Mohamed Basuony

Survey of Log-Based Anomaly Detection

From Classical ML to LLMs (Supervised by: Sadegh Keshtkar)

When Logs Save the Day

```
RuntimeError: CUDA out of memory. Tried to allocate 200.00 MiB (GPU 0; 15.78 GiB total
capacity; 14.56 GiB already allocated; 38.44 MiB free; 14.80 GiB reserved in total by
PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid
fragmentation. See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```

RuntimeError: CUDA out of memory. The kind of error you only see. . . in logs.

Agenda

What is Log-Based Anomaly Detection?

- Detects unexpected patterns in system logs
- Uses parsing, embeddings, or sequence modeling
- Helps catch software failures, intrusions, config errors
- Essential for observability in complex systems

Key Challenges

- Logs are noisy, high-volume, and unstructured
- Labels for anomalies are rare or missing
- Logs evolve due to system upgrades
- Sequence + semantic context matters

Taxonomy of Methods

- **Traditional ML:** PCA, Isolation Forest, OC-SVM
- **RNN-Based DL:** DeepLog, OC4Seq, LogRobust
- **Transformer-Based:** LogAnomaly, LogBERT, UniLog, LogFormer
- **LLM-Based:** LogGPT, LogPrompt, LogLLaMA, HuntGPT

Traditional ML Models

- **PCA (2009):** Linear subspace projection
- **Isolation Forest:** Randomly isolates outliers
- **OC-SVM:** One-class kernel decision boundary

Pros: Fast and interpretable

Cons: No sequence context, low F1

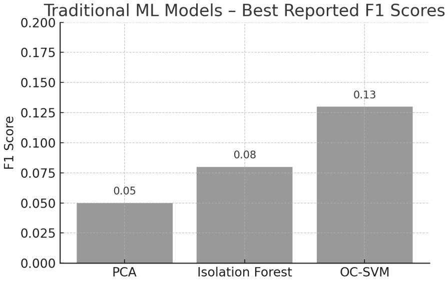
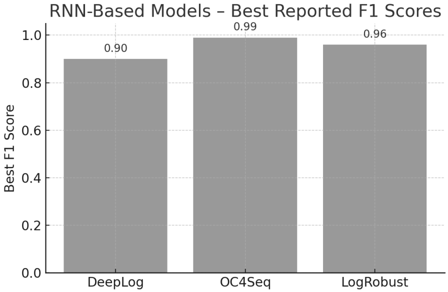


Illustration of model taxonomy used in this survey.

RNN-Based Models Overview

- **DeepLog (2017):** LSTM predicts next log key
- **OC4Seq (2021):** Multi-scale GRU with one-class loss
- **LogRobust (2019):** Bi-LSTM with attention + TF-IDF vectors

RNNs model sequence, but struggle with unseen logs.



OC4Seq – Multi-Scale One-Class GRU

- **Objective:** Detect anomalies in discrete event sequences without any labeled anomalies.
- **Architecture:**
  - Uses two GRU modules: Global and Local.

OC4Seq – Multi-Scale One-Class GRU

- **Objective:** Detect anomalies in discrete event sequences without any labeled anomalies.
- **Architecture:**
  - Uses two GRU modules: Global and Local.
- **Anomaly Scoring:**
  - Learns a compact hypersphere in latent space.
  - Measures how far a sequence deviates from learned normal embedding.

OC4Seq – Multi-Scale One-Class GRU

- **Objective:** Detect anomalies in discrete event sequences without any labeled anomalies.
- **Architecture:**
  - Uses two GRU modules: Global and Local.
- **Anomaly Scoring:**
  - Learns a compact hypersphere in latent space.
  - Measures how far a sequence deviates from learned normal embedding.
- **Loss Function:**
  - Inspired by Deep SVDD — minimizes distance to center point.
  - Combines global and local losses for multi-scale learning.

Transformer-Based Models – Overview

- Transformers use **self-attention** to capture long-range dependencies in log sequences.
- Unlike RNNs, they model all positions in parallel — ideal for complex, long, or noisy logs.
- Most models use pretrained language modeling (e.g., BERT-style) on logs, then fine-tune for detection.

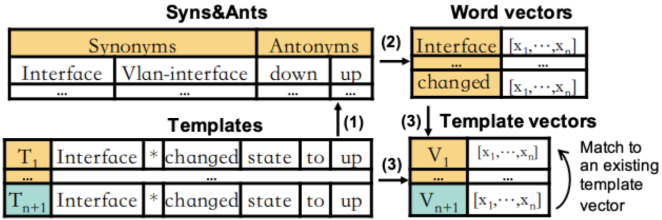
Transformer-Based Models – Overview

- Transformers use **self-attention** to capture long-range dependencies in log sequences.
- Unlike RNNs, they model all positions in parallel — ideal for complex, long, or noisy logs.
- Most models use pretrained language modeling (e.g., BERT-style) on logs, then fine-tune for detection.

Model	**Training Type**	**Highlights**
LogAnomaly	Supervised	Template2Vec + LSTM hybrid
LogBERT	Self-supervised	BERT masking on log keys
LogFormer	Adapter-tuned	Log-attention + efficient tuning
UniLog	Unified multitask	AD, prediction, summarization

LogAnomaly (2019)

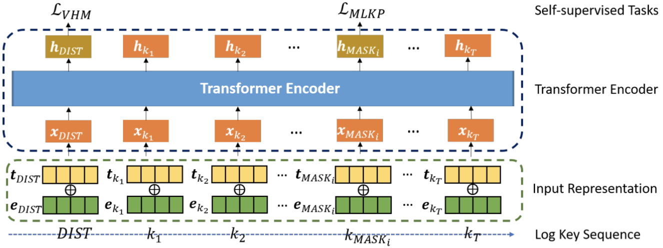
- **Architecture:** LSTM model with **Template2Vec** embeddings as input.
- **Dual Prediction:**
  - Predicts the next log template (sequence anomaly).
  - Predicts the expected frequency of log types (quantitative anomaly).
- **Detection Rule:** A sequence is flagged anomalous if either prediction deviates from expected behavior.



Template2Vec encodes log templates into dense vectors for sequence modeling.

LogBERT (2021)

- **Architecture:** Transformer model trained using **masked log key prediction** (BERT-style).
- **Training:** Self-supervised on normal logs, no need for labeled anomalies.
- **Anomaly Detection:**
  CLS token summarizes the sequence.
  - **Deep SVDD loss** forces normal embeddings into a compact hypersphere.



UniLog (2021)

- **Goal:** Provide a unified Transformer-based model for multiple log analysis tasks.
- **Tasks Handled:**
  - Anomaly detection (unsupervised)
  - Failure prediction (supervised)
  - Log summarization (sequence-to-sequence)
  - Log compression (semantic entropy modeling)
- **Architecture:**
  - Shared pretrained encoder with task-specific heads.
  - BERT-style masked modeling during pretraining.

LogFormer (2024)

- **Architecture:** Transformer encoder with **parallel adapter layers** for efficient fine-tuning.
- **Key Feature – Log-Attention:**
  - Injects structured information from parsed logs into attention scores.
  - Retains token-level semantics lost in traditional parsing.
- **Training Strategy:**
  - Pretrained on source domain logs.
  - Tuned on new domains by updating only adapter layers (5

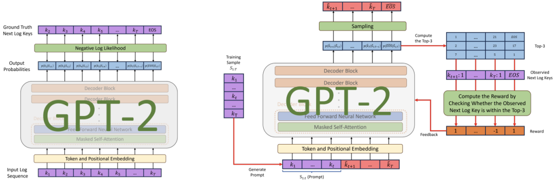
LLM-Based Detection – Overview

- Foundation models like GPT and LLaMA are now applied to log anomaly detection.
- These models are typically adapted using:
  - Fine-tuning (e.g., GPT-3, LogLLaMA)
  - Prompt engineering (e.g., LogPrompt, ChatGPT)
  - Reinforcement learning (e.g., LogGPT, LogLLaMA)

Model	Tuning Type	Highlights
LogGPT	RL fine-tuning	GPT-2 + Top-K reward
LogLLaMA	RL fine-tuning	LLaMA-2 + REINFORCE
LogPrompt	Prompting	ChatGPT + Chain-of-Thought
HuntGPT	Prompting	GPT + SHAP/LIME explanations

LogGPT (2023)

- **Architecture:** GPT-2 autoregressive model fine-tuned using reinforcement learning.
- **Training Objective:**
  - Maximize Top-K inclusion of the true next log key.
  - Rewards correct predictions via REINFORCE algorithm.
- **Detection Strategy:**
  - A log sequence is flagged as anomalous if the true next log key is outside the predicted Top-K.



(a) Pre-training

(b) Fine-tuning

Fig. 2: Framework of LogGPT.

LogLLaMA (2024)

- **Base Model:** LLaMA-2 fine-tuned for anomaly detection on normal logs only.
- **Anomaly Detection:**
  - Uses Top-K REINFORCE objective, similar to LogGPT.
  - Token-level prediction with binary decision threshold.
- **Training Strategy:**
  - Retains LLaMA backbone; only reward shaping is learned.
  - Fully unsupervised — trained only on normal logs.

A. Overview of our Framework

Fig. 1: Module 1: Log message preprocessing

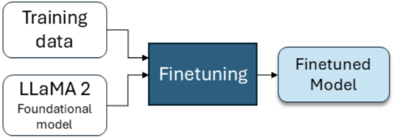


Fig. 2: Module 2: Model finetuning

Illustration of LogLLaMA architecture (GPT-style RL over LLaMA-2).

LogPrompt (2024)

- **Approach:** Uses ChatGPT with zero-shot and Chain-of-Thought (CoT) prompting.
- **Prompt Strategies:**
  - Few-shot examples.
  - Justifications + rules.
  - Context summarization.
- **Output:**
  - Human-readable explanations per anomaly.
  - Label + justification.
- **Tradeoff:** Most interpretable output, but lower F1 (0.38–0.45).

HuntGPT and Logsy

- **HuntGPT (2023):**
  - Combines a Random Forest anomaly detector with SHAP and LIME.
  - GPT-3.5 explains model decisions in a dashboard chatbot.
  - CISM-certified and readable (grade level: college).
  - F1 Score: 0.825

HuntGPT and Logsy

- **HuntGPT (2023):**
  - Combines a Random Forest anomaly detector with SHAP and LIME.
  - GPT-3.5 explains model decisions in a dashboard chatbot.
  - CISM-certified and readable (grade level: college).
  - F1 Score: 0.825
- **Logsy (2020):**
  - BERT encoder + attention + spherical loss.
  - Trained only on normal logs using self-supervised objectives.
  - Embeddings used directly for anomaly classification.
  - F1 Score: 0.86

What Are Log Generators and Why Are They Important?

- **Problem:** Real-world log datasets are limited in coverage, often lack labels, and are expensive to collect.
- **Solution: Log generators** automatically produce synthetic log sequences — simulating system behavior at scale.

What Are Log Generators and Why Are They Important?

- **Problem:** Real-world log datasets are limited in coverage, often lack labels, and are expensive to collect.
- **Solution: Log generators** automatically produce synthetic log sequences — simulating system behavior at scale.
- **Benefits:**
  - Create training data without requiring real system crashes.
  - Provide control over log coverage and anomaly types.
  - Help evaluate models on rare or future edge cases.
- **Approaches:**
  - Static program analysis (e.g., AutoLog)
  - LLM-based semantic simulation (e.g., AnomalyGen)

AutoLog (2023)

- **Purpose:** Automatically generate log sequences from code using static analysis.
- **Method:**
  - Builds Control Flow Graphs (CFGs) from source code.
  - Extracts log-related call paths without executing the program.

AutoLog (2023)

- **Purpose:** Automatically generate log sequences from code using static analysis.
- **Method:**
  - Builds Control Flow Graphs (CFGs) from source code.
  - Extracts log-related call paths without executing the program.
- **Benefits:**
  - Covers log paths even without runtime data.
  - Scales to large codebases.
- **Limitations:**
  - Misses dynamic behaviors (e.g., exceptions).
  - Does not annotate anomalies.

AnomalyGen (2024)

- **Purpose:** Generate realistic and annotated log sequences using LLMs.
- **Pipeline:**
  - Extracts call graphs and CFGs from code.
  - Uses LLM + Chain-of-Thought reasoning to simulate log flows.
  - Annotates both explicit (e.g., "ERROR") and implicit (semantic) anomalies.

AnomalyGen (2024)

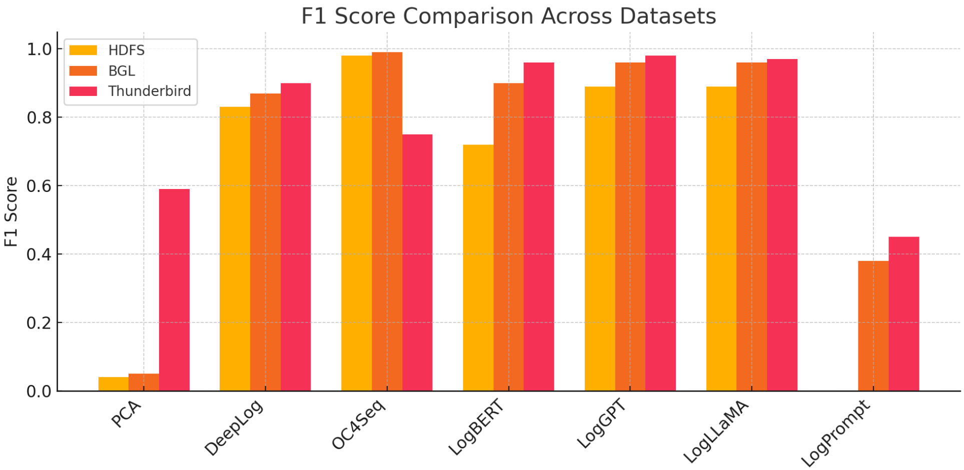
- **Purpose:** Generate realistic and annotated log sequences using LLMs.
- **Pipeline:**
  - Extracts call graphs and CFGs from code.
  - Uses LLM + Chain-of-Thought reasoning to simulate log flows.
  - Annotates both explicit (e.g., "ERROR") and implicit (semantic) anomalies.
- **Impact:**
  - Achieves **97.5% log event coverage**.
  - Improves downstream F1 scores by up to **3.7%**.
- **Strength:** Combines program structure with LLM semantics to generate high-quality training data.

AutoLog vs AnomalyGen

Feature	**AutoLog (2023)**	**AnomalyGen (2024)**
Log generation method	Static CFG analysis	CFG + LLM + CoT reasoning
Dynamic behavior support	✗	✓
Anomaly annotation	✗	✓
Needs runtime execution	✗	✗
Event coverage	✓	✓
Improves model performance	✗	✓
LLM involvement	✗	✓

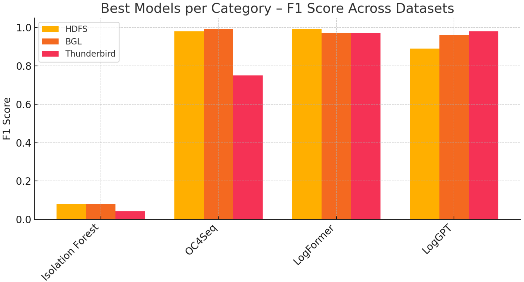
AnomalyGen significantly extends AutoLog by adding semantic reasoning and labeled output.

F1 Score Comparison Across Datasets



Which Model Wins per Category?

- **Traditional ML:** Isolation Forest
- **RNN-Based:** OC4Seq
- **Transformer-Based:** LogFormer
- **LLM-Based:** LogGPT



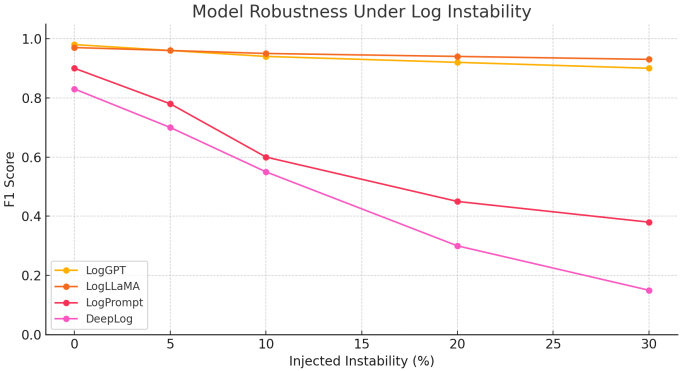
Capability Matrix

Model	OOV Support	Online?	Interpretable?	Uses RL?
PCA	✗	✗	✓	✗
DeepLog	✗	✗	✗	✗
OC4Seq	✗	✗	✗	✗
LogBERT	✓	✗	✗	✗
LogFormer	✓	✓	●	✗
LogGPT	✓	✓	●	✓
LogLLaMA	✓	✓	●	✓
LogPrompt	✓	✓	✓	✗
HuntGPT	✓	✗	✓	✗

✓ = *Supported* ✗ = *Not supported* ● = *Partial/indirect support*

Model Robustness on Unstable Logs

- **Real-world logs evolve:** new templates, dropped keys, noisy sequences.
- **LogGPT and LogLLaMA** maintain high F1 due to reinforcement learning.
- **LogPrompt and DeepLog** suffer major drops under instability.



Takeaways and Emerging Trends

- Log anomaly detection is shifting from **pattern matching** to **semantic modeling**.
- Transformer models (e.g., **LogFormer**) capture long-range structure effectively.
- **Prompting** (LogPrompt) enables quick deployment but trails fine-tuned models in accuracy.
- **Reinforcement learning** (LogGPT, LogLLaMA) improves robustness to unstable logs.
- Tools like **AnomalyGen** show that LLMs can help create data — not just analyze it.

What's next?

- **Token limits** make long log sequences hard to process.
- **High inference cost** prevents real-time LLM deployment.
- **Interpretability remains limited**, especially for autoregressive models.
- **No unified benchmarks** exist for LLM-based log anomaly detection.
- **Underexplored: Google's T5 model**
  - Uses a full **encoder–decoder** architecture
  - Reframes all tasks as `text` → `text`
  - Could generate natural-language justifications for anomalies — not just labels.

References

- DeepLog – ACM CCS 2017
- OC4Seq – WSDM 2021
- LogAnomaly – IJCAI 2019
- LogRobust – FSE 2019
- PCA – Classical Baseline
- Isolation Forest – Classical Baseline
- OC-SVM – Classical Baseline
- LogBERT – arXiv:2103.04475
- UniLog – arXiv:2112.03159
- LogFormer – AAAI 2024
- LogGPT – arXiv:2309.14482
- LogPrompt – arXiv:2308.07610
- HuntGPT – arXiv:2309.16021
- Logsy – arXiv:2008.09340
- Deep SVDD – ICML 2018
- AutoLog – ASE 2023
- AnomalyGen – arXiv:2504.12250
- LogHub – arXiv:2008.06448
- Drain – Log Parsing Baseline
- LogCluster – Traditional Baseline
- Invariant Mining – Traditional Baseline
- BGL / HDFS / Thunderbird – Benchmark Datasets
- T5 – Raffel et al., JMLR 2020