without any additional data or fine-tuning.

3 Time Series Representation

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

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

Other scaling methods, such as min-max scaling, can be utilized to achieve the same goal. However, reducing the set of possible values to a smaller range (e.g. [0, 1]), may cause a loss of information in the quantization step. On the other hand, increasing the range will mean there are more digits to tokenize. With our approach, we simply shift the range of the signal values, which allows us to reduce the number of individual digits that need to be tokenized while maintaining the original gaps between pairs of entries. Moreover, by projecting the values into a non-negative range, we eliminate the need for sign indicator "-/+" and save an additional token.

Quantization. Unlike the finite set of vocabulary words used to train LLMs (32k vocab tokens for MISTRAL) [3], the set of scaled time series values x_{s_t} is infinite, and cannot be processed by language models. Therefore, time series that are to be used with LLMs are generally quantized Ansari et al. [2024]; Gruver et al. [2023]. We use the rounding method, as proposed in in Gruver et al. [2023]. Because in some cases the number of decimal digits are redundant given a fixed precision, we round each value up to a predetermined number of decimals, and subsequently scale to an integer format to avoid wasting tokens on the decimal point. Hence, the input time series becomes $\mathbf{X}_q = (x_{q_1}, x_{q_2}, \ldots, x_{q_T})$, where $x_{q_t} \in \mathbb{Z}_{\geq 0}$. Below is an example of this operation :

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

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

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

Tokenization. Different tokenization schemes vary in how they treat numerical values. Several open-source LLMs, such as LLAMA-2 Touvron et al. [2023] and MISTRAL Jiang et al. [2023], utilize the SentencePiece Byte-Pair Encoding tokenizer Touvron et al. [2023], which segments numbers into individual digits. However, the GPT tokenizer tends to segment numbers into chunks that may not correspond directly with the individual digits Liu and Low [2023]. For instance, the number 234595678 is segmented into chunks [234, 595, 678] and assigned token IDs [11727, 22754, 17458]. Empirical evidence suggests that this segmentation impedes the LLM's ability to learn patterns in time series data Gruver et al. [2023]. To make sure GPT tokenizes each digit separately, we adopt the approach introduced by Gruver et al. [2023], which inserts spaces between the digits in a number.

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

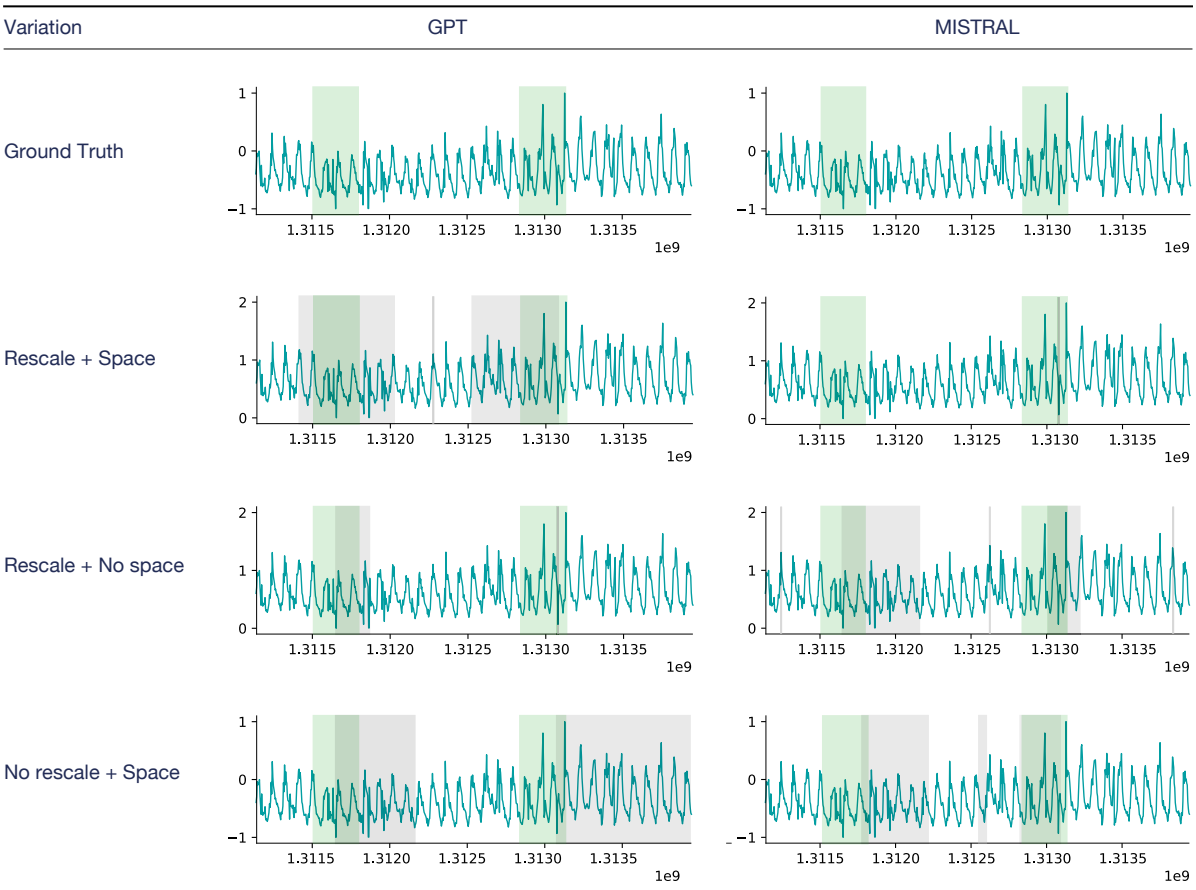

Figure 3: Visualizing the output of large language models (GPT and MISTRAL) under different variations of the transformation process. Each row depicts the `exchange-2_cpm_results` signal from the AdEx dataset, where the x-axis shows the timestamp and the y-axis is the signal value. The first row indicates the ground truth anomalies present in the time series (highlighted in green). The remaining rows indicate whether scaling and inserting space between digits has occurred during the conversion from signal to text. The gray intervals highlight the anomalies detected under these conditions; thus, we would like to maximize the overlap between the green and gray intervals. Overall we find that "scaling + space" is the configuration that yields a better output for GPT; and "scaling + no space" is better for MISTRAL.

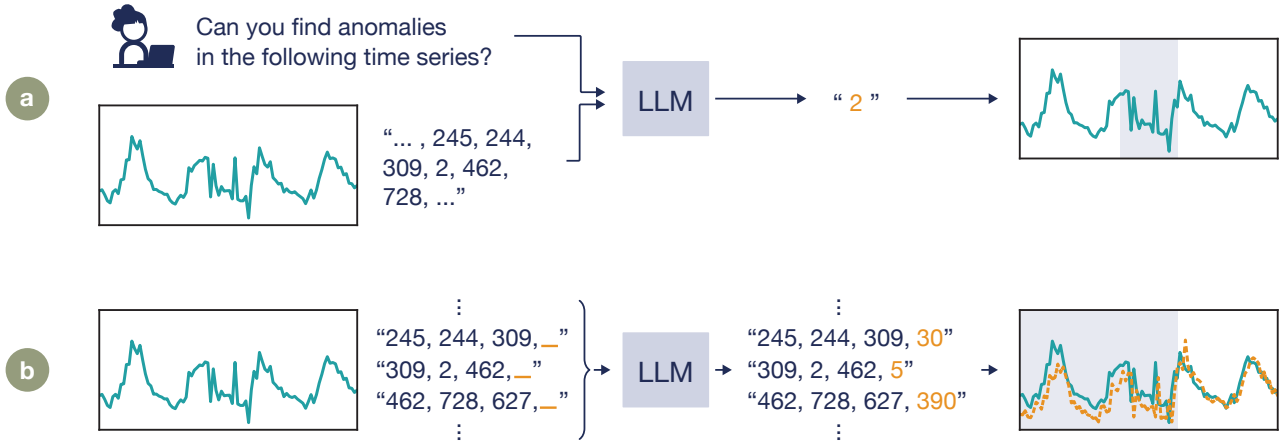

Figure 4: Anomaly detection methods in the SIGLLM framework. (a) PROMPTER: a prompt engineering approach to elicit large language models to identify parts of the input which are anomalies. (b) DETECTOR: a forecasting approach to use large language models as forecasting methods. DETECTOR then finds discrepencies between the original and forecasted signal, which indicate the presence of anomalies.

Continuing with the running example:

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

Where each digit is now encoded separately.
Figure 3 shows how different preprocessing steps affect the output of the model. Overall, we find that scaling reduces the number of tokenized digits, and yields better results than not scaling. Moreover, GPT performs better with added space between digits, while MISTRAL does not. These results accord with the forecasting representation presented in Gruver et al. [2023].

4 SIGLLM: Detecting Anomalies in Signals using Large Language Models

Given a univariate time series $\mathbf{X} = (x_1, x_2, \ldots, x_T)$, and assuming there exists a set of anomalies of varied length $\mathbf{A} = \{(t_s, t_e)^i \mid 1 \leq t_s < t_e \leq T\}_{i=1}^m$ that is unknown *a priori*, our goal is to find a set of m anomalous time segments, where t_s and t_e represent the start and end time points of an anomalous interval. We introduce two fundamentally different methods that can be used for anomaly detection with LLMs: PROMPTER and DETECTOR, as visualized in Figure 4.

4.1 PROMPTER: Finding Anomalies through Prompting

As depicted in Figure 4, this pipeline involves querying the LLMs directly for time series anomalies through a text prompt (as shown below) concatenated with the processed time series window $u^i_{1\ldots k} \coloneqq \text{prompt} \oplus (x^i_{1\ldots w})$, where k is the total length of the input after concatenation. LLMs will output the next token u_{k+1} sampled from an autoregressive distribution conditioned on the previous tokens $p_\theta(u_{k+1}|u_{1\ldots k})$.

Following a series of experiments, as shown in Table 1, we iterated over trial #5 and arrived at the following prompt for our study:

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

Under this prompt, the LLM generates a list of *values* it delineates as point-wise anomalies. It is noteworthy that the GPT-3.5-turbo model is capable of directly outputting anomalous indices using the prompt presented in Table 1, while MISTRAL lacks this ability, as shown in trial #5. To maintain consistency across our experiments, we conducted experiments on both models using the same prompt mentioned above.

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

4.2 DETECTOR: Finding Anomalies through Forecasting

As depicted in Figure 2, the first step in a typical ML pipeline involves training an ML model on a collection of time series. From Gruver et al. [2023], pretrained LLMs are capable of forecasting time series, allowing us to jump straight to the inference phase.

Pre-processing. As detailed in Section 3, our first step involves transforming a raw input into a textual representation, and creating samples ready for the LLM from the rolling window sequences $\{(x^i_{1\ldots w})\}_{i=1}^N$.

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

Post-processing. For each time point t, we now have multiple forecasted values $\hat{x}_t$ in different windows when the horizon is larger than 1, concretely $\{(\hat{x}_t^{i+h})\}$. We take the median from the collection as the final predicted value for $\hat{x}_t$. Furthermore, to increase the reliability of the prediction, we take n samples from the distribution for each window i. Therefore, each x_t has n samples. To map this back to a univariate time series, we explore the results by taking the mean, median, 5^{th}-percentile, and 95^{th}-percentile as values. For the purpose of anomaly detection, an extreme forecast value could indicate a precursor to an anomaly; therefore, acute values can be informative (see Section 5.1). Now, we have reconstructed the time series as $\hat{\mathbf{X}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_T)$.

We next compute the discrepancy between $\mathbf{X}$ and $\hat{\mathbf{X}}$. A large discrepancy indicates the presence of an anomaly. We denote this discrepancy as an error signal e by computing point-wise residuals, given their simplicity and ease of interpretation. We explore the usage of absolute difference suggested by Hundman et al. [2018] $e_t = |x_t - \hat{x}_t|$. Moreover, we explore how other functions, such as squared difference $e_t = (x_t - \hat{x}_t)^2$ will help reveal the location of anomalies. More complex functions that capture the difference between

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

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

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

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

two signals, such as dynamic time warping Müller [2007] can be used. However, Geiger et al. [2020] shows that discrepancies found with point-wise errors are sufficient for this purpose. Moreover, we apply an exponentially weighted moving average to reduce the sensitivity of the detection algorithm Hundman et al. [2018]. Error values that surpass the threshold are considered anomalous. We use a sliding window approach to compute the threshold to help reveal contextual anomalies that are abnormal compared to the local neighborhood. As such, we assign the window size and step size to $T/3$ and $T/10$ respectively. We set a static threshold for each sliding window as four standard deviations away from the mean. These hyperparameters were chosen based on preliminary empirical results that agree with previous settings in other approaches Hundman et al. [2018]; Geiger et al. [2020]; Wong et al. [2022].

5 Evaluation

In this section, we assess our framework and seek to answer the following research questions:

- **RQ1** Are large language models effective anomaly detectors for univariate time series?
- **RQ2** How does the SIGLLM framework compare to existing approaches?
- **RQ3** What are the success and failure cases and why?

Datasets. We examined SIGLLM on 11 datasets with known ground truth anomalies. These datasets were gathered from a wide range of sources, including a satellite telemetry signal corpus from **NASA** [6] that includes two sub-datasets: SMAP and MSL; **Yahoo S5** [7], which contains four sub-datasets: A1, which is based on real production traffic to Yahoo systems, and three others (A2, A3, and A4) which have been synthetically generated; and **NAB** [8], which includes multiple types of time series data from various application domains. We consider five sub-datasets: Art, AWS, AdEx, Traf, and Tweets. In total, these datasets contain 492 univariate time series and 2,349 anomalies. The properties of each dataset, including the number of signals and anomalies, the average signal length, and the average length of anomalies, are presented in Table 2. The table makes clear how properties differ between datasets; for instance, the NASA and NAB datasets contain anomalies that are longer than those in Yahoo S5, and the majority of anomalies in Yahoo S5's A3 & A4 datasets are point anomalies.

Models. We used `Mistral-7B-Instruct-v0.2` for PROMPTER and DETECTOR. Moreover, we used `gpt-3.5-turbo-instruct` for PROMPTER alone. Due to cost constraints, we explored the usage of GPT for DETECTOR on a 5% sample of all datasets, which produced similar results to using MISTRAL. We compared SIGLLM to state-of-the-art models in unsupervised time series anomaly detection. This includes a variety of models similar to the ones considered in Wong et al. [2022]; Alnegheimish et al. [2022]:

- Classic statistical methods including `ARIMA`, `Matrix Profiling (MP)`, and a simple `Moving Average (MAvg)`.

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

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

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

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

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

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

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

Metrics. We utilized anomaly detection-specific metrics for time series data Tatbul et al. [2018]; Alnegheimish et al. [2022]. Namely, we looked at the F1 score, under which both partial and full anomaly detection are considered correct identification.

Hyperparameters. For PROMPTER, GPU capacity means that the maximum input window length of SMAP and MSL is 500 values (for other datasets, it is 200 values). We chose a step size such that, on average, a value was contained in 5 overlapping windows (i.e, 100 steps for SMAP and MSL, and 40 for others). For DETECTOR, we set the window size to 140 and the step size to 1. With a rolling window strategy of step size 1, it is important to keep the windows as small as possible while still ensuring that they are large enough to make useful predictions, as more context tends to be useful for LLMs. Our preliminary results suggested that a window size of 140 was as performative as a window size of 200, and was better than a window size of 100. We set the horizon to 5.

Computation. For GPT, we used GPT-3.5-turbo due to its superior performance on time series data (demonstrated by Gruver et al. [2023]) and its affordability. For MISTRAL, we used the publicly available model hosted by HuggingFace [9] on an Intel i9-7920X 24 CPU core processor and 128GB RAM machine with 2 dedicated NVIDIA Titan RTX 24GB GPUs. For benchmarking, we use Intel Xeon processor of 10 CPU cores (9 GB RAM per core) and one NVIDIA Volta V100 GPU with 32 GB memory.

5.1 Are large language models effective anomaly detectors for univariate time series?

After running the models on the full datasets, we computed the precision, recall, and F1 scores, shown in Table 3. Overall, MISTRAL achieved better results than GPT for the PROMPTER method, with a 2× improvement in F1 score. In addition, DETECTOR performed better overall than PROMPTER. We investigate each approach below.

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

Table 3: Summary of Precision, Recall, and F1 Score

	Precision	Recall	F1 Score
PROMPTER MISTRAL	0.219 ± 0.108	0.311 ± 0.213	0.223 ± 0.104
PROMPTER GPT	0.162 ± 0.133	0.245 ± 0.191	0.133 ± 0.076
DETECTOR	**0.613** ± **0.184**	**0.514** ± **0.211**	**0.525** ± **0.167**

Ablation Study

PROMPTER. The PROMPTER approach originally produced an extremely high number of anomalies. We introduced the α and β hyperparameters to filter the end result. We performed an ablation study to test multiple combinations of α and β values on the F1 score. For some windows, the LLMs consistently outputted more than half of the window values as anomalous; thus, we discarded the predicted results of all windows containing all 10 samples, which was more than 50% of the windows. Fig 5 shows detection F1 scores from different combinations of α and β values. We observed that on all datasets, for MISTRAL, $\alpha = 0.4$ and $\beta = 0.9$ yielded the best F1 score; for GPT, $\alpha = 0.2$ and $\beta = 0.9$ yielded the best F1 score as shown in Fig 5.

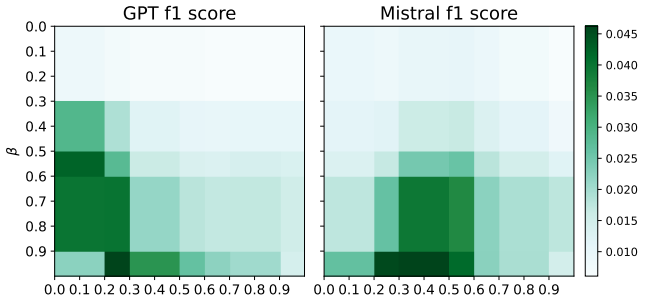

Figure 5: Optimizing the choice α and β values based on the average F1 scores on all datasets.

DETECTOR Since we sampled multiple instances from the probability distribution (namely 10 samples in our experiments), we obtained a possible range of values that could be assigned to each particular point in time. We studied a multiple aggregation function to recreate a one-dimensional signal. Table 4 shows the detection F1 score when the signal is recreated from mean, median, 5^{th}-percentile, and 95^{th}-percentile values of the predicted distribution. Moreover, we consider different error scores under smoothing operation and without.

We can see that, on average, squared error with smoothing on the median signal produced the best score, even though it was not the best performer on any individual dataset. The best configuration proved to be different for almost every

dataset, with squared error producing the best results on Yahoo S5, and absolute error on NAB. One interesting observation is that the signal reconstructed from the $5^{th}/95^{th}$-percentile showcased higher potential in revealing the location of anomalies.

5.2 How does the SIGLLM compare to existing approaches?

Table 5 highlights the F1 score obtained for each of the 11 datasets.

LLM-based methods can outperform transformer-based methods by 12.5%. Comparing DETECTOR to AT, which is a transformer-based method, we see DETECTOR outperforms AT in 7 out of the 11 datasets. However, this observation is only relevant to DETECTOR.

LLM-based methods can perform surprisingly well. Our methods achieved an F1 score 14.6% higher than that of MAvg, and only 10.9% lower than that of ARIMA which is a reasonable model. Moreover, the DETECTOR pipeline alone surpasses MAvg performance in 6 out of 11 datasets, and ARIMA in 4 out of 11 dataset. We can best see the potential of DETECTOR in the Tweets dataset, where the gap between the LLM's result and the highest result (from MAvg) is minimal, at only 1.8%.

AER, the current best deep learning model, is 30% better than LLM-based approaches. Deep learning methods still perform better than LLM-based methods by 18% on average. Looking more closely at DETECTOR shows that it does not perform as well as deep learning models, achieving a 30% lower score than that of the highest performing model (AER) and a 5% lower score than the that of least performative model (VAE). Moreover, LLMs do not hold the highest score for any dataset. It is clear that there is a significant gap here, and an opportunity for improvement.

In the following section, we aim to address where the LLMs succeeded at this problem and where they fell short.

5.3 What are the success and failure cases and why?

Figures 6 and 7 illustrate example outputs for both the PROMPTER and DETECTOR methods, respectively. We focus on MISTRAL since it yielded better overall results than GPT, as depicted in Table 3.

Success Cases. For both signals shown in Figure 7, DETECTOR correctly identified all the anomalies, with only one false alarm in the second signal. The PROMPTER approach is able to detect outliers and locally extreme values a lot more effectively than anomalies that are buried within the signal. For example, if the window is "244 , 309 , 2 , 462", both GPT and MISTRAL point to "2" as the observed anomaly. However, this problem becomes more ambiguous when the context is larger. As seen in Figure 6 on Twitter_volume_AMZN signal, PROMPTER identified some locally extreme values as anomalous. Oddly enough, it also missed some anomalies that were global outliers.

Failure Cases. Even though DETECTOR correctly identified all anomalies in the example shown in Figure 7, the forecast itself struggled to capture the non-stationary aspects of the signal, particularly its trend. This is due to the sensitivity of LLMs to context length. A window size larger than 140 is needed to capture this property. While it did not impact the detection in this case, this may explain failure cases in other signals.

PROMPTER raised a large number of false alarms, with an average precision of 0.219. Using the filtering method described in Section 4.1 does not eliminate false positives. An alternative strategy could be to use log probabilities as a measure of confidence for filtering. We recommend exploring this avenue in future work.

6 Discussion

Prompting Challenges. Over a three-month experimental period, various prompts were employed, as laid out in Table 1. It is evident that both GPT and MISTRAL fail to produce the desired responses unless a chat template is applied that attributes roles to the user and the system. Furthermore, to ensure the exclusivity of numerical values in the generated responses, in addition to specifying in the prompt to "just return numbers," we adjusted the likelihood of non-numerical tokens appearing in the output generated by the LLMs.

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

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

Addressing Memorization. Large language models are trained on a vast amount of data. The training data for most models – for instance, those provided by OpenAI – is completely unknown to the general public, which makes evaluating these models a nuanced problem. Given that large language models, especially GPT models Chang et al., are notorious for memorizing training data Biderman et al. [2023], how do we ensure that there was no data and label leakage for the benchmark datasets used? We posit that our transformation of the time series data into its string representation is unique, essentially making the input time series different from its original form and reducing the chances of blatant memorization. Moreover, unlike with the forecasting task, the task of anomaly detection is not inherent to the training convention used, which is next token prediction.

Practicality of Usage. The appeal of using LLMs for this task lies in their ability to be used in zero-shot, without necessitating any fine-tuning. However, this property is bottlenecked by the latency time. Figure 8 illustrates the average time it takes to use LLMs for each of the suggested approaches. It is unreasonable to wait half an hour to two hours for the model to produce a response, especially when deep learning models take less than an hour to train. Since these experiments were run in an offline setting, we can ex-

Table 4: F1 Score of all variations of DETECTOR

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

Table 5: Benchmark Summary Results depicting F1 Score.

	NASA		Yahoo S5				NAB					
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets	$\mu \pm \sigma$
`AER`	**0.587**	**0.819**	**0.799**	**0.987**	**0.892**	0.709	**0.714**	**0.741**	0.690	**0.703**	0.638	**0.753 ± 0.109**
`LSTM DT`	0.471	0.726	0.728	0.985	0.744	0.646	0.400	0.468	0.786	0.585	0.603	0.649 ± 0.161
`ARIMA`	0.525	0.411	0.728	0.856	0.797	0.686	0.308	0.382	0.727	0.467	0.514	0.582 ± 0.176
`MP`	0.474	0.423	0.507	0.897	0.793	**0.825**	0.571	0.440	0.692	0.305	0.343	0.570 ± 0.193
`TadGAN`	0.560	0.605	0.578	0.817	0.416	0.340	0.500	0.623	0.818	0.452	0.554	0.569 ± 0.142
`LSTM AE`	0.545	0.662	0.595	0.867	0.466	0.239	0.667	**0.741**	0.500	0.500	0.475	0.569 ± 0.158
`VAE`	0.494	0.613	0.592	0.803	0.438	0.230	0.667	0.689	0.583	0.483	0.533	0.557 ± 0.143
`AT`	0.400	0.266	0.571	0.565	0.760	0.576	0.414	0.430	0.500	0.371	0.287	0.467 ± 0.138
`MAvg`	0.171	0.092	0.713	0.356	0.647	0.615	0.222	0.408	**0.880**	0.157	**0.776**	0.458 ± 0.266
`MS Azure`	0.051	0.019	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176	0.243 ± 0.225
PROMPTER MISTRAL	0.160	0.154	0.194	0.235	0.338	0.336	0.370	0.268	0.000	0.135	0.257	0.223 ± 0.104
PROMPTER GPT	0.049	0.110	0.143	0.078	0.157	0.195	0.154	0.194	0.133	0.133	0.197	0.133 ± 0.076
DETECTOR	0.429	0.431	0.615	0.828	0.376	0.363	0.400	0.362	0.727	0.480	0.762	0.525 ± 0.167

pect that real-time deployment would be more sensible, especially since the context size is much smaller, eliminating the need for rolling windows. In other cases, a single window would be sufficient, which takes approximately 5 seconds to infer depending on the window size.

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

7 Conclusion

In this paper, we present large language models with a new and challenging task: detecting anomalies in time series data with no prior learning. To this end, we propose two methods, PROMPTER and DETECTOR, covering the prompting and forecasting paradigms respectively. We demonstrate that LLMs can find anomalies through the forecasting paradigm (DETECTOR method) more accurately than they can through the PROMPTER method. We present SIGLLM, a framework for converting signals into text, enabling LLMs to work with time series data. A major weakness of LLMs is their limited context window size. In SIGLLM, we rely on rolling windows to chop the time series up into smaller segments. This is both inefficient and costly. Moreover, it seriously challenges the possibility of expanding the framework to work with multivariate time series. As LLMs rapidly advance, more models can handle larger context sizes; however, they do not yet have the capacity to handle time series data without segmentation. Even with their limited capability, LLMs are able to beat a well-known transformer method, and to approach the performance of classical methods when tested against 492 signals. They fall short of deep learning models by a factor of $\times 1.2$. In anomaly detection, post-processing strategies are critical for revealing the locations of anomalies. In future work, we plan to investigate post-processing functionalities that can help PROMPTER filter false alarms. Similarly, we plan to conduct a thorough exploration of error functions that bring out anomalies for DETECTOR.

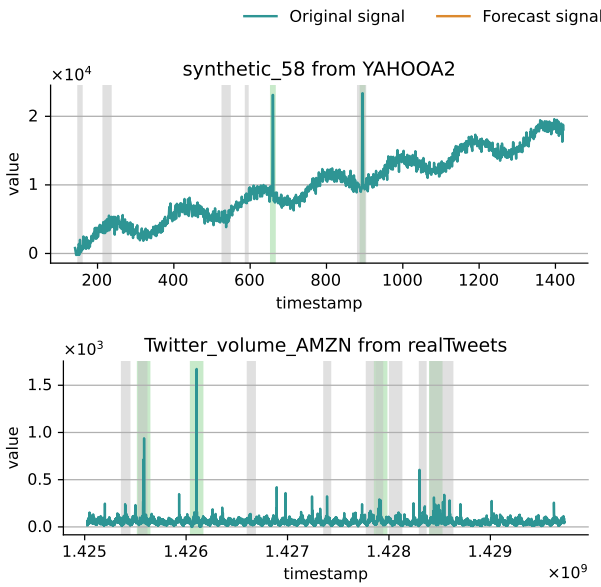

Figure 6: Examples of anomalies identified through PROMPTER. While the model was able to find anomalies, the number of false positives was high, and there were false negatives.

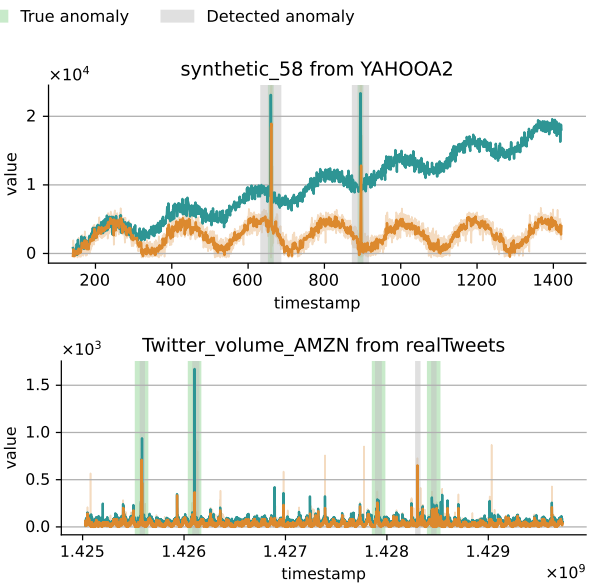

Figure 7: Examples of anomalies successfully identified by DETECTOR. Even though the model did not capture the trend present in `synthetic_58`, it still managed to find the anomalous intervals.

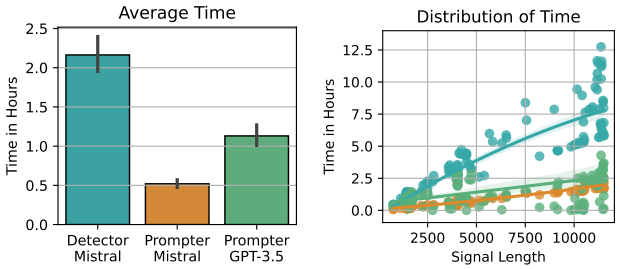

Figure 8: Recorded time for PROMPTER and DETECTOR. (left) On average, DETECTOR takes the longest to infer, almost double the time of PROMPTER. (right) Distribution of signal length and execution time.

Acknowledgments

Our research is supported by SES S.A., Iberdrola and ScottishPower Renewables, and Hyundai Motor Company.

Change Log

Updated the acknowledgment section. Removed the previous version which had acknowledgements to the folks who had helped create the IJCAI template.

References

Sarah Alnegheimish et al. Sintel: A machine learning framework to extract insights from signals. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 1855–1865, New York, NY, USA, 2022. Association for Computing Machinery.

Sarah Alnegheimish et al. Making the end-user a priority in benchmarking: Orionbench for unsupervised time series anomaly detection, 2024.

Abdul Fatir Ansari et al. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024.

Jacob Austin et al. Program synthesis with large language models, 2021.

Stella Biderman et al. Emergent and predictable memorization in large language models. *Advances in Neural Information Processing Systems*, 36, 2023.

George EP Box and David A Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, 65(332):1509–1526, 1970.

Tom Brown et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Kent Chang et al. Speak, memory: An archaeology of books known to ChatGPT/GPT-4. In *EMNLP 2023*.

Mark Chen et al. Evaluating large language models trained on code, 2021.

Aakanksha Chowdhery et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Samuel Dooley et al. Forecastpfn: Synthetically-trained zero-shot forecasting. In *Advances in Neural Information Processing Systems*, volume 36, pages 2403–2426. Curran Associates, Inc., 2023.

Alexander Geiger et al. TadGAN: Time series anomaly detection using generative adversarial networks. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 33–43. IEEE, 2020.

Nate Gruver et al. Large language models are zero-shot time series forecasters. *Advances in Neural Information Processing Systems*, 36, 2023.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics.

Kyle Hundman et al. Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 387–395, 2018.

Albert Q. Jiang et al. Mistral 7b, 2023.

Jing Yu Koh et al. Generating images with multimodal language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 21487–21506. Curran Associates, Inc., 2023.

Tiedong Liu and Bryan Kian Hsiang Low. Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks, 2023.

Pankaj Malhotra et al. LSTM-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

Daehyung Park et al. A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018.

Eduardo H.M. Pena et al. Anomaly detection using forecasting methods ARIMA and HWDS. In *2013 32nd International Conference of the Chilean Computer Science Society (sccc)*, pages 63–66. IEEE, 2013.

Alec Radford et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Kashif Rasul et al. Lag-Llama: Towards foundation models for time series forecasting. *arXiv preprint arXiv:2310.08278*, 2023.

Chitwan Saharia et al. Photorealistic text-to-image diffusion models with deep language understanding. In *Advances in Neural Information Processing Systems*, volume 35, pages 36479–36494. Curran Associates, Inc., 2022.

Victor Sanh et al. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*, 2022.

Nesime Tatbul et al. Precision and recall for time series. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023.

Shreshth Tuli et al. Tranad: deep transformer networks for anomaly detection in multivariate time series data. *Proc. VLDB Endow.*, 15(6):1201–1214, feb 2022.

Jason Wei et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. Survey Certification.

Lawrence Wong et al. AER: Auto-encoder with regression for time series anomaly detection. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 1152–1161. IEEE, 2022.

Frank F Xu et al. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 1–10, 2022.

Jiehui Xu et al. Anomaly Transformer: Time series anomaly detection with association discrepancy. In *International Conference on Learning Representations*, 2022.

Hao Xue and Flora D. Salim. Promptcast: A new prompt-based learning paradigm for time series forecasting, 2023.