International Conference KNOWLEDGE-BASED ORGANIZATION
Vol. XXXI No 1 2025

GUARD: A GUIDED AI SYSTEM FOR INTRUSION DETECTION AND AUTOMATED RESPONSE IN CRITICAL INFRASTRUCTURE ENVIRONMENTS

Ivan ZZIWA, Hrishikesh PAWAR, Cedric NARTEY, Maurice DAWSON

Illinois Institute of Technology, Chicago, USA
Ivanlivingstone206@gmail.com

Abstract: *An anomaly-based IDS using Large Language Models was developed by a team of four within a three-week time frame. The initiative commenced on July 25th, with the initial week dedicated to evaluating the research papers, sources, and existing code examples. The task was to implement the idea in a way that would encompass the supply of a fully operational IDS. Within the next three weeks, we developed Shell scripts in python to effectively capture and preprocess captured network packet data. This preprocessed data would be fed into an IDS to identify potentially suspicious activity. Empirical data indicated that the system had the capability to detect anomalies in the network traffic. Thereby, proving its value for enhancing the security controls through an IDS based on a language Model. The present study presents the potential for the augmentation of LLM-based solutions within the domain of intrusion detection.*

Keywords: Intrusion Detection Systems, Cybersecurity, AI, Networking, Critical Infrastructures

1. Introduction

Intrusion detection systems (IDS) play a crucial role in safeguarding digital infrastructure against cyber threats. However, traditional IDS approaches often struggle to keep pace with the evolving complexity and sophistication of modern attacks. This research explores the potential of leveraging large language models to enhance the capabilities of intrusion detection systems.

The limitations of conventional IDS include their reliance on predefined rules and signatures, which can be circumvented by novel attack vectors. Additionally, traditional systems often generate high false positive rates and struggle with real-time processing of large-scale network traffic. These shortcomings highlight the need for more adaptive and intelligent approaches to intrusion detection.

Artificial intelligence, particularly large language models, offers promising solutions to address these challenges.

By harnessing the power of natural language processing and deep learning, AI-based IDS can potentially improve anomaly detection, reduce false positives, and enhance real-time threat analysis.

This study aims to evaluate the shortcomings of traditional IDS, implement AI-based approaches using large language models, assess their performance and efficiency, and analyze their scalability and real-time processing capabilities. The research employs three main approaches: one-class classification with SecBERT, SecBERT embeddings with Isolation

DOI: 10.2478/kbo-2025-0028

Forest, and PyTorch Tabular.
By exploring these AI-enhanced methodologies, this study seeks to contribute to the advancement of intrusion detection systems and provide insights into the practical application of large language models in cybersecurity.

2. Background & Related Work

Traditional IDS typically relies on rule-based or signature-based detection, failing to mitigate novel threats. These systems either use signature-based methods, which prove to be highly effective for known attacks but fail to address zero-day exploits – or they utilize anomaly-based methods, which can detect novel threats but at the risk of providing higher false positive rates [1]. Recent approaches ponder on the incorporation of AI for anomaly detection, using statistical models, machine learning, and deep learning [2].
A recent study conducted by Markevych and Dawson (2023) highlighted the growing importance of large language models such as ChatGPT in cybersecurity. Emphasizing the potential of AI, their work highlights its usage to improve detection accuracy, reduce false positives, and enhance IDS capabilities through the usage of natural language processing for classifying threats in real time [2].

2.1. AI and Intrusion Detection

AI-driven IDS research has advanced rapidly, encompassing approaches from artificial neural networks to deep learning. Early work by Liao et al. (2013) established the foundation for anomaly-based detection using statistical models [1]. Khraisat et al. (2019) demonstrated the efficacy of CNNs and RNNs in high-throughput network environments but noted substantial computational costs [6]. More recent studies have explored transformer-based models: SecBERT has been shown to capture nuanced security log semantics, while GPT-4 and similar LLMs exhibit strong contextual understanding for threat detection [2].

2.2. Conventional vs. AI-Based IDS

Despite these advantages, AI-based IDS also have a few challenges. Research conducted by Sommer and Paxon caution that the introduction of machine learning to real time network traffic is not a straightforward process, as models may overfit to training data or create an excessive amount of false positives when confronted with "unpredictable 'background noise' of operational networks" [5]. A difference between conventional IDS is that AI models require diverse, extensive training data and feature engineering. Once well trained, these models can uncover subtle anomalies that static signatures miss. Kim et al. shows that deep neural networks can detect web-based intrusions in real time with high accuracy, outperforming typical pattern matching techniques [7]. Ensuring the performance of these systems over time demands continuous updates as network behaviour evolves, known as concept drift [5]. While recent research into efficient AI for security looks to address these concerns, it remains an ongoing challenge.

3. Subsystem Design and Functionality

This section details the design and implementation of the core GUARD subsystems, focusing on how raw network traffic is captured, preprocessed and transformed into inputs suitable for LLM-based anomaly detection models. Each module is designed to operate in real time, handle high throughput traffic and integrate seamlessly with downstream scoring and response components.

3.1. Data Collection Module

The Data Collection Module is responsible for continuously harvesting live network traffic and preparing it for analysis. A lightweight Python wrapper invokes Tshark on linux hosts, capturing packets in rolling intervals. For instance, this is done every second as the host exchanges packets with the web server. This approach helps in balancing granularity and storage constraints. Captured packet streams are

converted to CSV format using a pcap_to_csv.py script that extracts key attributes such as timestamp, source IP, destination IP, transport protocol, packet length and session metadata [1]. These CSV files act as a temporary staging area for downstream modules. During conversion, normalization routines unify timestamp formats, lowercase text fields and strip non alphanumeric characters from payload descriptors to ensure consistency. The resulting CSV dataset is stored on a local file system and indexed by capture window, enabling efficient batch loading for embedding and anomaly scoring.

A1 | fx No.

	A	B	C	D	E	F	G
1	No.	Time	Source	Destination	Protocol	Length	Info
2	1	0	192.168.229.254	192.168.202.79	TLSv1	117	Change Cipher Spec, Encrypted Handshake Message
3	2	0	192.168.202.79	192.168.229.254	TLSv1	269	Application Data
4	3	0	192.168.202.79	192.168.229.251	TCP	70	50463 > 80 [FIN, ACK] Seq=1 Ack=1 Win=980 Len=0 TSval=8812998 TSecr=46388446
5	4	0	192.168.229.254	192.168.202.79	TCP	70	443 > 46117 [ACK] Seq=48 Ack=200 Win=32768 Len=0 TSval=319644339 TSecr=8812998
6	5	0	192.168.202.79	192.168.229.251	TCP	78	50465 > 80 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=8812999 TSecr=0 WS=16
7	6	0	192.168.202.79	192.168.229.153	SMB	217	Session Setup AndX Request, User: \\administrator
8	7	0	192.168.229.251	192.168.202.79	TCP	70	80 > 50463 [ACK] Seq=1 Ack=2 Win=64074 Len=0 TSval=46388446 TSecr=8812998
9	8	0	192.168.229.254	192.168.202.79	TLSv1	178	Application Data
10	9	0	192.168.229.251	192.168.202.79	TCP	82	80 > 50465 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1380 WS=1 TSval=0 TSecr=0 SACK_PERM
11	10	0	192.168.229.254	192.168.202.79	TLSv1	120	Application Data
12	11	0	192.168.229.254	192.168.202.79	TLSv1	135	Application Data
13	12	0	192.168.229.254	192.168.202.79	TLSv1	299	Application Data
14	13	0	192.168.229.254	192.168.202.79	TLSv1	97	Application Data
15	14	0	192.168.229.254	192.168.202.79	TCP	70	443 > 46117 [FIN, PSH, ACK] Seq=527 Ack=200 Win=32768 Len=0 TSval=319644341 TSecr=0
16	15	0	192.168.229.153	192.168.202.79	SMB	109	Session Setup AndX Response, Error: STATUS_LOGON_FAILURE
17	16	0	192.168.202.79	192.168.229.251	TCP	70	50465 > 80 [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=8812999 TSecr=0
18	17	0	192.168.202.79	192.168.229.251	HTTP	236	HEAD /DEASLog02.nsf HTTP/1.1
19	18	0	192.168.202.79	192.168.229.153	SMB	212	Session Setup AndX Request, User: \\webadmin
20	19	0	192.168.229.153	192.168.202.79	SMB	109	Session Setup AndX Response, Error: STATUS_LOGON_FAILURE
21	20	0	192.168.229.251	192.168.202.79	HTTP	284	HTTP/1.1 404 Not Found
22	21	0	192.168.202.79	192.168.229.254	TLSv1	97	Encrypted Alert
23	22	0	192.168.202.79	192.168.229.254	TCP	70	46117 > 443 [FIN, ACK] Seq=227 Ack=528 Win=16616 Len=0 TSval=8813000 TSecr=319644341

Figure 1: Sample Dataset of captured packet streams

3.2. Preprocessing and Inferencing Script

The inferencing pipeline transforms staged CSV data into LLM embeddings and feature vectors for anomaly detection considered as a part of data preparation for ML workflows. First, a preprocessing script reads each CSV row to perform data preprocessing on the acquired dataset. This includes dropping irrelevant columns through feature engineering (e.g., frame number, raw IP addresses). The 'info' field is lowercase and filtered to remove punctuation and control characters. The 'protocol' column is label-encoded and numeric length values are retained. The cleaned fields are concatenated into text inputs compatible with SecBERT's tokenizer. Batches of up to 32 records are tokenized with dynamic padding and truncation to a maximum sequence length of 512 tokens, then fed to a pre-trained SecBERT model via the HuggingFace Transformers library [8]. The [CLS] token embedding is an aggregate representation of the input sequence which is extracted for each record, producing a 768 dimensional vector. Concurrently, the encoded protocol and scaled length values are projected into the same latent space and summed with the LLM embeddings to incorporate protocol-specific context. Finally, the combined feature vectors are standardized using a prefit scaler and exported as input to the anomaly detection model. Embedding extraction and preprocessing are designed to leverage GPU acceleration for low-latency inference in production environments.

3.3. Detection Module

The Detection Module implements three anomaly detection approaches, each leveraging SecBert embeddings:

1. **One-Class SecBERT Classification**: A one-class neural classifier built in PyTorch is trained on benign traffic embeddings to learn a decision boundary around "normal" behaviour. The model fine-tunes the SecBERT based on calibration data before freezing encoder layers and training a small dense head to distinguish normal samples. At inference, the classifier outputs a probability score, with samples below a threshold (e.g., 0.5) flagged as

anomalies. [9,10].

2. **SecBERT + Isolation forest**: Embeddings extracted from SecBERT are passed to an Isolation model Forest from scikit-learn (n_estimators = 100, contamination = 0.1) for unsupervised anomaly scoring. The isolation_forest model was trained on a baseline dataset of benign flows at real-time runtime, the *score_samples* method yields continuous anomaly scores, while it predicts output in binary normal / anomalous labels. A dynamic percentile-based thresholding (top 5% of scores) maintains sensitivity under varying network loads [11].

3. **PyTorch Tabular:** A third approach uses PyTorch Tabular to train an ensemble model combining SecBERT embeddings with feature engineered numeric features (protocol label, packet length). The ensemble consists of a simple feedforward network with two hidden layers (128 and 64 units) and dropout regularization. The model is trained for binary classification using cross-entropy loss, with early stopping based on validation F1 score. At inference, it produces an anomaly probability score for each record [12].

4. Security Design & Considerations

While GUARD's core research centred on detection and response architecture, deploying such systems in production mandates rigorous security precautions to safeguard against both system integrity and sensitive network data.

To ensure the security of both in-flight and stored data within the GUARD framework, two complementary encryption strategies are recommended. First, all inter-module communications, whether conveyed through message queue systems or representational state transfer application programming interfaces should be protected by Transport Layer Security version 1.2 (TLS 1.2) or above, as empirical analyses conducted by F5 Labs reveal that widespread adoption of this protocol markedly diminishes the risk of man-in-the-middle exploitation while imposing only a marginal performance burden on networks engineered for high throughput [13]. Second, to safeguard intermediate data artifacts such as raw packet captures and comma-separated value staging files, full-disk or file-level encryption mechanisms (for example, Linux Unified Key Setup) should be employed. These empirical studies indicate that using either self-encrypting drives or modern software-based encryption on NVMe storage yields minimal impact on system performance [14].

To complement these data protection measures, GUARD's exposed application programming interfaces for ingestion, inference, and response orchestration must enforce OAuth 2.0 token authentication in conjunction with role based access control (RBAC). As per recent implementations within industrial and Internet of Things (IoT) environments demonstrate that finely grained role definitions and well-managed token lifecycles are essential for thwarting unauthorized operations and constraining lateral movement [15]. Ensuring the integrity of SecBERT fine tuning artifacts and detection models entails adopting machine learning security operations best practices specifically, the use of cryptographic checksums and digital signatures to validate model binaries, together with a centralized model registry (for example, MLflow) that provides version control and rollback support, while the routine registry audits and dependency scans guard against tampering [16][17]. Furthermore, encapsulating GUARD subsystems within containers promotes modular deployment but introduces novel attack vector compliance with the CIS Docker Benchmark version 1.3.1 or later enforcing least privilege. The removal of superfluous capabilities and restrictive seccomp profiles combined with the automation of benchmark adherence checks in continuous integration and continuous delivery pipelines mitigates these risks at vast distances. Finally, comprehensive recording of administrative

actions, API requests, and configuration modifications is indispensable for forensic readiness where centralized event correlation not only accelerates incident triage but also enables long-term trend analysis, and emerging next generation SIEM architectures which offers scalable correlation engines that improves detection fidelity [18].

5. Integration & Deployment Strategies

This section maps the GUARD framework from rapid prototyping to production ready deployments, emphasizing modular integration, GPU-accelerated development, hybrid cloud versatility and practical maintenance cycles with automated updates.

During initial development, the team used Google Colab Pro+ notebooks provisioned with NVIDIA Tesla T4 GPUs and A100 GPUs. The GPU's parallel computer reduced embedding extraction and SecBERT fine-tuning times by up to 70% compared to CPU-only execution, enabling iterative experimentation on large batches. Notebooks integrated MLflow for experiment tracking with hyperparameters (learning rate, batch size), metrics(accuracy and loss) and model artifacts (tokenizer config, checkpoint files) were logged, ensuring reproducibility and facilitating collaborative review of model performance.

To extend beyond prototyping, the team adapted the framework for deployment on hybrid environments combining local clusters and cloud instances. Model checkpointing and reproducibility were ensured using MLflow, which also enabled consistent tracking of training metrics and artifact versioning.

Hyperparameter optimization was carried out manually through structured experimentation, with key metrics (validation loss, accuracy, F1-score) logged and compared across runs. This hands-on approach ensured greater control over model tuning and performance evaluation.

6. Evaluation & Testing

To assess GUARD's effectiveness in real-world conditions, we adopted a three-phase evaluation strategy aligned with our implemented workflows and available data.

6.1 Integration

Module functionality was verified using bespoke validation scripts included in our python toolkit rather than formal testing frameworks. For example, packet preprocessing scripts include assertion checks for expected CSV schema and handle edge cases such as zero-byte captures or malformed entries, raising descriptive errors when inconsistencies occur. Integration tests were conducted by replaying sample pcap files through the pipeline: packet capture, preprocessing, embedding generation, anomaly detection and scoring using scapy generated traffic to confirm correct data serialization and end to end operability.

6.2 Prototype System Benchmarks

Although formal system testing with labeled attack datasets was outside this study's scope, we measured key processing metrics during prototyping in Google Colab Pro+ and our hybrid cluster:

- **Embedding Throughput:** On Tesla T4 GPUs, our SecBERT embedding script processed approximately 500 packets/second, reflecting the combined overhead of tokenization and transformer inference within notebook environments.
- **Anomaly Scoring Latency:** The Isolation Forest and one-class classifier scripts applied to 768-dimensional vectors exhibited sub-5 ms per-batch scoring time for batches of 32 embeddings, as measured by Python's timeit module.

These benchmarks demonstrate the feasibility of real-time processing at modest traffic volumes (~30,000 pkt/min) in proof-of-concept deployments, though production workloads will necessitate horizontal scaling.

6.3 Simulated Anomaly Detection

To evaluate GUARD's detection logic under controlled conditions, we injected various synthetic anomaly patterns into benign traffic captures. These patterns

included port scans (sequential connection attempts across multiple ports), atypical packet-size distributions, and randomized session attributes. A port scan is a reconnaissance technique where an attacker systematically probes a range of network ports on a target host to identify open or vulnerable services. In our experiments, Scapy-generated port scans (e.g., 30 probes over 30 seconds) yielded elevated anomaly scores across all three detection approaches, consistently exceeding the predefined thresholds. Similarly, synthetic bursts of abnormally large packets (e.g., oversized payloads) and irregular session metadata combinations triggered alerts, demonstrating GUARD's capacity to identify both volumetric and behavioural anomalies. These semi-synthetic tests, conducted within Jupyter notebooks, provided qualitative validation of the models' responsiveness to diverse threat patterns while highlighting the need for richer labeled datasets for quantitative benchmarking.

6.4 Inference Distribution for SecBERT + Isolation Forest

During prototype runs, the SecBERT + Isolation Forest pipeline performed a sample inference on 800 network records drawn from mixed benign and injected anomaly captures. The observed prediction distribution was as follows:

- Anomalous (malicious) predictions: 214 (26.75%)
- Normal predictions: 586 (73.25%)

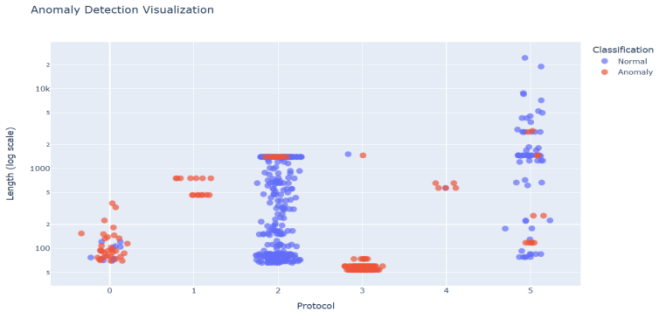

Figure 2: Isolation forest anomaly detection

These figures reflect raw model outputs in the absence of explicit ground-truth labels and thus cannot directly yield precision, recall, or F1 metrics. They do, however, indicate the model's sensitivity and the relative volume of flagged events in a typical operational batch. Detailed performance metrics will require labelled datasets and classification reports (e.g., via classification_report) once true labels are available.

6.5 PyTorch Tabular

GUARD incorporates the PyTorch Tabular ensemble model as one of its anomaly detection approaches. This model synergizes deep learning techniques with tabular data processing to identify anomalies in network traffic effectively [19].

Model Architecture and Training

The ensemble model consists of a simple feedforward neural network architecture featuring two hidden layers with 128 and 64 units respectively, ReLU activation functions, and dropout layers for regularization [20]. These design choices mitigate overfitting and ensure generalization to unseen data. Input features are processed through embedding layers, concatenated with SecBERT's [CLS] token output (768 dimensions), and passed through the network to predict the anomaly score.

Inference and Runtime Performance

At inference, PyTorch Tabular efficiently computes anomaly scores in of 32 records (McCaffrey, sub-5 ms per batch 2023), making it a viable candidate for near real-time intrusion detection applications. Each output score corresponds to the likelihood that a packet sequence deviates from established benign patterns.

The modular API and GPU acceleration capabilities of PyTorch Tabular align with GUARD's emphasis on scalable, production-grade systems. Furthermore, the model's native support for early stopping, mixed precision training, and MLflow integration proved essential during our deployment phase.

6.6 Data Limitations and Future Metrics

A key challenge was the lack of publicly available, labelled critical infrastructure traffic for quantitative evaluation. Our prototype relied on semi-synthetic injections rather than standardized benchmarks, limiting calculations of false-positive rates and detection accuracy. Future work will integrate datasets such as UNSW-NB15 or CIC-IDS2017 to compute precision, recall, and ROC-AUC metrics, and to validate GUARD's performance under varied threat scenarios.

7. Use Cases & Applications

The versatility of GUARD's design allows it to be applied across various industries and domains that require strong cybersecurity systems. Several key use cases and application areas can demonstrate how GUARD can be tailored to each context such as the following:

- **Defense Sector & Military Networks:** National defence organizations operate highly targeted and sensitive networks – such as command and control systems or intelligence networks. These networks face advanced persistent threats (APTs) and attacks from Nation-state actors. GUARD can serve as an advanced IDS in these environments by detecting stealthy intrusion patterns that evade conventional methods. For example, within an military network GUARD can be utilized to identify anomalous patterns in network traffic indicative of an insider threat exfiltrating classified data. The automated response can immediately trace and cut off the user's access and alert the appropriate security operations. With extensive, guided learning the system can adapt to new espionage tactics quickly with analyst feedback. Defense networks also benefit from GUARD's integration capability – it can work alongside existing defense-in-depth measures – such as tactical firewalls and endpoint monitors – and provide an extra analytical layer that is constantly improving and learning. Given the military's need for quick containment, GUARD's automated isolation of compromised systems can prevent an adversary from moving laterally across a secure network.
- **Government and Critical Infrastructure Protection:** Beyond defense, civilian government agencies and critical infrastructure operators (energy utilities, water treatment facilities, transportation systems, etc.) can leverage GUARD to protect against cyber attacks that could disrupt essential services. Critical Infrastructure (CI) systems often involve industrial control systems (ICS/SCADA) that cannot afford downtime. GUARD's real-time monitoring is well-suited to detect subtle signs of cyber-physical attacks – for instance, a hacker trying to subtly change sensor readings or control commands in a power grid.
- **Healthcare Sector:** Hospitals and healthcare networks have become prime targets for ransomware and other cyberattacks, given the critical nature of their operations, as well as the sensitive value of patient data. GUARD can be deployed in hospital networks to monitor medical IoT devices, health record systems, and general IT infrastructures with the system learning normal patterns of its devices and alert if a device starts to exhibit unusual behaviour.
- **Financial Sector:** Banks and financial institutions also deal with constant cyber threats, from fraud attempts to advanced intrusion by criminal groups. They typically have strong security operation centres (SOCs) that can benefit from AI augmentation. GUARD can analyze transaction networks, employee and customer behaviour on banking applications, and internal network traffic to spot anomalies indicative of fraud or breaches.

8. Conclusion & Future Work

Conclusion: The AI cybersecurity research group successfully employed

LLMs to utilize preprocessed recorded network data for the IDS. This challenge demonstrated that implementation was feasible; nonetheless, due to time constraints, a considerable amount of work was accomplished within the group. Upon project completion, the team successfully delineated the requirements for ongoing operations. Demonstrating that an AI Intrusion Detection System can identify anomalies in network traffic is crucial, as this was essential for the project's success.

Future Work: The future trajectory of this AI IDS is to augment and incorporate static source code analysis, offering insights into vulnerabilities and exploits present within the system. This will offer a more extensive network perspective, especially at the application layer, where several vulnerabilities exist. Furthermore, the AI IDS will incorporate additional features, including an improved Graphical User Interface (GUI). This backend processes .csv files for enhanced integration with a Business Intelligence (BI) analytics tool. This forthcoming endeavour will facilitate usability, implementation, and integration with pre-existing security environments, and finally, an examination of the algorithm's adaptability.

Acknowledgements: We gratefully acknowledge the Center for Cyber Security and Forensics Education (C²SAFE) for providing access to its core facilities, including the Ed Kaplan Family Institute for Innovation and Tech Entrepreneurship. Furthermore, we acknowledge funding support from Motorola Solutions Foundation.

References

[1] Liao, H. J., Lin, C. H. R., Lin, Y. C., Tung, K. Y. “Intrusion detection system: A comprehensive review”. Journal of Network and Computer Applications. 2013 Jan; 36(1):16–24. Available from: https://www.sciencedirect.com/science/article/pii/S1084804512001944.

[2] Markevych, M., Dawson, M. “A Review of Enhancing Intrusion Detection Systems for Cybersecurity Using Artificial Intelligence (AI)”. International Conference KNOWLEDGE-BASED ORGANIZATION. Sibiu: Vol. XXIX, No. 2023. DOI: 10.2478/kbo-2023-0000.

[3] Dawson, M., Bacius, R., Gouveia, L. B., & Vassilakos, A. “Understanding the challenge of cybersecurity in critical infrastructure sectors”. Land Forces Academy Review. March, 2021; 26(1): 69–75. https://sciendo.com/article/10.2478/raft-2021-0011.

[4] Kumar, G., Kumar, K., & Sachdeva, M. (2010). “The use of artificial intelligence based techniques for intrusion detection: a review”. Artificial Intelligence Review. December, 2010; 34(4), 369–387. https://link.springer.com/article/10.1007/s10462-010-9179-5.

[5] Sommer, R., & Paxson, V. “Outside the Closed World: On Using Machine Learning for Network Intrusion Detection”. In: IEEE Symposium on Security and Privacy. 2010. 305–316.

[6] Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. “Survey of intrusion detection systems: techniques, datasets and challenges”. Cybersecurity. 2019; 2(1), 20. https://cybersecurity.springeropen.com/articles/10.1186/s42400-019-0038-7.

[7] Kim, A., Park, M., & Lee, D. H. “AI-IDS: Application of Deep Learning to Real-Time Web Intrusion Detection”. IEEE Access. 2020; 8, 70245–70261. https://ieeexplore.ieee.org/document/9063416.

[8] Ferrag, M. A., Ndhlovu, M., Tihanyi, N., Cordeiro, L. C., Debbah, M., Lestable, T., & Thandi, N. S. “Revolutionizing cyber threat detection with large language models: A privacy-preserving BERT-based lightweight model for IoT/IIoT devices” [Preprint]. 2024. arXiv. https://arxiv.org/abs/2306.14263.

[9] Seliya, N., Abdollah Zadeh, A., & Khoshgoftaar, T. M. “A literature review on one-class classification and its potential applications in big data”. Journal of Big Data. 2021; 8, Article 122. https://doi.org/10.1186/s40537-021-00514-x.

[10] Sarhan, M., Kulatilleke, G., Lo, W. W., Layeghy, S., & Portmann, M. “DOC-NAD: A hybrid deep one-class classifier for network anomaly detection” [Preprint]. 2022. arXiv. https://arxiv.org/abs/2212.07558.

[11] Seliya, N., Abdollah Zadeh, A., & Khoshgoftaar, T. M. “A literature review on one-class classification and its potential applications in big data”. Journal of Big Data. 2021; 8, Article 122. https://doi.org/10.1186/s40537-021-00514-x.

[12] McCaffrey, J. D. “Anomaly detection for tabular data using a PyTorch transformer with numeric embedding: [Blog post]. April, 20, 2023. https://jamesmccaffrey.wordpress.com/2023/04/20/anomaly-detection-for-tabular-data-using-a-pytorch-transformer-with-numeric-embedding/.

[13] Warburton, D., & Vinberg, S. “The 2021 TLS telemetry report. F5 Labs”. October, 20. 2021. https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report?utm_source=chatgpt.com.

[14] Daoud, L., & Huen, H. “Performance study of software-based encrypting data at rest”. In: Proceedings of the 37th International Conference on Computers and Their Applications (EPiC Series in Computing, Vol. 82: 122–130). 2022. EasyChair. Retrieved from https://easychair.org/publications/paper/GHB6/open?utm_source=chatg.

[15] Singh, J., Rani, S., & Kumar, V. “Role-based access control (RBAC) enabled secure and efficient data processing framework for IoT networks”. International Journal of Communication Networks and Information Security. 2024. 16(2): 19-32. https://ijcnis.org/.

[16] Menashe, S., Peles, O., Hollander, O., & Yavnieli, U. “Machine learning bug bonanza – exploiting ML clients and “safe” model formats”. JFrog. December, 4, 2024. https://jfrog.com/blog/machine-learning-bug-bonanza-exploiting-ml-clients-and-safe-model-formats/.

[17] Hassan, N. “Secure your machine learning models with these MLSecOps tips”. TechTarget. January 30, 2024. https://www.techtarget.com/searchITOperations/tip/Secure-your-machine-learning-models-with-these-MLSecOps-tips.

[18] Fortra. “2022 SIEM report” (Technical Report). Fortra. 2022. Retrieved from https://static.fortra.com/core-security/pdfs/reports/cs-siem-report-2022.pdf

[19] Jamesdmccaffrey. “Anomaly Detection for Tabular Data Using a PyTorch Transformer with Numeric Embedding”. James D. McCaffrey. May, 8, 2023. https://jamesmccaffrey.wordpress.com/2023/04/20/anomaly-detection-for-tabular-data-using-a-pytorch-transformer-with-numeric-embedding/

[20] Manujosephv. (n.d.). “GitHub - manujosephv/pytorch_tabular: A standard framework for modelling Deep Learning Models for tabular data.” GitHub. https://github.com/manujosephv/pytorch_tabular