Case Study: AI-Driven Log Extraction and Trace Outlier Detection for Efficient Post-Silicon Validation

Kowshic A. Akash*, Tobias Wulf*, Torsten Valentin*, Alexander Geist*, Ulf Kulau†, John Jose‡ and Sohan Lal†
*NXP Semiconductors, Hamburg, Germany
†Hamburg University of Technology, Germany
‡Indian Institute of Technology Guwahati, India

***Abstract*—In the *post-silicon validation* process, various functionalities and boundaries of a system-on-chip (SoC) are tested, generating a large amount of data in the form of log files, trace data, and oscilloscope images. Log files provide essential information regarding a test run, such as test setup, while trace files offer insights into internal register statuses and sweep parameters like voltage, frequency, and temperature. Manually analyzing and debugging these files is time-consuming, inefficient, costly, and prone to errors. To address these challenges, we propose an AI-powered approach to automatically extract critical log messages from extensive datasets, generating concise log files with only the most pivotal information. Our method utilizes a multi-class Long Short Term Memory (LSTM) neural network. Our primary focus is to *minimize false negatives* (high recall) to ensure that critical anomalies are not overlooked, thus delivering more reliable SoCs. Simultaneously, we aim to *minimize false positives* (high precision) to reduce manual debugging efforts. Our proposed method achieves high recall/precision of 94%/99% for normal, 99%/99% for information, 92%/64% for error, and 98%/88% for warning log categories. Additionally, for outlier detection in trace data, we propose an unsupervised method based on Isolation Forest, which achieves high recall/precision of 95%/100% and 92%/73% for anomalous data points across two distinct datasets, and nearly 100% for normal data points.**

***Index Terms*—Post-silicon validation, Anomaly detection, ML**

I. Introduction

With the rise of digital technology and handheld devices, the demand for diverse SoC architectures has surged. The growth of LLMs and generative AI further accelerates this need, intensifying competition and pressure to reduce time-to-market. Post-silicon validation is crucial for ensuring SoC functionality under real conditions, despite challenges like limited observability and reliance on fabricated chips [1].

Unfortunately, the validation produces vast data, including logs, traces, and oscilloscope images. While the log files store information regarding a test run such as a test setup, the trace data files provide insights into the internal register statuses and various sweep parameters such as voltage and frequency. Manually analyzing and debugging these files is time-consuming, inefficient, and error-prone. To address these issues, we propose an AI-driven method to automatically extract critical log messages and detect outliers in the trace data, reducing manual effort for debugging. For automatically extracting critical logs, we train a novel LSTM neural network model using labeled datasets from various SoCs validation projects. The LSTM model learns to capture complex sequential patterns in log messages, identifying anomalous text segments in various log categories. To ensure more reliable SoCs, our focus is to minimize false negatives (high recall) while simultaneously keeping false positives within an acceptable range (high precision) to reduce manual debugging. Our proposed model achieves high recall/precision of 94%/99%, 99%/99%, 92%/64%, and 98%/88% for normal, information, error, and warning log categories, respectively.

For outliers detection in the trace data, we propose an unsupervised Isolation Forest based method, which uses a synthetic anomaly injection approach to overcome the issue of unlabeled trace data. The method achieves high recall/precision of 95%/100% and 92%/73% for anomalous data points within two datasets and almost 100% for normal data points.

While the state-of-the-art has used machine learning (ML) for post-silicon validation [2]–[4], our work advances the state-of-the-art by applying supervised learning on a dataset annotated by experts for the log analysis, achieving high accuracy and a novel unsupervised learning technique that analyzes the trace data for each sweep group to detect outliers. In summary, we make the following main contributions:

- We propose an LSTM model for the automated extraction of critical log messages from large datasets. We train our multi-class classifier model on expert-annotated data.
- Our model exhibits a high recall/precision of 94%/99%, 99%/99%, 92%/64%, and 98%/88% for normal, information, error, and warning log categories, respectively.
- We also propose an Isolation Forest-based method for automatic outlier detection in trace data, using synthetic anomaly injection to handle the issue of unlabeled data.
- The outlier detection method achieves 95%/100% and 92%/73% recall/precision for anomalous data points for two datasets and almost 100% for normal data points.

The paper is organized as follows: Section II describes the AI models, Section III outlines the experimental setup, Section IV presents the results, Section V discusses related work, and Section VI provides the conclusions.

978-1-6654-7763-5/25/$31.00 ©2025 IEEE

II. Automated Log Extraction and Trace Data Outlier Detection

A. Overview of Post-silicon Debugging Process

The post-silicon validation and debugging environment, at a high level, consists of a *Validation Application* to run tests on a *Device Under Test (DuT)*, a database to store all results, and a *Test Owner* responsible for implementing and debugging the tests. A *Test Run* is an instance of a specific test, containing several parameter *sweeps* that generate the result files. The result files include: 1) *Log files* with a complete log of all activities before, during, and after a test run. 2) *Trace data files* containing sweeps and measurement data with both successful and failed results. 3) *Oscilloscope images* recording I/O signals, and 4) Other test-specific data.

B. Log Data Preparation for Training

A log file is generated after a test run and it contains important information such as whether any exception, error etc. occurred during the test run. For the analysis, the log messages are categorized into 3 base classes based on the information severity. The base classes are divided into sub-classes. In total, we have 12 different classes. Normal and info messages belong to class 1, warnings belong to class 2, and all kinds of anomalous messages belong to class 3. From a huge pile of log messages, the goal is to extract and summarize class 3 log messages. Table I shows the 3 base classes and sub-classes of the log messages. We preprocess the log messages before the training in various steps as described below:

TABLE I: Three classes of log messages based on severity.

Class	Sub-Classes	Extract
Class 1	Normal, Info, Setup Done	False
Class 2	Warnings	False
Class 3	Exceptions, Errors, Test Failed, Setup Failed, Timing Violation, Thermo-Stream Failed, Sweep Failed, Frame Error	True

Log Tagging: Log messages in each file are tagged manually based on the sub-classes. Table II shows an example of data after the manual multi-class log tagging.

TABLE II: Representative logs after the manual tagging.

Example Log Messages	Possible Classes
2023-02-25 13:21:34, [Error] Timing Violation	[Error, Violation]
2023-02-25 14:01:22, [Info] Test Succeed	[Info, Normal]
2023-02-25 15:05:02, [Error] Exception occurred	[Error, Exception]

Features and Labels: The data is converted into features and labels as shown in Table III. Features refer to log messages for training and the labels are the corresponding classes. There are 12 labels, however, we only show 3 for brevity.

TABLE III: Representative logs and labels.

Example Log Messages	Info	Error	Violation
2023-02-25 13:21,[Error] Timing Violation	0	1	1
2023-02-25 14:01,[Info] Test Succeeded	1	0	0
2023-02-25 15:05,[Error] Exception Occurred	0	1	0

Adding Weights: As most log messages are normal and only a fraction are abnormal, a class imbalance problem occurs which we fix by assigning higher weights to the classes with least amount of data as shown in Table IV.

TABLE IV: Adding weights to imbalanced classes.

Log Classes	Count	Weight
Normal	145,923	1.0
Info	121,016	1.0
Error	2,929	2.6
Violation	9	6.7
Exception, frame was not acknowledged	5	8.5

Cleanup, Tokenization, Sequencing, and Padding: We cleanup, tokenize, sequence, and pad the data as shown in Figure 1. We then feed the sequence of numbers into the embedding layer of a multi-class classifier, which learns and transforms them into meaningful context vectors.

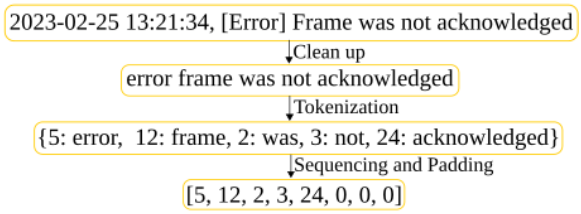


Fig. 1: Cleaning up, tokenizing, sequencing, and padding data.

C. Trace Data Preparation for Training

We preprocess trace data for training in three steps:

- Dataset Preparation: Concatenate trace files from multiple test runs into a comprehensive table containing sweeps and trace results.
- Feature Engineering: Analyze the dataset to explore correlations and fill missing values with domain knowledge.
- Extracting and Scaling: Extract numerical columns and scale the data for outlier detection.

D. Multi-class Log Classifier

We use an LSTM model for multi-class log classification, leveraging its ability to capture long-range dependencies and context [5]. The input format is (n, s, d) = (samples, sequence length, data dimension), where each token is processed at each time step. While longer sequences improve accuracy, they also increase training time. Testing sequence lengths of 500, 600, and 700 showed consistent accuracy. To extract base class 3 log messages, we train an LSTM model to predict if a log message belongs to one of 12 classes, with the final Dense Layer using a Sigmoid activation to output class probabilities. A threshold is applied to determine the class. The multi-class approach improves robustness, ensuring anomalies are captured even if one category is misclassified. After training, the model predicts classes for new log data.

E. Trace Data Outlier Detection

We use Isolation Forest for unsupervised outlier detection in trace data [6]. The algorithm trains by creating an ensemble of trees (iTrees) and assigns anomaly scores based on tree

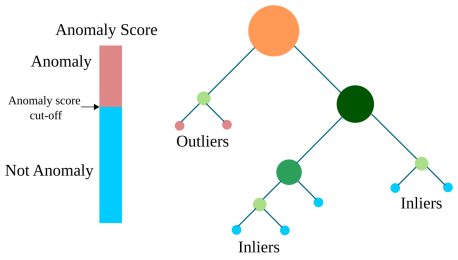


Fig. 2: An example of Isolation Forest tree.

depth. Anomalies are identified when their scores exceed a cut-off value. Figure 2 illustrates the Isolation Forest's working principle. The algorithm keeps cutting away the data points until all points are isolated from one another. Since the two data points on the left branch are far away from the other points, they are marked as anomalies. For these two data points, the anomaly score is higher than the cut-off value. The *contamination* parameter estimates anomaly percentages, but we manually define a cut-off after analyzing decision scores. Initially, we train in *auto* mode before refining the threshold.

III. Experimental Setup

We use TensorFlow and standard Python libraries for experiments. The data comes from NXP's SoC projects, with log data validated against a labeled dataset and trace data cross-verified with injected anomalies.

A. Multi-class Log Classifier

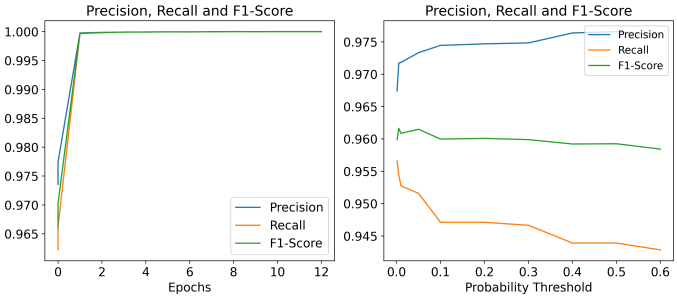
We train and evaluate the multi-class log classification model on a total of 282,599 log messages. For the training, validation, and testing we use 228,905, 25,434 (10% of training), and 28,260 (10% of total) messages, respectively. The training took about 6 hours on Intel core i5 with an integrated GPU. We use the Adam optimizer, binary cross-entropy as the loss function, and precision, recall, and F1-score metrics to evaluate the accuracy of our models.

B. Unsupervised Trace Data Outlier Detection

A dataset is created by concatenating multiple data files for a specific test case, but factors like test-case dependency, SoC architecture, and identical feature columns must be considered. Thus, we used data from a recent SoC project where column names were unified. We use 24 trace files with 518 data points (sweeps) from a communication test case. Due to the lack of labeled data, we apply an unsupervised outlier detection algorithm. However, without labeled data, identifying anomalies is difficult. To address this, we employ synthetic anomaly injection [4], generating a labeled dataset with known anomalies for training and evaluation.

Synthetic anomalies are injected by analyzing each sweep group's data, calculating means and standard deviations (STD), and adding anomalies a few STDs away from normal data. These are mixed with normal data, resulting in 538 data points in dataset 1 and 530 in dataset 2. We evaluate on both datasets since unsupervised clustering requires no train/test split.

We train forest algorithm with 100 estimators, setting *max features* to 1.0, and *max samples* and *contamination* to *auto* for automatic sample size and contamination determination.



(a) On training and validation data during the training at 0.5 probability.

(b) On test data with varying probability thresholds.

Fig. 3: Precision, recall, and F1-scores across all sub-classes.

IV. Results

A. Evaluation of Multi-class Log Classifier

1) Model's Performance: Figure 3a shows the precision, recall, and F1-score at a 0.5 probability threshold. The threshold determines the log message class, with values above classified as 1 and below as 0. The figure shows metrics approaching 1.0. During inference, precision increases by raising the threshold, while recall increases by lowering it (see Figure 3b).

2) Effect of Probability Threshold: Figure 3b shows how precision, recall, and F1-score change with the probability threshold. There is a trade-off, as increasing recall reduces precision. We use a lower threshold of 0.05 to improve recall and minimize the risk of missing anomalies, helping classify base class 3 log messages correctly. While this may increase false positives, it reduces false negatives, which are more critical for post-silicon validation.

3) Model Evaluation on Test Data: There is a confusion matrix for each of the 12 sub-classes. Figure 4 shows results for some classes, with 94.4% and 99.4% of data points correctly classified for normal and info categories. For normal, there are 835 false negatives and 127 false positives. For error, info, and warning, there are 28, 66, and 4 false negatives, and 171, 1, and 26 false positives, respectively.

TABLE V: Precision and recall at 0.05 probability threshold.

Decision	Precision	Recall	Decision	Precision	Recall
Normal	0.991	0.944	Exception	0.152	0.116
Info	0.999	0.994	Test Failed	0.181	0.522
Error	0.647	0.918	Frame Error	0.643	1.000
Warning	0.887	0.981	Violation	0.004	1.000

Table V shows the multi-class log classifier performance on the test data at a 0.05 probability cut-off. The model achieves high recall/precision for frequent categories: normal (94%/99%), info (99%/99%), error (92%/64%), and warning (98%/88%), highlighting its ability to reduce manual post-silicon validation effort. The model performs better for categories with more data points. The test data, 10% of randomly selected pre-processed data, is used only for evaluation, so some sub-classes may be underrepresented.

B. Evaluation of Trace Data Outlier Detection

Table VI shows the number of predicted outliers (56 / 109 for dataset 1 / dataset 2) exceeds the number of injected outliers (20 / 12 for dataset 1 / dataset 2). To further analyze

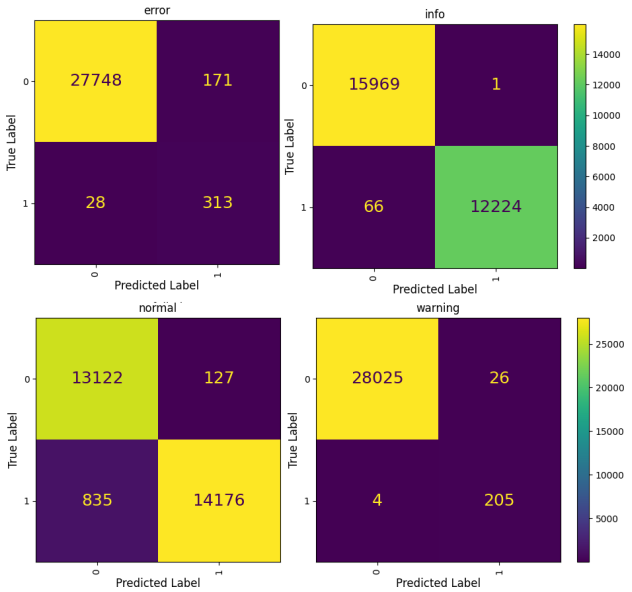


Fig. 4: Performance of the multi-class classifier on test data for selected sub-classes at a probability threshold of 0.05.

TABLE VI: Number of predicted outliers in the auto mode.

Dataset	# of Inliers	# of Outliers	STD of Anomaly Score
1	482	56	0.0502
2	421	109	0.0462

these results, we examine the decision scores provided by the Isolation Forest algorithm and establish a cut-off threshold.

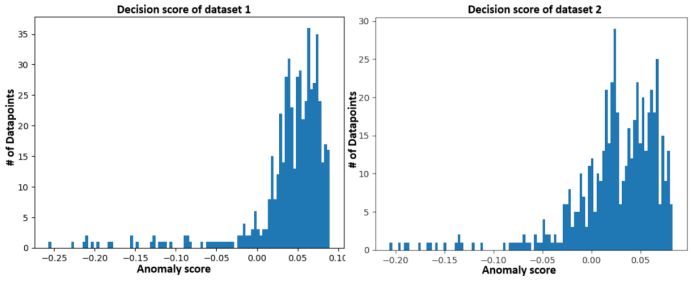


Fig. 5: Anomaly decision score for datasets 1 and 2.

Figure 5 shows histograms of decision scores for datasets 1 and 2. To balance anomaly detection and false positives, we selected a threshold of -0.1. Figure 6 shows that the Isolation Forest algorithm correctly identifies 19 of 20 anomalies (95%) in dataset 1 and 11 of 12 (92%) in dataset 2.

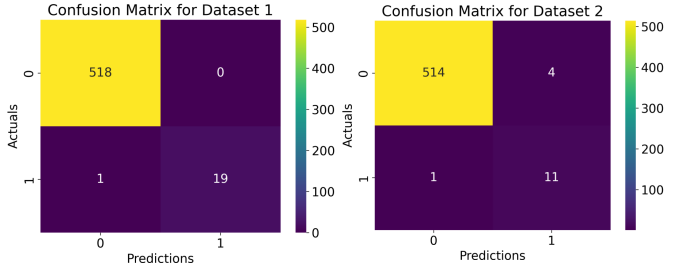


Fig. 6: Performance of the outlier detection method for datasets 1 (left) and 2 (right) (0 = Not Outlier, 1 = Outlier).

Table VII shows the performance of the outlier detection model on datasets 1 and 2 for decision score of - 0.1.

TABLE VII: Precision, recall and F1-score of the model.

Dataset	Decision	Precision	Recall	F1-score	Thresh
Dataset 1	Not outlier	1.00	1.00	1.00	-0.10
	Outlier	1.00	0.95	0.97	-0.10
Dataset 2	Not outlier	1.00	0.99	1.00	-0.10
	Outlier	0.73	0.92	0.81	-0.10

The model achieves high recall/precision of 95%/100% and 92%/73% for anomalous data points for datasets 1 and 2 and almost 100% for normal data points. The results show that AI can significantly automate the complex process of post-silicon validation, enabling faster delivery of more reliable SoCs.

V. Related Work

While ML has been used for log analysis [2]–[4], [7], our work advances several aspects. Unlike prior studies on HPC performance issues [2], [8], we focus on post-silicon debugging logs for hardware validation. Amrouch et al. [9] optimize registers via reinforcement learning but do not process logs, while DeOrio et al. [4] estimate bug cycles. We employ an LSTM classifier on expert-labeled post-silicon logs, integrating NLP for high accuracy. Mandouh et al. [10] use K-means for rare trace clustering. Similarly, we apply unsupervised clustering but analyze anomalies within sweep groups, as their output patterns differ. To manage unlabeled data, we inject synthetic anomalies.

VI. Conclusions

We propose automated log extraction and trace data outlier detection methods for efficient post-silicon validation. Our LSTM-based log classifier achieves high recall/precision rates of 94%/99%, 99%/99%, 92%/64%, and 98%/88% for normal, information, error, and warning log categories, respectively. For trace data, we use Isolation Forest with synthetic anomaly injection to handle unlabeled data, ensuring reliable outlier detection across sweep groups. The outlier detection method achieves recall/precision rates of 95%/100% and 92%/73% for anomalies across two datasets, and nearly 100% for normal data. These techniques significantly aid post-silicon validation by extracting critical insights efficiently.

References

[1] S. Mitra, S. A. Seshia, and N. Nicolici, "Post-Silicon Validation Opportunities, Challenges and Recent Advances," in *DAC*, 2010.

[2] C. Egersdoerfer, D. Zhang, and D. Dai, "ClusterLog: Clustering Logs for Effective Log-based Anomaly Detection," in *IEEE FTXS'22*.

[3] P. M. Paidipeddi Pridhiviraj, Tomar Dheerendra S, "Adaptive Post-silicon Server Validation using Machine Learning," *IJAIS*, 2015.

[4] A. DeOrio, Q. Li, M. Burgess, and V. Bertacco, "Machine Learning-based Anomaly Detection for Post-silicon Bug Diagnosis," in *DATE'13*.

[5] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, 1997.

[6] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *IEEE International Conference on Data Mining*, 2008.

[7] A. Dwaraki, S. Kumary, and T. Wolf, "Automated Event Identification from System Logs Using Natural Language Processing," in *ICNC*, 2020.

[8] A. Das, F. Mueller, P. Hargrove, E. Roman, and S. Baden, "Doomsday: Predicting Which Node Will Fail When on Supercomputers," in *SC18*.

[9] H. Amrouch *et al.*, "Machine Learning for Test, Diagnosis, Post-Silicon Validation and Yield Optimization," in *IEEE ETS*, 2022.

[10] E. Mandouh and A. G. Wassal, "Application of Machine Learning Techniques in Post-Silicon Debugging and Bug Localization," *E. Test.'18*.