Intrusion Detection Monitoring for Linux

John Ramsden
University of British Columbia
john@ece.ubc.ca

Abstract

Traditional monitoring tools often overwhelm administrators with excessive noise, hindering real-time detection of actionable threats. This project introduces a real-time automated detection system leveraging the extended Berkeley packet filter (eBPF) for system-call tracing on Linux systems. The captured data is analyzed by a machine learning model to identify anomalous patterns indicative of malicious activity. Unlike traditional methods, this approach prioritizes reducing alert fatigue by filtering noise and delivering precise, actionable insights. Evaluation focuses on high accuracy, low false-positive rates, and operational efficiency, aiming to deliver a scalable, production-ready solution that enhances Linux system security.

1 Introduction

Existing auditing tools such as `auditd` [1] provide methods to log certain system-calls and resource accesses. However, the generality and large volume of alerts often overwhelm administrators, making it difficult to detect and respond to actual threats [13, 18]. This project aims to design an automated detection tool that monitors Linux systems for signs of abuse. While the tool can be categorized as an intrusion detection system (IDS), it is distinct in focusing exclusively on system-call monitoring, rather than relying on diverse metrics such as network data or generic service logs commonly used in IDS systems.

The first issue this project addresses is the excessive noise generated by traditional monitoring tools. Detailed logs from tools like `auditd` often overwhelm administrators and cause alert fatigue, leading to critical alerts being overlooked [14]. This can lead to missed critical events and delayed responses to threats.

The second issue is that of system modification. Previous work has largely focused on monitoring various logs, often requiring modification of either the monitoring application to parse diverse log formats or the system to standardize log formats. These approaches do not scale well across large numbers of services and necessitate ongoing maintenance. In contrast, this project requires no modification on individual hosts beyond the installation of a system-call monitor.

To detect abnormal behaviors such as privilege escalation, this project uses system-call tracing analyzed by a machine learning model. The complexity of modern attacks makes them difficult to detect reliably with rule-based systems [12, 18]. Machine learning, with its ability to identify complex patterns, is better suited to this task. For evaluation, the ADFA-LD [5] dataset is used. This dataset contains system-call traces from simulated intrusion scenarios and is a recognized benchmark for intrusion detection systems (IDS).

The central question this project seeks to address is whether a machine learning-based system-call monitoring tool can effectively detect malicious behavior while reducing irrelevant alerts that burden administrators. The innovation of this project lies in its targeted and automated detection of abuse in Linux environments, aiming to reduce administrator overhead and enable prompt responses to threats without manual filtering.

2 Background

Detecting intrusions on Linux systems is critical for maintaining security and reliability. IDS identify threats and abnormal behavior by analyzing system metrics such as network logs, system logs, and system-calls. Among these, system-call monitoring offers a standardized and comprehensive method for gathering metrics across an entire system.

System-calls serve as the primary interface between user applications and the operating system. By analyzing sequences of system-calls, it is possible to detect patterns indicative of abnormal or malicious activity. Traditional tools like `auditd` provide mechanisms to monitor system-calls, but they often generate excessive alerts and impose significant overhead. This can overwhelm administrators and lead to alert fatigue, where critical threats are overlooked. eBPF addresses these challenges by enabling the execution of lightweight programs directly within the kernel. This allows eBPF to collect key metrics in real time with minimal impact on system performance. Unlike traditional tools, eBPF's low overhead and flexibility make it an ideal foundation for modern intrusion detection systems. This paper leverages eBPF for system-call monitoring, avoiding many of the drawbacks associated with legacy approaches.

Identifying meaningful patterns in system-call data, however, requires advanced analytical methods. Machine learning has become a cornerstone of anomaly detection, offering the ability to uncover complex patterns that rule-based systems struggle to detect. For sequential data like system-call traces, models such as Long Short-Term Memory (LSTM) [10] networks and Temporal Convolutional Networks (TCN) [3, 11] excel at capturing temporal dependencies. These models provide a robust framework for detecting abnormal behavior and adapting to evolving threats.

3 Related Work

Creech and Hu [6] introduced a semantic approach to analyzing system-call patterns, demonstrating its effectiveness in detecting anomalies. Similarly, the ADFA-LD [5] dataset, which consists of labeled system-call traces, has become a standard benchmark for evaluating IDS. However, many prior implementations relied on rule-based approaches, which often struggle to identify complex sequences of anomalous behavior.

eBPF has emerged as a powerful tool for real-time monitoring of Linux systems due to its low overhead and flexibility. Falco [2] is a widely used intrusion detection tool for cloud-native environments that leverages eBPF to monitor key metrics, including system-calls and network traffic. However, unlike the machine learning approach explored in this project, Falco relies on rule-based anomaly detection, which, while effective, lacks the adaptability to handle complex and evolving threats.

Machine learning has been extensively applied to anomaly detection. While LSTM models [10] are commonly used for sequence analysis, Bai et al. [4] demonstrated that TCN models provide higher accuracy and better performance for certain tasks. This project builds on these findings by comparing LSTM and TCN models to determine the most effective approach for system-call analysis.

Existing solutions also often fail to address the problem of alert fatigue. Kearney et al. [14] investigated the severe impact of high false positive rates on administrators and highlighted the potential of machine learning to mitigate this issue. Their findings emphasize the need for actionable alerts and low false positive rates. This project aims to bridge this gap by combining the strengths of eBPF and machine learning to create a high-accuracy, low-noise intrusion detection system tailored for Linux environments.

4 Solution

This project leverages eBPF [8] to efficiently capture system-wide system-calls on Linux in real time, which are then sent to a remote machine learning pipeline for anomaly detection. The primary objective is to identify anomalous behavior by analyzing patterns in sequences of system-calls using a pre-trained machine learning model.

An eBPF program is used to monitor all system-calls across the system. The program captures system-calls and streams them to the machine learning pipeline, as illustrated in Fig. 1. While this implementation does not yet include real-time streaming directly to the model for classification, metric recording, and notifications, it provides the necessary building blocks for achieving full real-time alerting functionality.

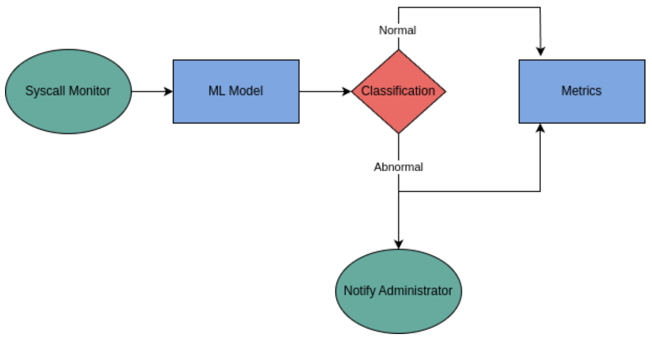

Figure 1: High-level flow of the proposed system.

The eBPF monitor relies on a simple `kprobe` [15] handler attached to the kernel for each system-call. As shown in Fig. 2, the eBPF code uses a minimal `printk` function call to emit numeric identifiers corresponding to the operation codes (opcodes) of system-calls. System-call arguments were intentionally excluded, as they often introduce sparsity in the dataset [6]. Including such features risks overfitting, as the model might assign undue importance to rare arguments.

Machine learning models analyze the collected system-call data to detect anomalous patterns. Two models were evaluated: LSTM and TCN. These models, pre-trained on the ADFA-LD dataset [5], analyze sequences of system-calls to identify deviations indicative of suspicious activity. Both LSTM and TCN are designed for sequential data and excel at capturing temporal dependencies. However, these models require fixed-length inputs, necessitating the use of a sliding window technique. This technique divides sequences into fixed-length segments, preserving temporal information while ensuring compatibility with the model's input requirements.

To optimize model performance, Bayesian optimization [17] and five-fold cross-validation were used to fine-tune critical hyperparameters, such as window size. This approach efficiently explored the hyperparameter search space and mitigated overfitting, with window size emerging as a key factor in improving accuracy.

```
SEC("ksyscall/read")
int BPF_KSYSCALL(read_entry)
{
    bpf_printk("IDSTAG,0");
    return 0;
}

// ...

SEC("ksyscall/setuid")
int BPF_KSYSCALL(setuid_entry)
{
    bpf_printk("IDSTAG,105");
    return 0;
}
```

Figure 2: Example eBPF syscall probes for `read` and `setuid`.

By combining the efficient data collection capabilities of eBPF with advanced machine learning techniques, this solution provides a foundation for a real-time, low-overhead intrusion detection system capable of addressing the challenges posed by traditional monitoring tools.

5 Evaluation

The evaluation of the proposed system focuses on two primary aspects: the effectiveness of the machine learning models in detecting anomalies and the performance of the eBPF monitor. These aspects are measured using a combination of accuracy, efficiency, and resource utilization metrics. Experiments were conducted on two systems: an Ubuntu machine with a single core and 1GiB of RAM for the eBPF monitor, and an Arch Linux system with 24 cores, 128GiB of RAM, and an Nvidia GeForce RTX4090 GPU for model training and evaluation.

5.1 Machine Learning Model Performance

To assess the performance of the machine learning models, the ADFA-LD dataset [5] was used. This dataset consists of 1,579 labeled intrusion traces, divided into 1,263 training traces and 316 testing traces (80/20 split), corresponding to 491,284 training system-calls and 134,182 testing system-calls.

The models were evaluated based on accuracy, precision, recall, F1-score, inference time, and the rate of system-calls processed per second. Precision was prioritized to minimize false positives and reduce administrator fatigue, while recall ensured that most anomalies were detected. The F1-score provided a balanced metric by combining precision and recall. Table 1 summarizes the performance of the TCN and LSTM models.

While both models achieved high accuracy (roughly 95%), they exhibited complementary strengths. The LSTM model had higher recall and F1-score, making it slightly better at detecting anomalies overall. However, the TCN model demonstrated superior precision (96.18%) and significantly faster inference time (approximately 1/3 of LSTM's), processing over three times as many system-calls per second. This makes TCN particularly well-suited for high-throughput environments. Throughout the experiments typical system-call rate was found to be in the low thousands per second, however busy systems can execute upwards of 70,000 system-calls per second [9]. This underscores the importance of efficiency.

Table 1: Model Performance Comparison

Model	Accuracy	Precision	Recall	F1-Score	Inference Time	Syscalls per sec
TCN	0.9494	0.9618	0.9421	0.9488	2.7315s	49,124
LSTM	0.9567	0.9425	0.9776	0.9574	8.7281s	15,373

Figures 3 and 4 provide confusion matrices for the two models. The TCN model exhibited a false positive rate of 4.3%, compared to 6.4% for LSTM. In the example of the training data, this would correspond to 885 fewer alerts the administrator would observe as false positives. While the LSTM model performed better on false negatives, minimizing false positives is prioritized to reduce alert fatigue.

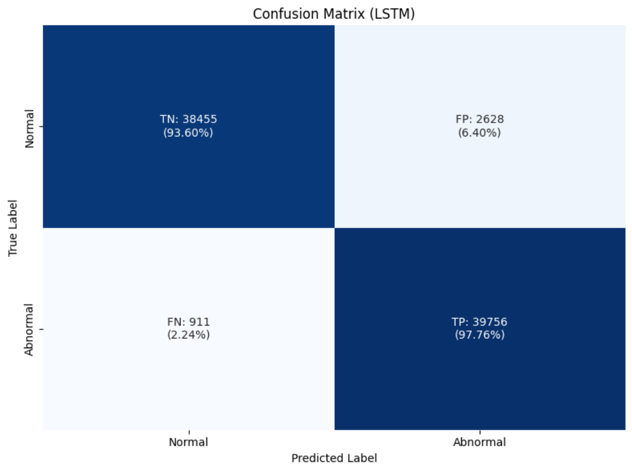

Figure 3: LSTM confusion matrix.

Receiver operating characteristic (ROC) curves demonstrate how well models are able to discriminate between different classes. Fig. 5 shows the ROC curve of both TCN and LSTM. They do this by plotting the true positive rate against the false positive rate while adjusting the classification threshold throughout. A perfect classifier reaches the top left corner, yielding a true positive rate of 1 and a false positive rate of 0. The Area Under the Curve (AUC) score quantifies how well the model is doing at discriminating, with 1.0 being a perfect score. Both the LSTM and TCN models demonstrate strong discrimination with AUC scores of 0.97. This shows that the models achieve minimal false positives while maintaining high precision and accuracy.

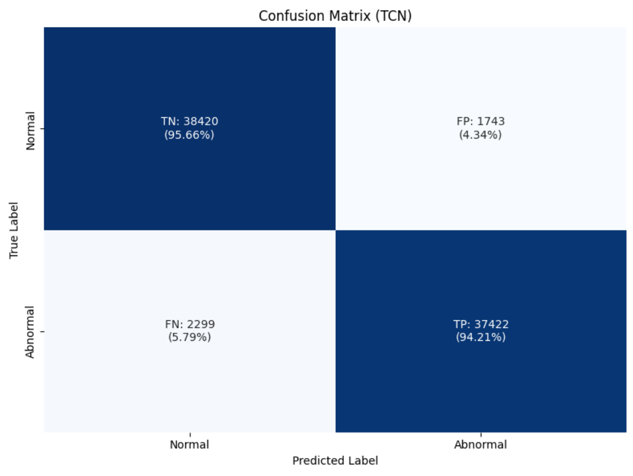

Figure 4: TCN confusion matrix.

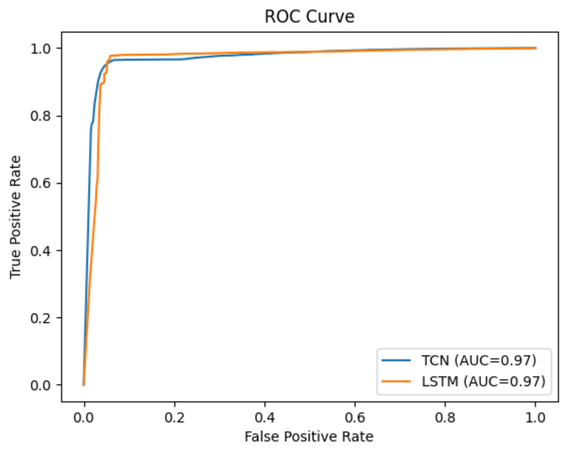

Figure 5: ROC curve for LSTM and TCN models.

5.2 eBPF Monitor Performance

The eBPF monitor was evaluated for resource utilization and latency. Resource usage was measured during a time-of-check time-of-use (TOCTOU) attack exploiting CVE-2018-0492 [7]. Table 2 compares the eBPF monitor's overhead against `auditd`. The eBPF monitor added only 0.63% to system utilization, significantly less than the 52.04% overhead introduced by `auditd`.

Additionally latency of a system-call being recorded was measured. To do so a new kprobe was added for the open call (Fig. 6). This kprobe triggers only from opens that originated from a specific python script where latency is being measured. In the python script, a timestamp is recorded followed by an open system-call, this triggers the eBPF probe to log. Once the eBPF log updates a new time stamp is recorded and as a result the latency of system-call to data being logged is known. The results confirm that the eBPF monitor is capable of providing real-time data without introducing significant delays. Fig. 7 and Fig. 8 present scatterplot and histogram analyses of latency, which averaged 18.82 microseconds.

Table 2: eBPF vs auditd system utilization

Tool	usr	sys	iowait	Total	Net increase vs baseline
Baseline	6.36	38.22	0.02	44.56	0
eBPF	6.21	38.96	0.02	45.19	0.63
auditd	35.77	52.04	8.79	96.6	52.04

```
SEC("ksyscall/open")
int BPF_KSYSCALL(open_entry,
                 const char *pathname)
{
    char comm[TASK_COMM_LEN];
    bpf_get_current_comm(
        &comm, sizeof(comm)
    );
    int cmp = __builtin_memcmp(
        comm, "read_lat.py", 11
    );
    if (cmp != 0) {
        return 0;
    }
    bpf_printk("READMETRIC␣%s", pathname);
    return 0;
}
```

Figure 6: eBPF probe for latency measurement.

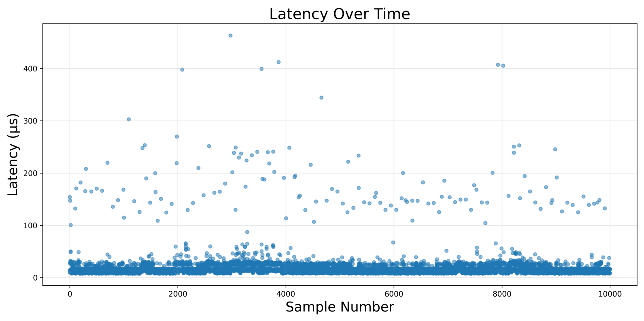

Figure 7: Scatterplot of eBPF latency measurements.

5.3 Conclusion

The evaluation demonstrates that the TCN model is better suited for real-time anomaly detection due to its higher precision and efficiency. The eBPF monitor's low latency and minimal resource usage further validate its applicability in production environments, even under high system-call rates.

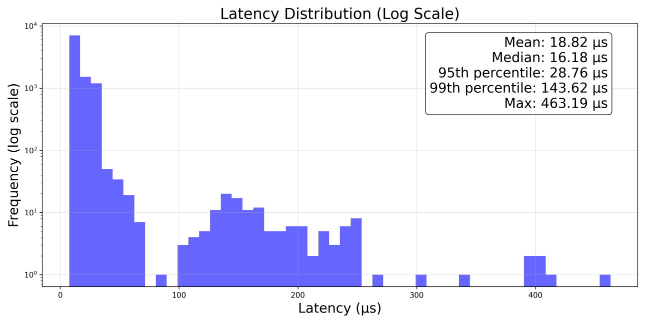

Figure 8: Histogram of eBPF latency measurements.

6 Limitations and Future work

The current work demonstrates that a TCN-based model can effectively identify system-call sequences with high accuracy and a low false positive rate. Additionally, the eBPF-based monitor provides system-call sequences with low latency and minimal overhead, making it suitable for real-time data collection. However, several components remain unimplemented, representing key areas for future work.

6.1 Known Limitations

There are some known limitations of the evaluated system:

1. **Performance Under High Load**: - In environments with high system-call rates (e.g., exceeding 50,000 calls per second), the machine learning model may struggle to process data in real time on hardware similar to that used in this evaluation. While introducing a queue could help buffer system-calls during temporary spikes, this approach may not suffice if the load remains consistently high.

2. **Inference Speed**: - The current system relies on pre-trained models with relatively high computational requirements. The inference time of the TCN model, while faster than LSTM, may still require further optimization to handle real-time demands at scale.

3. **Monitoring Scope**: - Monitoring all system-calls may introduce unnecessary overhead. Reducing the scope to focus on critical system-calls could improve performance, but further evaluation is needed to ensure that accuracy is not compromised.

6.2 Future Work

To address these limitations and advance the system toward a fully functional real-time notification tool, several areas of future work are proposed:

1. **Generating Real-World System-Call Traces**: - A comprehensive collection of system-call traces simulating real-world abnormal behavior must be developed. This involves setting up exploitable systems and simulating attacks, such as privilege escalation or file manipulation, that exploit known vulnerabilities. These traces will provide a more realistic training and testing environment compared to the ADFA-LD dataset.

2. **Cloud Deployment of Machine Learning Models**: - Deploying the machine learning model in a cloud environment would enable real-time classification of system-call sequences. Cloud-based deployment would allow for scalability and facilitate integration with distributed monitoring systems.

3. **Real-Time Alert Mechanism**: - A mechanism to notify administrators when abnormal behavior is detected needs to be integrated. This could include customizable alert thresholds and interfaces that prioritize actionable alerts to reduce the risk of administrator fatigue.

4. **Inference Optimization**: - Optimizing the inference rate of the machine learning models is critical for high-throughput scenarios. Techniques such as model quantization, pruning, or using hardware accelerators like GPUs or TPUs could significantly reduce latency.

5. **Selective System-Call Monitoring**: - Reducing the number of monitored system-calls may improve system performance. Future work should involve identifying a minimal set of critical system-calls and evaluating whether this reduction maintains acceptable levels of accuracy.

6. **Exploration of Advanced Machine Learning Techniques**: - Integrating advanced techniques, such as reinforcement learning, could allow the system to adapt dynamically to evolving threats. This approach may enhance detection capabilities, particularly in highly variable environments.

By addressing these limitations and advancing the proposed future work, the system can be developed into a robust, real-time intrusion detection solution capable of operating effectively under high-load conditions.

7 Conclusion

This project proposes a system for detecting abnormal behavior in Linux environments by leveraging eBPF for real-time system-call tracing and a machine learning-based detection pipeline. By utilizing a TCN model, the system achieves high accuracy and a low false positive rate while maintaining minimal latency, addressing key limitations of traditional monitoring tools.

The integration of eBPF ensures efficient, low-overhead data collection and processing, making the system scalable and suitable for production environments. The use of machine learning enables the accurate detection of malicious behavior by learning from both benign and anomalous activity, significantly reducing false alerts and alleviating administrator fatigue.

The evaluation demonstrates the system's effectiveness, achieving accuracy rates of 95% and a low false positive rate of 4.3% with the TCN model. The eBPF monitor further complements this by providing system-call data with an average latency of only 18.82 microseconds and minimal resource utilization, making it practical for real-world deployment.

However, challenges remain, particularly in handling high system-call rates and achieving real-time notification capabilities. Future work includes optimizing inference performance, generating real-world attack traces, and integrating an alerting mechanism to create a fully operational intrusion detection system.

This project demonstrates the potential of combining eBPF and machine learning to advance host-based intrusion detection systems, offering a scalable and efficient solution for modern Linux environments. With further development, this work could contribute significantly to improving security monitoring and threat detection in real-world applications.

Availability

All source code for the projects described in the paper can be found at

- `github.com/johnramsden/ADFA-LD-Analysis`
- `github.com/johnramsden/ebpf-syscall-tracer`
- `github.com/johnramsden/exploits-pe-vuln`

8 Footnotes

ChatGPT [16] was used for assisting with text re-structuring and code generation [1]

References

[1] auditd - linux man page. https://linux.die.net/man/8/auditd. Accessed: 2024-12-08.

[2] The falco project. https://falco.org/. Accessed: 2024-12-13.

[3] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. In *arXiv preprint arXiv:1803.01271*, 2018.

[4] Yongliang Cheng, Yan Xu, Hong Zhong, and Yi Liu. Hs-tcn: A semi-supervised hierarchical stacking temporal convolutional network for anomaly detection in iot. In *IEEE IPCCC*, 2019.

[5] Gideon Creech and Jiankun Hu. Generation of a new IDS test dataset: ADFA-IDS. https://research.unsw.edu.au/projects/adfa-ids-datasets, 2013. Accessed: 2024-12-08.

[1]ChatGPT by OpenAI, accessed December 20, 2024, https://chat.openai.com/

[6] Gideon Creech and Jiankun Hu. A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns. In *IEEE TC*, 2013.

[7] National Vulnerability Database. Cve-2018-0492: Gnu c library vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2018-0492, 2018. Accessed: 2024-12-09.

[8] eBPF. ebpf - extended berkeley packet filter. https://ebpf.io/, 2024. Accessed: 2024-12-08.

[9] Luis Gerhorst, Benedict Herzog, Stefan Reif, Wolfgang Schröder-Preikschat, and Timo Hönig. Anycall: Fast and flexible system-call aggregation. In *ACM PLOS*, 2021.

[10] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. In *IEEE TNNLS*, 2016.

[11] Yangdong He and Jiabao Zhao. Temporal convolutional networks for anomaly detection in time series. In *Journal of Physics: Conference Series*, 2019.

[12] Fehmi Jaafar, Gabriela Nicolescu, and Christian Richard. A systematic approach for privilege escalation prevention. In *IEEE TIFS*, 2021.

[13] Weihang Jiang, Chongfeng Hu, Shankar Pasupathy, Arkady Kanevsky, Zhenmin Li, and Yuanyuan Zhou. Understanding customer problem troubleshooting from storage system logs. In *USENIX FAST*, 2009.

[14] Paul Kearney, Mohammed Abdelsamea, Xavier Schmoor, Fayyaz Shah, and Ian Vickers. Combating alert fatigue in the security operations centre. In *SSRN*, 2023.

[15] Jim Keniston, Prasanna S Panchamukhi, and Masami Hiramatsu. Kernel probes (kprobes). https://docs.kernel.org/trace/kprobes.html, 2024. Accessed: 2024-12-08.

[16] OpenAI. Chatgpt: Assisting with text structuring and restructuring. https://chat.openai.com, 2024. Accessed: 2024-12-08.

[17] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *NeurIPS*, 2012.

[18] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *ACM SOSP*, 2009.