LogGenST: A Framework for Synthetic Log Generation using LLMs for Smart-Troubleshooting

Sania Partovian$^{1,2,3*[0009-0009-9081-5476]}$, Francesco Flammini$^{1[0000-0002-2833-7196]}$, Alessio Bucaioni$^{1[0000-0002-8027-0611]}$, and Johan Thornadtsson^{2}

[1] School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden `{sania.partovian,francesco.flammini, alessio.bucaioni}@mdu.se`
[2] Sigma Technology Information, Stockholm, Sweden
`johan.thornadtsson@sigmatechnology.com`
[3] *Corresponding author

Abstract. Log files are essential for monitoring, diagnosing, and troubleshooting smart systems, capturing critical operational events. However, privacy concerns and limited access to real-world log datasets hinder the development and evaluation of anomaly detection techniques. Synthetic log generation has emerged as a viable solution to this challenge, enabling researchers to create diverse datasets that include both normal and fault-related logs. In this paper, we introduce novel methodologies for generating synthetic log files using Generative Adversarial Networks and Large Language Models. First, we propose a GAN-based approach, leveraging different GAN implementations to produce high-quality synthetic logs. A comprehensive evaluation reveals that CTGAN outperforms other models in generating realistic and varied log entries. Building on these findings, we present LogGenST, an innovative synthetic log generation framework that employs three LLMs in an adversarial setup. Unlike traditional GAN-based methods, LogGenST features a unique Prompt Engineer LLM that refines prompts based on feedback from generator and discriminator LLMs. This approach ensures temporal consistency, logical coherence, and domain-specific patterns without requiring extensive model training. Comparative analysis shows that LogGenST significantly enhances log authenticity, pattern consistency, and fault representation, supporting advanced smart-troubleshooting experimentation in industrial cyber-physical systems and the Internet of Things.

Keywords: Generative Adversarial Network · Synthetic Data · Log Files · Industry 4.0 · Smart-Troubleshooting · Large Language Models.

1 Introduction

As technology rapidly advances, systems and applications become increasingly complex, resulting in a diverse array of anomalies that include faults, errors, and failures [1]. Faults refer to defects in a system that may cause errors, errors are deviations from expected behavior, and failures occur when a system's intended function is not performed. Anomalies are challenging for humans to detect because of system complexity, large data volumes, and dynamic environments [2]. Smart-troubleshooting is a method that involves collecting critical data from various interconnected devices, analyzing this data for anomaly detection and prediction, and integrating it with troubleshooting guidelines and software solutions [3]. Compared to traditional troubleshooting, which often relies on manual diagnostics and is reactive in nature, smart-troubleshooting offers a more proactive and automated approach, significantly enhancing the speed and accuracy of problem resolution [3]. Log files are central to smart-troubleshooting, as they play a crucial role in tracing the status of systems, applications, or devices and recording critical events. This data assists administrators in effectively diagnosing anomalies, including system bugs, failures, and errors [4]. There exists a multitude of anomaly detection techniques grounded in the analysis of log files. The utilization of log files for anomaly detection is often hindered by several issues, including the followings. First, the presence of sensitive information within them [5]. Due to privacy concerns and data protection regulations, many companies are reluctant to share log file data publicly [6]. Additionally, despite numerous systems generating large amounts of log data, there are very few publicly available log files labeled with error or failure information [7]. The lack of publicly available and usable datasets of log files makes new anomaly detection methodologies grounded in the analysis of log files less reliable due to a lack of proper validation. Specifically, the absence of large-scale benchmark datasets for software engineering research is a well-known problem, as highlighted in several studies [8].

In previous paper [9], the problem of generating synthetic log files was tackled using Generative Adversarial Networks (GANs). The proposed methodology utilized GANs, including VanillaGAN, CTGAN (Conditional Tabular GAN), and SeqGAN (Sequential GAN), to generate realistic synthetic logs tailored for troubleshooting in Industry 4.0. These GAN-based approaches ensured that the generated logs mirrored the statistical properties and patterns of real-world logs. Specifically, VanillaGAN served as a baseline model, CTGAN was optimized for tabular data generation, and SeqGAN targeted the sequential characteristics of log data. The study also provided a comparative analysis of these GAN architectures and demonstrated their applicability in generating high-quality synthetic logs. To foster transparency and reproducibility, a replication package was made publicly available, enabling further validation by the research community.

Building on the foundation established in previous research, this work extends the synthetic log generation methodology by introducing LogGenST, a novel approach that addresses key limitations of GAN-based methods. Traditional GAN models often rely on extensive training data or predefined templates, which limit their adaptability and scalability in diverse industrial scenarios. LogGenST overcomes these challenges by incorporating a Large Language Model (LLM)-based adversarial framework into the log generation pipeline.

LogGenST integrates three specialized LLMs into its architecture: the Generator LLM (G-LLM), which produces synthetic logs, and the Discriminator LLM (D-LLM), which evaluates the authenticity of the generated logs. This adversarial framework eliminates the need for conventional training processes or predefined templates, ensuring scalable and adaptable log generation with minimal setup. By leveraging the advancements in LLMs, LogGenST achieves high-quality log generation that goes beyond the capabilities of GAN-based systems, offering a more flexible and efficient solution for synthetic log generation in Industry 4.0.

This paper introduces a novel extension to prior work by unifying GANs and LLMs into a cohesive framework for synthetic log generation. While GANs laid a solid foundation for generating realistic logs in previous work [9], the integration of LLMs in LogGenST represents a significant advancement, enhancing both the scalability and adaptability of the log generation process. This innovative hybrid approach not only improves the quality and efficiency of generated logs but also addresses key limitations of traditional methods, marking a substantial contribution to the state-of-the-art in smart troubleshooting methodologies.

2 Preliminaries and related work

In this section, we provide background information to support the context of our paper, including foundational concepts and relevant studies. Additionally, we discuss related work to offer a comprehensive understanding of the current state of research in topics relevant to our study.

2.1 Log Files

There is a wide range of formats for different systems that generate and utilize log files. The created content depends on the specific domain that a system or application is built for. The diversity in log file types is crucial for generating synthetic log files, as it ensures that the synthetic data can accurately reflect the variety of real-world scenarios and patterns [10]. Some common types of log files include the following. *System logs* record events related to the operating system, such as startup/shutdown sequences, hardware errors, and system resource usage [11]. *Application logs* are generated by software applications, which capture information about all user interactions, containing errors, warnings, and other relevant events within the application [12]. *Access logs* record all requests made by the system or user, accessing individual files, in connection to a request from a system [13]. *Security logs* document the status of a system and significant security events. These logs assist security experts in identifying intrusions and compromises, providing valuable insights into the overall system status [14]. *Server logs* files, or sets of files, are automatically generated and maintained records of the activities performed by a server. An example of this is the web server, which keeps a history of page requests[15].

2.2 Generative Adversarial Networks

Synthetic data generation creates artificial datasets to address challenges in obtaining real data, such as privacy concerns or data scarcity just like lack of public log files. Traditional methods cannot often replicate key statistical properties of the original data. Machine Learning (ML) and Deep Learning (DL) offer innovative approaches by automatically learning patterns from existing data. ML and DL models can capture distribution, patterns, and correlations, providing statistical fidelity to the original dataset. They require fewer user-defined rules, are flexible across domains, scalable for large datasets, and aid in privacy preservation e.g. in the case of customer data. However, challenges like over-fitting and data diversity need consideration in implementing ML and DL for synthetic data generation [16]. A productive method for acquiring knowledge about generative models involves utilizing the GANs framework, proposed by Goodfellow et al. in 2014 [17]. GAN functions essentially as follows: the underlying data distribution is modeled by a generative model, represented by G. On the other hand, a discriminative model, represented by the letter D, is responsible for determining the probability that a particular sample comes from the training data rather than from G. G is trained by maximizing the likelihood that D will classify samples incorrectly. This framework can be thought of as a two-player minimax game until it reaches Nash equilibrium [17]. The loss function for a GAN consists of two components: the generator loss ($\mathcal{L}_G$) and the discriminator loss ($\mathcal{L}_D$). The loss function for the generator can be formulated as:

$$\mathcal{L}_G = -E_{z\sim p(z)}[\log D(G(z))]$$

where:

- z is a random noise vector sampled from a prior distribution $p(z)$.
- $G(z)$ represents the generated data by the generator.
- $D(G(z))$ is the discriminator's output when fed with the generated data.

The loss function for the discriminator can be formulated as:

$$\mathcal{L}_D = -E_{x\sim p_{data}(x)}[\log D(x)] - E_{z\sim p(z)}[\log(1 - D(G(z)))]$$

where:

- x represents real data samples.
- $D(x)$ is the discriminator's output when fed with real data.

The discriminator aims to maximize $\mathcal{L}_D$, while the generator aims to minimize $\mathcal{L}_G$.

Basic GAN, CTGAN [18], and SeqGAN (GRU-based) [19] were chosen for their ability to address the unique aspects of log data. Basic GANs offer a fundamental approach, making them valuable for establishing a baseline understanding. CTGAN excels in processing tabular and categorical data, making it highly suitable for structured datasets. SeqGAN, enhanced with GRU, is specifically tailored for managing sequential data. Although other variants such as

WGAN-GP or StyleGAN may provide additional advantages, we selected these models for their comprehensive capability to meet the diverse needs of log data generation, particularly their proficiency in handling both categorical and sequential data types. While the use of GANs in our project came with higher CPU and memory costs during the training phase, the investment was justified by the superior quality and realism of the synthetic data generated. The inference phase of GANs was relatively efficient, making them practical for generating large amounts of synthetic data once trained. By balancing the computational costs with the quality of synthetic data, our choice of GANs provided significant advantages, especially given the complexity and diversity of the data types we were working with. This approach ensured that we could generate realistic and high-quality synthetic data, which was crucial for achieving our research's objectives.

2.3 Large Language Models

Large Language Models (LLMs) have revolutionized the field of natural language processing (NLP) by demonstrating remarkable capabilities in tasks such as text generation, classification, summarization, and question answering. These models, often comprising billions of parameters, leverage extensive pre-training on diverse datasets to capture linguistic patterns and contextual semantics [20]. Their capability to produce coherent and contextually relevant text has facilitated applications spanning conversational agents, content creation, data synthesis, and augmentation. Recent research has investigated the use of LLMs to generate synthetic datasets for training smaller, task-specific models, enhancing data and computational efficiency in resource-constrained environments [20]. For example, iterative data synthesis approaches reduce the distribution gap between synthetic and real-world data by dynamically refining datasets based on model feedback, leading to improved performance [20].

Furthermore, approaches utilizing federated learning and differential privacy have enabled the deployment of LLMs in privacy-sensitive applications, such as mobile keyboards, while preserving user data confidentiality [21]. These advancements underline the potential of LLMs to enhance both data quality and model performance, making them integral to modern artificial intelligence pipelines. Despite their strengths, LLMs face challenges related to distribution mismatches, bias, and noise in synthetic data, which can affect model reliability [20]. Addressing these issues often involves prompt engineering, filtering mechanisms, and iterative refinement strategies to ensure high-quality data synthesis and adaptation to target domains [20]. As research progresses, novel methodologies combining synthetic data generation, error extrapolation, and privacy-preserving mechanisms continue to enhance the performance and usability of LLMs across diverse domains. These advancements not only improve data efficiency but also demonstrate the broader potential of LLMs as foundational tools for modern artificial intelligence systems [21].

2.4 Related work

The state-of-the-art methods in synthetic log file generation are either based on systematic approaches or generative models. The first method involves systematically generating synthetic log data that replicates the behavior of linear program execution logs, incorporating both normal operations and various types of anomalies to create a comprehensive dataset for testing anomaly detection algorithms. For example, a method is proposed to synthetically generate a log dataset that mimics the behavior of a linear program execution log file, including both normal and anomalous behaviors. The method involves simulating multiple program executions of a machine, with each execution consisting of a sequence of steps and possible errors [22]. However, a drawback of this approach is that synthetic data may not capture all the complexities and unpredictabilities of real-world data, potentially limiting the generalizability of research findings based on this dataset. The use of Generative models to generate synthetic log files is another innovative method that tackles the issue of insufficient labeled log data for training anomaly detection models. For instance, in a research, a generative model was used to create adversarial log files with subtle modifications designed to evade anomaly detection systems. The primary goal of employing this generative model is to demonstrate the vulnerabilities in current log parsing tools and show how adversaries can manipulate logs to avoid detection, thereby compromising the security and reliability of system log analysis [7]. Its focus is on log parsing and log structure rather than on the dataset itself.

In a research conducted by Kim et al., a Deep Neural Network (DNN) model was utilized to generate synthetic density log data for wells in the Golden field, Alberta, Canada. The primary purpose of employing this generative model is to address the data shortage problem commonly faced in reservoir modeling due to the high costs and logistical challenges of acquiring well log data through drilling [23]. Wurzenberger et al. employed a generative model to create synthetic log files by utilizing log line clustering and Markov chain simulation. This model generates Network Event Sequence (NES) data that accurately reflects actual system behavior by using a small set of real network data as input. The main purpose of this approach is to produce realistic synthetic log data for evaluating and testing security and network analysis tools offline, ensuring these tools can manage the complexities of real-world data without compromising the integrity of a live network [24]. While Markov chains are useful for certain types of synthetic data generation, especially when the system has simple and well-defined transitions, they are less powerful compared to GANs for handling complex data [25].

Regarding the use of GANs in this research area, Chen et al. propose a framework using GAN to address the imbalance problem in network threat detection datasets, particularly those with few-shot samples. The method involves generating synthetic samples through GANs to augment the small number of malicious logs, network traffic data, and other cyber threat-related data [26]. Compared to the research discussed, our work focuses on generating synthetic logs for a broader range of applications, including system monitoring and operational analysis, rather than being confined to security threats. Therefore, our approach extends beyond addressing few-shot problems and specific security scenarios.

Generally, prior work in synthetic log generation has focused on several approaches:

1. Traditional Methods: Traditional methods for log generation have relied on predefined rules and structured frameworks. Rule-based generation systems operate by defining explicit conditions and rules to create logs based on domain-specific requirements. These systems are straightforward but often lack flexibility when dealing with complex patterns or anomalies. Template-based approaches extend this concept by using predefined templates that incorporate placeholders, enabling faster log generation with minimal customization. However, they struggle to capture the diversity and variability found in real-world logs. Statistical modeling methods, such as Hidden Markov Models (HMMs) and n-grams, leverage probabilistic techniques to simulate patterns observed in log data. While these methods provide some adaptability, they are often limited by the need for large training datasets and may fail to generalize effectively in dynamic environments.
2. Machine Learning Approaches: Modern machine learning methods have significantly advanced log generation by capturing complex dependencies and patterns. GAN-based architectures use a generator-discriminator framework to produce synthetic logs that closely resemble real data, leveraging adversarial training to improve authenticity. Sequence-to-sequence models, including recurrent neural networks (RNNs) and transformers, generate logs by learning sequential dependencies, making them effective for capturing time-series patterns and contextual information. Variational autoencoders (VAEs) offer another powerful approach, encoding input data into a latent space and generating outputs by decoding sampled latent variables, thus supporting diverse and flexible log synthesis. These techniques address many limitations of traditional methods, providing higher adaptability and scalability.
3. Recent LLM Applications:
   Recent advancements in large language models (LLMs) have further revolutionized log generation by leveraging pre-trained models capable of understanding and generating human-like text. Single LLM generation approaches use powerful models such as GPT to generate logs directly, reducing the need for extensive training on task-specific data. Prompt engineering techniques enhance the performance of LLMs by designing input prompts that guide the model to generate logs with specific structures and characteristics, enabling customization for different scenarios. Fine-tuning approaches adapt pre-trained LLMs to domain-specific tasks by updating model weights with task-relevant datasets, ensuring greater accuracy and contextual relevance. These methods provide scalable and flexible solutions for generating logs that closely mimic real-world data, making them particularly suitable for modern troubleshooting and diagnostic applications. This evolution from traditional systems to advanced LLM-based approaches highlights the increasing so-

phistication of log generation methods, enabling more accurate, diverse, and scalable solutions for complex systems.

3 Methodology

In this section, we explained methodology proposed to this purpose. The sequence diagram in Figure 1 illustrates the process of log generation, verification, and refinement involving multiple components: the user, LogSynthesizer, LogVerifier, Hadoop, and Metrics. The process begins when the user initiates log generation, prompting the LogSynthesizer to fetch data from Hadoop via the LogVerifier. After receiving a response, the LogSynthesizer generates logs, which enter a feedback loop for refinement. During this loop, logs are iteratively refined based on feedback from the LogVerifier until they meet the required standards. Once the logs are finalized, they are sent to both the Metrics and Hadoop systems for evaluation and comparison with real data. Metrics processing occurs in parallel, analyzing generated logs against real data to validate performance and accuracy. The process concludes when the logs are verified, and the results are communicated back to the user, ensuring iterative improvements and high-quality log synthesis.
Overview LogGenST consists of three main components working in an adversarial setup:

3.1 Component roles

Generator LLM (G-LLM): G-LLM plays a critical role in the log generation process by handling multiple key responsibilities. It receives refined prompts from the Prompt Engineering LLM (PE-LLM) to guide the generation of synthetic log entries. G-LLM ensures that the generated logs maintain temporal and causal relationships, preserving logical sequences and dependencies within the data. Additionally, it incorporates system-specific patterns, aligning the synthetic logs with the structural and contextual characteristics of real-world data, thereby enhancing authenticity and usability.
Discriminator LLM (D-LLM): D-LLM serves as a critical evaluation component in the log generation process, ensuring the quality and authenticity of synthetic logs. Its primary functions include analyzing the generated logs by comparing them against real examples to assess their accuracy and consistency. The D-LLM provides detailed feedback on authenticity, highlighting specific issues and suggesting improvements to refine the logs further. Additionally, it identifies patterns or anomalies that may deviate from real-world data and assigns quality scores to the generated logs, enabling continuous enhancements and ensuring high standards in log generation.
Prompt Engineer LLM (PE-LLM): PE-LLM plays a pivotal role in optimizing the log generation process by facilitating effective communication between the Generator LLM (G-LLM) and the Discriminator LLM (D-LLM). Its core responsibilities include analyzing feedback provided by the D-LLM

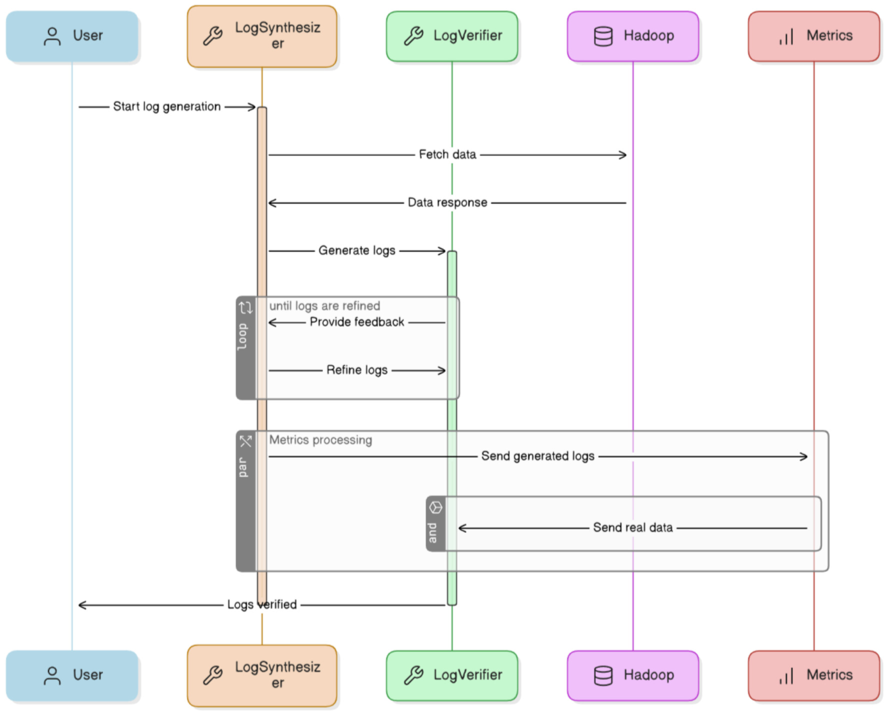

Fig. 1. Overview of proposed LogGenSt process

to identify areas for improvement in the generated logs. Based on this analysis, the PE-LLM refines prompts for the G-LLM, ensuring they are tailored to produce more accurate and authentic outputs. It maintains the quality of prompts by iteratively enhancing their structure and content, promoting consistency and relevance. Additionally, the PE-LLM drives continuous improvement, enabling adaptive refinements to achieve higher standards in log generation over successive iterations.

3.2 System architecture

Adversarial Process: The project relies on several Python libraries to implement its functionality and evaluation processes. The replicate library is used to interface with the Replicate API, enabling seamless integration with large language models (LLMs). For data visualization, matplotlib is employed to create graphical representations of performance metrics, providing clear insights into trends and improvements. Additionally, sklearn is utilized to compute key evaluation metrics, including precision, recall, and F1 score, ensuring robust performance assessment throughout the log generation and validation process.

Figure 2 demonstrates the initial setup required for a Python script to perform log generation and evaluation using machine learning techniques. It begins by importing essential libraries, including os for environment variable management, replicate for interfacing with the Replicate API to access LLMs, random for generating random numbers, and matplotlib.pyplot for data visualization. Additionally, it imports precision, recall, and F1 score metrics from sklearn.metrics, which are critical for evaluating model performance. The snippet also sets the environment variable REPLICATE with a placeholder API token, enabling secure authentication with the Replicate service. This setup ensures seamless integration with external APIs and provides the tools necessary for generating, visualizing, and assessing synthetic logs.

```
import os
import replicate
import random
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score,
recall_score, f1_score

os.environ["REPLICATE_API_TOKEN"] = "your_api_token_here"
```

Fig. 2. Key Setup Code.

4. Log Generation
   Figure 3 defines a function called call generator replicate, which interacts with the Replicate API to generate synthetic logs using a large language model (LLM). The function initializes a client connection to the Replicate service by retrieving the API token from the environment variable REPLICATE API TOKEN. It then sends a request to the specified LLM model, meta/meta-llama-3-8b-instruct, with the provided prompt as input. The response from the API is processed by concatenating all parts into a single text string. Finally, the function filters and returns non-empty lines from the generated response after stripping any whitespace. This implementation effectively utilizes the Replicate API for dynamic log generation, enabling seamless integration with advanced language models to produce structured and contextually relevant synthetic logs.
5. Log Validation
   Figure 4 defines a function, analyze logs with discriminator, designed to validate and analyze generated logs using a discriminator model. It constructs a detailed prompt by combining a predefined DISCRIMINATOR PROMPT with the provided logs, ensuring context for analysis. The function initializes

```
def call_generator_replicate(prompt):
api = replicate.Client(api_token=os.environ
["REPLICATE_API_TOKEN"])
response = api.run(
"meta/meta-llama-3-8b-instruct",
input={"prompt": prompt}
)
response_text = "".join(response)
return [line.strip() for line in
response_text.split('\n') if line.strip()]
```

Fig. 3. Log Generation Code.

a connection to the Replicate API using an API token stored in the environment variable REPLICATE API TOKEN. It then sends the constructed prompt to the meta/meta-llama-3-8b-instruct model for processing. The response, which contains the analysis results, is returned as a concatenated string. This approach leverages the discriminator's capabilities to evaluate log authenticity, highlight anomalies, and provide feedback for improving log quality.

```
def analyze_logs_with_discriminator(generated_logs):
prompt = f"{DISCRIMINATOR_PROMPT}\n\nAnalyze this
log:\n" + "\n".join(generated_logs)
api = replicate.Client(api_token=os.environ
["REPLICATE_API_TOKEN"])
response = api.run(
"meta/meta-llama-3-8b-instruct",
input={"prompt": prompt}
)
return "".join(response)
```

Fig. 4. Log Validation Code.

4 Experimental Setup

We chose two datasets of real log files from various sources to serve as the training data for our GAN models. These log files include a variety of faults, errors, and failure patterns typical of industrial applications. The dataset was preprocessed to standardize the log formats. Then, three GAN models were selected for this study and tailored for log file generation to meet our specific requirements. These models were then trained using adversarial methods, where the generator and discriminator networks were iteratively refined to achieve the desired quality of synthetic log files. This training process ensured that the generated logs closely mimicked the characteristics and patterns of the real log data. In the next step, we evaluated our results using these metrics: Generator Loss, Discriminator Loss, F1 score, precision, recall and Cumulative Sum (Cumsum) Plots. Generator Loss assessed the quality of the synthetic log entries, while Discriminator Loss evaluated the accuracy of distinguishing real from synthetic logs, with balanced discriminator losses being desirable. Cumsum Plots provided a visual comparison of the distributions of real and synthetic log data, helping to identify any discrepancies.

4.1 Dataset

We used publicly available dataset as input to our GANs. The logs include data from various systems such as Hadoop Distributed File System (HDFS)[4] and ZooKeeper[5]. Apache Spark

4.2 Architecture of proposed GANs

In this subsection, we will define the architecture of our customized GAN models.

Vanilla GAN: First, we customize the GAN for our dataset as it is shown in Table 1. Both the Discriminator and Generator models utilize a series of dense layers with ReLU activation for the intermediate layers to ensure non-linearity. For the output layers, we use specialized activation functions (Tanh for the Generator and Sigmoid for the Discriminator)to suit their specific tasks within the GAN framework.

Table 1. Vanilla GAN Generator, Discriminator Model Architecture

	Layer Type	Units	Input Shape	Activation
Generator	Dense	128	(input_dim,)	relu
	Dense	64	(128,)	relu
	Dense	32	(64,)	relu
	Dense	output_dim	(32,)	relu
Discriminator	Dense	128	(input_dim,)	relu
	Dense	64	(128,)	relu
	Dense	32	(64,)	relu
	Dense	1	(32,)	sigmoid

[4] https://github.com/logpai/loghub/tree/master/Hadoop

[5] https://github.com/logpai/loghub

CTGAN: The table 2 describes the CTGAN architecture. The Generator starts with an input layer for the latent vector, followed by dense layers with 256 and 128 units using ReLU activation, interspersed with batch normalization and dropout layers. The final layer uses Tanh activation to produce the output. The Discriminator begins with an input layer, followed by dense layers with 256 and 128 units using ReLU activation, and dropout layers to prevent overfitting. The final layer uses Sigmoid activation to output a probability score, distinguishing real from generated data.

Table 2. CTGAN Generator, Discriminator Model Architecture

	Layer Type	Units	Output Shape	Activation
Generator	Input	-	(latent_dim,)	-
	Dense	256	(256,)	relu
	BatchNormalization	-	(256,)	-
	Dropout	-	(256,)	-
	Dense	128	(128,)	relu
	BatchNormalization	-	(128,)	-
	Dropout	-	(128,)	-
	Dense	output_dim	(output_dim,)	tanh
Discriminator	Input	-	(input_dim,)	-
	Dense	256	(256,)	relu
	Dropout	-	(256,)	-
	Dense	128	(128,)	relu
	Dropout	-	(128,)	-
	Dense	1	(1,)	sigmoid

SeqGAN: Table 3 describes the SeqGAN model architecture for both the Generator and Discriminator. The Generator starts with an input layer followed by GRU layers with units decreasing from 256 to 32, each with batch normalization, LeakyReLU activation, and dropout layers. The final layer uses Tanh activation for output. The Discriminator also begins with an input layer, followed by GRU layers with units decreasing from 256 to 32, with LeakyReLU activation and dropout layers. The final dense layer uses Sigmoid activation to output a probability score. This architecture uses GRU layers for sequential data, with activation and regularization techniques to improve performance and robustness.

The a priori hypothesis is that CTGAN would perform well due to its specific design for handling tabular and categorical data, which aligns well with log data characteristics. SeqGAN (GRU-based) was expected to capture the sequential dependencies effectively. These approaches were selected based on their compatibility with the nature of log data, aiming to leverage their strengths in modeling the unique properties of log datasets.

Table 3. SeqGAN Generator, Discriminator Model Architecture

Layer Type	Units	Output Shape	Activation
Input	-	(input_dim, 1)	-
GRU	256	(input_dim, 256)	-
BatchNormalization	-	(input_dim, 256)	-
LeakyReLU	-	(input_dim, 256)	-
Dropout	-	(input_dim, 256)	-
GRU	128	(input_dim, 128)	-
BatchNormalization	-	(input_dim, 128)	-
LeakyReLU	-	(input_dim, 128)	-
Dropout	-	(input_dim, 128)	-
GRU	64	(input_dim, 64)	-
BatchNormalization	-	(input_dim, 64)	-
LeakyReLU	-	(input_dim, 64)	-
Dropout	-	(input_dim, 64)	-
GRU	32	(32)	-
BatchNormalization	-	(32)	-
LeakyReLU	-	(32)	-
Dropout	-	(32)	-
Dense(Generator*)	output_dim	(output_dim)	tanh
Dense(Discriminator*)	1	(1)	sigmoid

5 Results

This section highlights the performance trends observed during the iterative rounds of log generation and classification. The analysis distinguishes between real logs, sourced from authentic Hadoop datasets, and synthetic logs, synthesized by G-LLM and classified using D-LLM. Figure 5 illustrates the performance metrics—Precision, Recall, and F1 Score—evaluated over five rounds of log generation and refinement. The Recall metric consistently achieves the highest values, peaking at approximately 0.95 in the second round before slightly declining but remaining above 0.91. The F1 Score follows a similar trend, reaching its maximum around 0.91 in the second round and then fluctuating slightly in subsequent rounds. Precision, while starting lower, peaks near 0.88 in the second round and experiences moderate variations across the remaining rounds. These trends highlight the system's ability to refine logs iteratively, achieving significant improvements early in the process and stabilizing performance over time. The variations also emphasize the need for continued optimization to maintain consistency and maximize overall performance across all metrics.

Figure 6 visualizes the performance metrics—Precision, Recall, and F1 Score—across five rounds of log generation and refinement. Each cell displays the numerical value of the respective metric, with colors representing the score intensity based on the accompanying color bar. Darker shades indicate higher scores, while lighter shades correspond to lower values. Notably, Recall consistently achieves higher scores compared to Precision and F1 Score, peaking at 0.950 in Round 2 and maintaining high values throughout the iterations. F1 Scores also follow a similar trend, reflecting improvements in balancing precision and recall, reach-

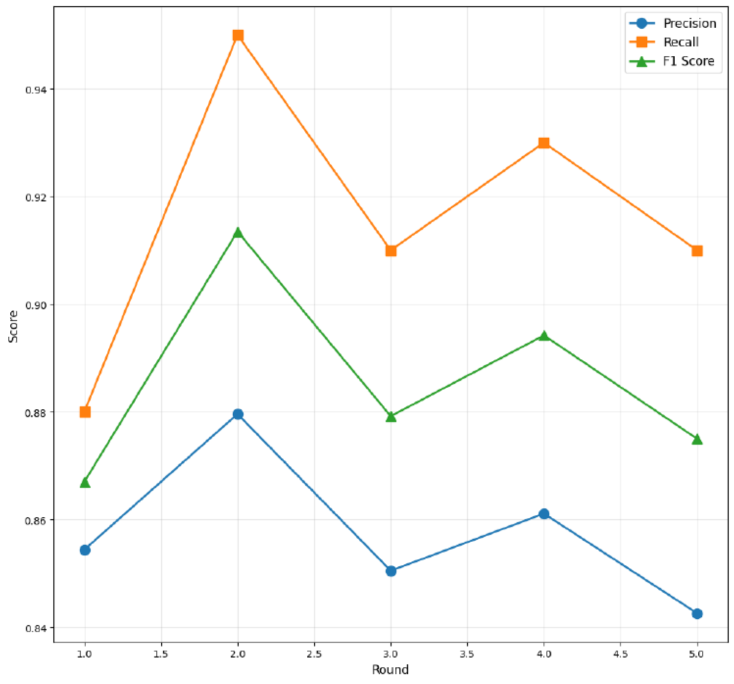

Fig. 5. Classification Metrics Over Time.

ing 0.913 in Round 2 and stabilizing in subsequent rounds. This visualization highlights the iterative refinement process, demonstrating enhanced log quality early on, followed by slight fluctuations, which emphasize the need for further optimization to maintain consistency across rounds.

5.1 Evaluation Metrics

Each round of log generation and validation evaluates system performance by producing predictions and calculating key metrics. precision measures the proportion of correctly identified real logs out of all predicted real logs, while recall assesses the proportion of correctly identified real logs relative to all actual real logs. To balance these measures, the F1 Score, which represents the harmonic mean of precision and recall, is also computed. These metrics are tracked and visualized across multiple rounds, enabling the monitoring of performance trends and improvements as the system undergoes iterative refinements. The visualizations provide insights into the model's ability to generate and classify logs more accurately over time, showcasing its adaptability and effectiveness in handling log authenticity tasks.

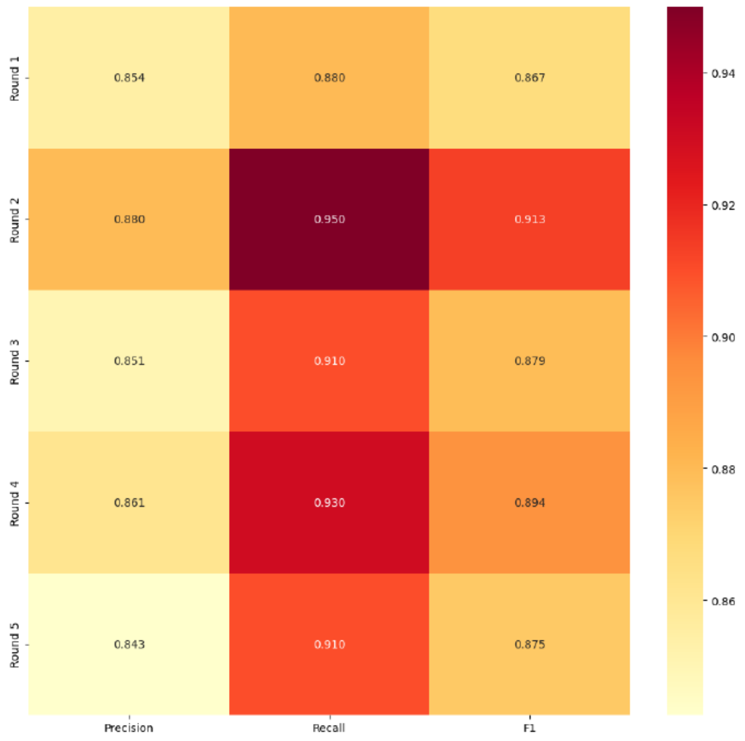

Fig. 6. Metrics Heatmap by Round.

6 Discussion

The proposed framework for synthetic log generation and validation demonstrates several advantages, yet it also presents challenges and limitations that need to be addressed for broader applicability.

One of the key strengths of the framework is Automated Prompt Refinement, which leverages iterative feedback loops to dynamically adjust prompts based on discriminator evaluations. This automation reduces manual intervention, ensuring that logs are continuously refined and improved. Additionally, the framework supports Continuous Quality Improvement by integrating evaluation metrics, enabling it to adapt and evolve over multiple iterations to produce more accurate and realistic logs. Another important feature is Explainable Changes, which provide transparency by highlighting specific refinements made during the process. This ensures traceability and improves trust in the generated outputs. Furthermore, the framework offers Flexible Adaptation, allowing it to accommodate diverse log formats, domains, and system requirements. Its ability to modify prompts and integrate domain-specific knowledge makes it versatile for different use cases.

Despite these strengths, the framework faces several challenges. Prompt Convergence can be a complex task, as ensuring that prompts converge to optimal configurations may require multiple iterations and fine-tuning. Maintaining Quality Consistency across logs is another challenge, as variations in inputs or prompts can lead to fluctuations in output quality. Performance Optimization remains a critical area, particularly for reducing processing time and computational overhead while sustaining high performance. Additionally, Cost Management must be carefully handled, as frequent API calls and evaluations can result in increased operational expenses, making scalability a concern for larger deployments.

Alongside challenges, the framework has inherent Limitations that need to be considered. API Rate Limits imposed by service providers may restrict the number of requests that can be processed within a given timeframe, impacting scalability. Response Time Latency can also be a bottleneck, especially when dealing with large datasets or complex prompts, potentially delaying the refinement process. Moreover, Context Window Constraints in LLMs limit the amount of data that can be processed in a single prompt, posing difficulties in handling logs with extensive contextual dependencies or multi-step sequences.

While the framework offers significant advantages through automation, adaptability, and continuous improvement, it also faces challenges related to performance and cost optimization. Furthermore, limitations like API constraints and latency highlight areas for future enhancement. Addressing these challenges and limitations will be essential to ensure broader adoption and scalability of the framework in real-world applications.

7 Conclusions and Future Work

In our previous work, we addressed the challenge of public log data scarcity from heterogeneous interconnected devices by leveraging GANs to synthesize realistic log files. This approach was particularly valuable for generating log data tailored for smart-troubleshooting, ensuring the synthetic logs captured faults, errors, and failures in patterns resembling real-world scenarios. Specifically, we employed Vanilla GAN, CTGAN, and SeqGAN to generate synthetic logs, with CTGAN demonstrating the highest performance in producing high-quality data. This work laid a solid foundation for enhancing the reliability and efficiency of synthetic log generation, paving the way for scalable and effective diagnostics in industrial applications. Despite these promising results, challenges such as improving data accuracy, ensuring fairness, and addressing privacy concerns remained key areas for further development.

Building upon this foundation, we developed LogGenST, an advanced and adaptable framework for synthetic log generation that integrates LLMs to overcome some of these challenges. LogGenST employs a novel approach by combining generator and discriminator LLMs with iterative prompt engineering, achieving high authenticity without requiring extensive model training. This method significantly reduces the computational overhead while maintaining the quality and contextual relevance of the generated logs. Through this approach, LogGenST provides a flexible and scalable solution for generating logs across various domains, facilitating improvements in software testing and development processes.

The iterative prompt engineering used in LogGenST enables dynamic refinement of synthetic logs, allowing the system to adapt to different error scenarios and data formats. By leveraging LLMs, LogGenST captures complex patterns and anomalies in logs, closely mimicking real-world data distributions. This adaptability makes it particularly suitable for scenarios where data diversity and authenticity are critical, such as testing distributed systems and debugging software pipelines.

Future work will focus on expanding support for multi-system log formats, incorporating domain-specific knowledge for error simulation, and automating feedback analysis through advanced prompt engineering techniques. Further enhancements will address scalability, privacy, and fairness concerns, ensuring the generated logs maintain high quality and relevance across diverse scenarios. By exploring reinforcement learning and real-time log generation, LogGenST aims to become a more versatile tool for modern log generation and troubleshooting challenges.

References

1. P. Singh, M. Saman Azari, F. Vitale, F. Flammini, N. Mazzocca, M. Caporuscio, and J. Thornadtsson, "Using log analytics and process mining to enable self-healing in the internet of things," *Environment Systems and Decisions*, vol. 42, no. 2, pp. 234–250, 2022.
2. M. Caporuscio, F. Flammini, N. Khakpour, P. Singh, and J. Thornadtsson, "Smart-troubleshooting connected devices: Concept, challenges and opportunities," *Future Generation Computer Systems*, vol. 111, pp. 681–697, 2020.
3. S. Partovian, A. Bucaioni, F. Flammini, and J. Thornadtsson, "Analysis of log files to enable smart-troubleshooting in industry 4.0: a systematic mapping study," *IEEE Access*, pp. 1–1, 2023.
4. B. Xia, Y. Bai, J. Yin, Y. Li, and J. Xu, "LogGAN: a log-level generative adversarial network for anomaly detection using permutation event modeling," *Information Systems Frontiers*, vol. 23, pp. 285–298, 2021.
5. L. Baruh, E. Secinti, and Z. Cemalcilar, "Online Privacy Concerns and Privacy Management: A Meta-Analytical Review," *Journal of Communication*, vol. 67, pp. 26–53, 01 2017.
6. M. Zahid, A. Bucaioni, and F. Flammini, "Model-based trustworthiness evaluation of autonomous cyber-physical production systems: A systematic mapping study," *ACM Comput. Surv.*, vol. 56, feb 2024.
7. J. Sun, B. Liu, and Y. Hong, "Logbug: Generating adversarial system logs in real time," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, (New York, NY, USA), p. 2229–2232, Association for Computing Machinery, 2020.
8. F. Bozyigit, T. Bardakci, A. Khalilipour, M. Challenger, G. Ramackers, Ö. Babur, and M. R. Chaudron, "Generating domain models from natural language text using

nlp: a benchmark dataset and experimental comparison of tools," *Software and Systems Modeling*, pp. 1–19, 2024.

9. S. Partovian, F. Flammini, and A. Bucaioni, "Leveraging gans to generate synthetic log files for smart-troubleshooting in industry 4.0," in *2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 1–8, 2024.
10. S. Kim, K. H. Kim, B. Min, J. Lim, and K. Lee, "Generation of synthetic density log data using deep learning algorithm at the golden field in alberta, canada," *Geofluids*, vol. 2020, 2020.
11. A. Tomono, M. Uehara, and Y. Shimada, "Improvement and evaluation of a method to manage multiple types of logs," in *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, pp. 601–606, 2011.
12. B. Stuike and Y. Amannejad, "Pairwise application log classification," in *2023 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pp. 349–350, 2023.
13. L. Karimi, M. Aldairi, J. Joshi, and M. Abdelhakim, "An automatic attribute-based access control policy extraction from access logs," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2304–2317, 2022.
14. A. Xu, L. Chen, Y. Jiang, H. Lv, H. Yang, and B. Li, "Finding gold in the sand: Identifying anomaly indicators though huge amount security logs," in *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pp. 140–144, 2020.
15. P. Sharma, S. Yadav, and B. Bohra, "A review study of server log formats for efficient web mining," in *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pp. 1373–1377, 2015.
16. D. Garcia Torres, *Generation of Synthetic Data with Generative Adversarial Networks.* PhD thesis, KTH ROYAL INSTITUTE OF TECHNOLOGY, 10 2018.
17. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.
18. L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, *Modeling tabular data using conditional GAN*, p. 7335–7345. Red Hook, NY, USA: Curran Associates Inc., 2019.
19. L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: sequence generative adversarial nets with policy gradient," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, p. 2852–2858, AAAI Press, 2017.
20. R. Wang, W. Zhou, and M. Sachan, "Let's synthesize step by step: Iterative dataset synthesis with large language models by extrapolating errors from small models," in *Findings of the Association for Computational Linguistics: EMNLP 2023* (H. Bouamor, J. Pino, and K. Bali, eds.), (Singapore), pp. 11817–11831, Association for Computational Linguistics, Dec. 2023.
21. S. Wu, Z. Xu, Y. Zhang, Y. Zhang, and D. Ramage, "Prompt Public Large Language Models to Synthesize Data for Private On-device Applications," *arXiv e-prints*, p. arXiv:2404.04360, Apr. 2024.
22. S. Luftensteiner and P. Praher, "A synthetic dataset for anomaly detection of machine behavior," in *Database and Expert Systems Applications - DEXA 2022 Workshops* (G. Kotsis, A. M. Tjoa, I. Khalil, B. Moser, A. Taudes, A. Mashkoor, J. Sametinger, J. Martinez-Gil, F. Sobieczky, L. Fischer, R. Ramler, M. Khan, and G. Czech, eds.), (Cham), pp. 424–431, Springer International Publishing, 2022.

23. S. Kim, K. H. Kim, B. Min, J. Lim, and K. Lee, “Generation of synthetic density log data using deep learning algorithm at the golden field in alberta, canada,” *Geofluids*, vol. 2020, pp. Article ID 5387183, 26 pages, 2020.
24. M. Wurzenberger, F. Skopik, G. Settanni, and W. Scherrer, “Complex log file synthesis for rapid sandbox-benchmarking of security- and computer network analysis tools,” *Information Systems*, vol. 60, pp. 13–33, 2016.
25. A. Figueira and B. Vaz, “Survey on synthetic data generation, evaluation methods and gans,” *Mathematics*, vol. 10, no. 15, 2022.
26. L. Chen, Y. Song, and J. Chen, “A framework for few-shot network threats based on generative adversarial networks,” in *2023 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, 2023.