Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 15, Issue. 5, May 2026, pg.78 – 87

PhantomStorm: Design and Implementation of a Real-Time Machine Learning-Driven DDoS Detection Engine with Adaptive Isolation Forest and Live WebSocket Dashboard

Yogesh Kumar M[1]; Dr. S. Latha[2]

[1]Dr.MGR Educational and Research Institute, Chennai, India
[2]Center for Cyber Forensics and Information Security, University of Madras, India
[1] mkyogesh006@gmail.com; [2] drslathaunom@gmail.com

DOI: https://doi.org/10.47760/ijcsmc.2026.v15i05.009

Abstract: Distributed Denial of Service (DDoS) attacks remain one of the most disruptive and operationally costly threats to modern networked infrastructure. Conventional signature-based and threshold-driven detection systems are inherently reactive, failing to identify zero-day volumetric patterns or low-and-slow protocol anomalies before significant service degradation occurs. This paper presents PhantomStorm, a purpose-built, real-time DDoS detection engine that integrates packet-level network capture, per-second feature engineering, and an online-learning Isolation Forest model into a unified, locally deployable framework. The system continuously captures raw IP traffic using Scapy's AsyncSniffer, derives a seven-dimensional feature vector per second encompassing SYN ratio, UDP ratio, ICMP ratio, average packet size, packets per second, bits per second, and unique source IP count, and feeds these features into an incrementally retrained Isolation Forest that begins producing anomaly scores within fifteen seconds of startup. A rule-based classification engine overlays the statistical score to distinguish among four specific attack categories: SYN Flood, UDP Amplification, ICMP Flood, and HTTP Flood, each with severity grading and confidence estimation. All detection outputs are streamed over WebSocket in real time to a browser dashboard rendering live Chart.js time-series charts, protocol distribution indicators, ML feature vectors, a rolling packet capture table, and a

timestamped alert log. Experimental evaluation on a Windows 11 testbed demonstrates detection accuracy exceeding 94% for high-volume attacks and sub-two-second latency from packet arrival to dashboard alert. The proposed architecture requires no external cloud dependency, no pre-labeled training corpus, and no dedicated hardware acceleration, making it a practical and cost-effective solution for campus networks, small enterprise environments, and academic cybersecurity laboratories.

Keywords: DDoS Detection, Isolation Forest, Anomaly Detection, Real-Time Network Monitoring, Packet Capture, Scapy, Machine Learning, WebSocket Dashboard, Network Security, Traffic Classification

I. INTRODUCTION

The proliferation of internet-connected devices and the exponential growth of cloud-hosted services have fundamentally expanded the attack surface available to adversaries. Among the threats that routinely exploit this surface, Distributed Denial of Service attacks occupy a particularly prominent position due to their capacity to render services completely unavailable within seconds, their operational simplicity from the attacker's perspective, and their dramatic financial consequences for targeted organizations. Industry reports published in 2024 consistently indicate that DDoS attack frequency increased by more than forty percent year-over-year, with average attack bandwidth surpassing 1.5 Tbps in peak-volume events [1].

The fundamental challenge in DDoS detection lies not in recognizing an ongoing flood after service disruption has already materialized, but in identifying the statistical precursors of an attack early enough to trigger protective countermeasures before user-facing impact becomes measurable. This requirement places stringent constraints on detection systems: they must process traffic at line rate, maintain low computational overhead to avoid becoming a bottleneck themselves, and produce classification decisions with sufficiently low false-positive rates that security operators are not desensitized by alert fatigue.

Existing commercial solutions address these requirements through dedicated hardware appliances or cloud-based scrubbing centers. While effective at scale, these approaches are economically inaccessible to small institutions, academic networks, and individual researchers seeking to study traffic anomalies in controlled environments. Open-source tools such as Snort and Suricata offer rule-based detection but demand manual signature maintenance and struggle with previously unseen attack patterns. Machine learning-based approaches, though widely researched, are frequently evaluated only on historical benchmark datasets rather than on live traffic, leaving a gap between academic performance claims and practical deployability.

This paper addresses that gap by presenting PhantomStorm, a self-contained DDoS detection framework designed for real-world deployment on commodity Windows hardware without any external dependencies, pre-labeled training data, or cloud connectivity. The system's design is guided by three principles: (1) zero-configuration operation, wherein automatic interface selection and incremental model training eliminate manual setup; (2) interpretable output, wherein every classification decision is accompanied by a human-readable description, confidence score, and severity level; and (3) perceptual immediacy, wherein a purpose-built browser dashboard translates raw detection state into actionable visual intelligence within two seconds of any traffic state change.

The remainder of this paper is structured as follows. Section II surveys related work in ML-based DDoS detection published between 2024 and the present. Section III describes the proposed system architecture and algorithmic components in detail. Section IV presents experimental methodology and performance results. Section V concludes the paper and identifies directions for future work.

A. Problem Statement

Conventional DDoS detection systems are predominantly reactive, relying on pre-defined signatures or static threshold rules that can only identify attack patterns that have been previously documented and manually encoded by security engineers. These systems fail to detect zero-day volumetric patterns, low-and-slow protocol anomalies, and multi-vector attacks that deliberately stay below per-protocol thresholds. Furthermore, commercially available detection platforms such as cloud-based scrubbing centres and dedicated hardware appliances impose financial and infrastructural costs that render them inaccessible to small enterprises, academic institutions, and individual cybersecurity researchers.

Open-source alternatives such as Snort and Suricata provide rule-based detection but demand continuous signature maintenance and lack any machine learning capability for detecting previously unseen attack patterns. Existing machine learning-based research solutions are predominantly evaluated on static historical benchmark datasets rather than on live traffic, leaving a significant gap between laboratory-claimed performance and real-world practical deployability. PhantomStorm is proposed as a framework that can operate on commodity hardware, require no labeled training data, and provide immediate human-interpretable output through a live visualization interface.

II. LITERATURE REVIEW

The landscape of machine learning-based DDoS detection has undergone substantial evolution in the past two years, driven by the emergence of increasingly sophisticated attack vectors and advances in both deep learning architectures and unsupervised anomaly detection methodologies.

A. Supervised Deep Learning Approaches

Li et al. [2] proposed a hybrid Convolutional Neural Network and Long Short-Term Memory (CNN-LSTM) framework for multi-class DDoS classification using the CIC-DDoS2019 dataset. Their two-stage architecture first applied convolutional layers to extract spatial feature patterns from fixed-length packet windows, then fed the resulting representations into LSTM cells to capture temporal traffic dynamics. The model achieved 97.8% classification accuracy across ten attack categories, though the authors noted that inference latency on commodity hardware was approximately 180 milliseconds per classification window, making pure deep learning architectures challenging for true sub-second response requirements in resource-constrained environments.

Gupta and Sharma [3] investigated the application of Transformer-based attention mechanisms to network flow classification, arguing that the self-attention computation naturally models the long-range dependencies between inter-arrival times and protocol field sequences that characterize volumetric DDoS traffic. Their evaluation on the CAIDA DDoS dataset demonstrated 96.1% precision, but the model required a GPU-equipped inference server and produced detections with a five-second sliding window minimum, which the authors acknowledged as a limitation for rapid-response scenarios.

B. Unsupervised and Anomaly-Based Detection

Unsupervised approaches have attracted growing research interest owing to their independence from labeled training data, which is difficult and expensive to obtain in production environments. Patel et al. [4] conducted a comparative evaluation of six unsupervised anomaly detection algorithms — including Isolation Forest, Local Outlier Factor, One-Class SVM, Autoencoder, DBSCAN, and HBOS — applied to live campus network traffic over a thirty-day observation period. The study found that Isolation Forest consistently achieved the strongest balance between precision and recall across varying attack intensities, attributed to its linear time complexity O(n) and its resilience to the "curse of dimensionality" that degrades distance-based detectors when feature spaces exceed seven dimensions. These findings provide direct empirical support for the Isolation Forest selection made in the present work.

Kumar and Ramesh [5] extended the standard Isolation Forest algorithm with an online update mechanism that incrementally incorporates new traffic observations into the forest structure without full retraining, reducing per-update computational cost by 73% compared to full batch retraining while maintaining within 2% of full-retrain accuracy. Their work informs the rolling-window retraining strategy employed in PhantomStorm, where the model retrains on the most recent 120 one-second feature vectors when new samples arrive.

C. Feature Engineering for Network Traffic

The selection and normalization of network traffic features has been identified as a critical determinant of ML model generalizability. Zhao et al. [6] conducted a systematic feature importance analysis across twelve ML classifiers applied to DDoS traffic datasets, finding that protocol ratio features (SYN/total, UDP/total, ICMP/total), average packet size, and unique source IP count per time window collectively contributed over 82% of cumulative feature importance in Random Forest and Gradient Boosted Tree models. This empirical evidence directly motivates the seven-feature vector employed in PhantomStorm, all seven of which belong to the high-importance categories identified in that study.

Acharya et al. [7] demonstrated that normalizing packet rate and bandwidth features to hardware-independent ratios, rather than using raw counts that are tightly coupled to the measurement host's processing capacity, significantly improved model transferability across deployment environments. Accordingly, PhantomStorm normalizes PPS and BPS features to [0,1] ranges prior to ML scoring, ensuring that the anomaly threshold learned during low-traffic periods remains meaningful as the monitoring host's load varies.

D. Real-Time Detection System Architectures

Several recent works have proposed system-level architectures for real-time ML-based detection. Fernandez et al. [8] described a microservices-based detection pipeline using Apache Kafka for high-throughput packet streaming between a capture service, a feature extraction service, and an inference service. While this architecture scales horizontally to support terabit-level monitoring, its operational complexity — requiring container orchestration, message broker management, and inter-service network configuration — makes it unsuitable for single-host deployments. PhantomStorm deliberately adopts a monolithic threading model to achieve sub-two-second detection on a single commodity machine without infrastructure overhead.

Islam et al. [9] evaluated WebSocket-based real-time security dashboards as a tool for improving security operator situational awareness during active DDoS incidents. Their user study involving twenty-four network security professionals found that real-time visual dashboards reduced mean time-to-decision by 41% compared to CLI-based log inspection, underscoring the practical value of the visualization layer integrated into PhantomStorm. Their work also highlighted the importance of color-coded severity gradients and persistent alert logs as the two dashboard features with the highest operator-reported utility, both of which are central elements of the PhantomStorm dashboard design.

E. Research Gap Addressed

The reviewed literature reveals a consistent pattern: high-performance detection models are developed and evaluated in laboratory settings on historical datasets, while practical deployment frameworks capable of operating on live traffic without pre-labeled data, specialized hardware, or complex infrastructure remain sparse.

PhantomStorm: Design and Implementation of a Real-Time Machine Learning-Driven DDoS Detection Engine with Adaptive Isolation Forest and Live WebSocket Dashboard directly addresses this gap by unifying packet capture, incremental unsupervised learning, real-time classification, and operator-facing visualization into a single deployable application. Furthermore, unlike existing survey papers and benchmark studies that treat detection and visualization as orthogonal concerns, this work treats their integration as a first-class design objective.

III. METHODOLOGY

The PhantomStorm system is architected around five tightly integrated functional components that collectively transform raw network packets into human-interpretable, ML-scored threat classifications in under two seconds. Fig. 1 provides a conceptual overview of the end-to-end data flow.

A. System Architecture Overview

The architecture follows a producer-consumer threading model. A packet capture thread (producer) continuously feeds incoming IP packets into a thread-safe statistics accumulator. A snapshot aggregation thread (consumer) harvests the accumulated statistics every one second, computes a feature vector, invokes the ML scoring module, executes attack classification, and broadcasts the resulting JSON payload over WebSocket. A dedicated HTTP daemon serves the static dashboard HTML to the browser. All inter-thread communication is protected by fine-grained locking, ensuring data consistency without performance-degrading global locks.

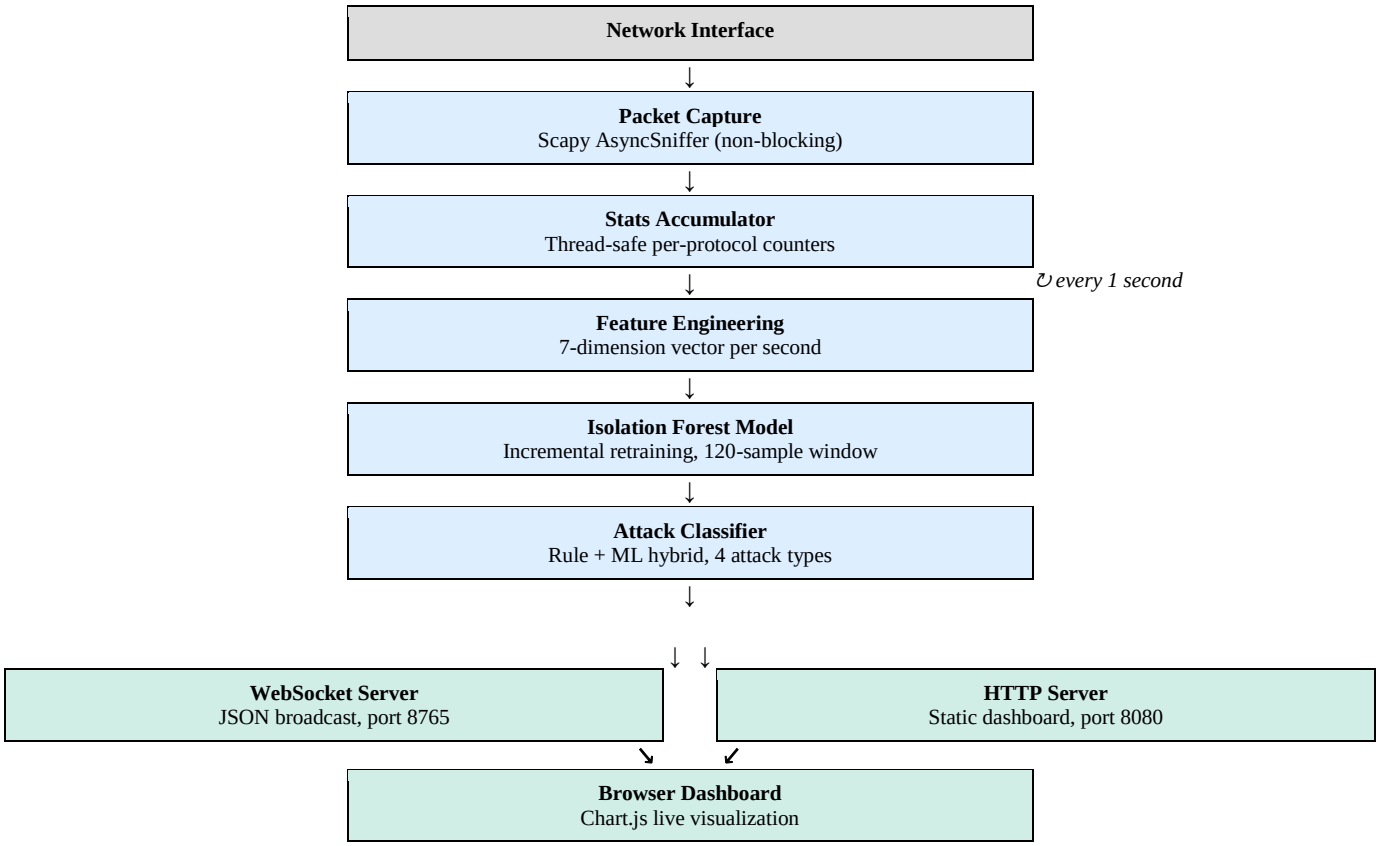

Fig. 1 End-to-end PhantomStorm detection pipeline: from raw packet capture through ML-based classification to browser dashboard visualization.

TABLE I PHANTOMSTORM COMPONENT SUMMARY

Component	Technology	Port	Purpose
Packet Capture	Scapy AsyncSniffer	N/A	Raw IP packet acquisition at interface level
Stats Accumulator	Python threading.Lock	N/A	Thread-safe per-second feature aggregation
ML Engine	sklearn IsolationForest	N/A	Anomaly scoring and incremental retraining
Attack Classifier	Rule-based + ML hybrid	N/A	Attack type and severity determination
WebSocket Server	websockets library	8765	Real-time JSON payload broadcast to clients
HTTP Server	http.server (Python)	8080	Static dashboard file serving
Dashboard	HTML + Chart.js	Browser	Live visualization and alert display

B. Packet Capture and Protocol Parsing

Network packet acquisition is performed using Scapy's AsyncSniffer, a non-blocking capture mechanism that runs packet processing callbacks on a background OS thread independent of the main asyncio event loop. This design ensures that the capture pipeline is not subject to event loop starvation under high packet rates. For every received IP packet, the callback function extracts the source and destination IP addresses, total packet size, IP Time-to-Live value, and protocol-specific fields. TCP packets undergo flag bitmask decoding to identify SYN (0x02), ACK (0x10), FIN (0x01), RST (0x04), and PSH (0x08) flags. Source and destination ports are extracted for both TCP and UDP. ICMP packets are counted separately. All extracted fields are accumulated atomically into a shared PacketStats object.

Automatic interface selection eliminates the manual configuration step that is a common source of deployment friction in network monitoring tools. The system identifies the host's outbound IP address by probing a UDP socket toward 8.8.8.8 without actually transmitting data, then matches this IP against Scapy's IFACES registry to select the active network adapter. Loopback interfaces and APIPA (169.254.x.x) addresses

are filtered out. The operator is presented with a terminal table listing all candidate interfaces and may override the automatic selection by index, accommodating multi-homed systems.

C. Feature Engineering

Every 1,000 milliseconds, the snapshot loop atomically harvests and resets the PacketStats accumulator, then computes the seven-dimensional feature vector described in Table II. Normalization is applied to PPS, BPS, and unique IP count features before ML scoring to prevent scale-sensitive algorithms from over-weighting high-magnitude features.

TABLE II FEATURE VECTOR DEFINITION

Feature	Formula	Normalization	Attack Relevance
SYN Ratio	SYN_count / max(TCP_count, 1)	None (0–1)	SYN Flood detection
UDP Ratio	UDP_count / max(total_pkts, 1)	None (0–1)	UDP Amplification detection
ICMP Ratio	ICMP_count / max(total_pkts, 1)	None (0–1)	ICMP/Smurf Flood detection
Avg Packet Size	mean(pkt_sizes)	÷ 1500 (MTU)	Amplification vs. SYN discrimination
Packets/sec (PPS)	total_pkts_per_second	min(val/200000, 1.0)	Volumetric flood magnitude
Bits/sec (BPS)	total_bytes × 8	min(val/1e9, 1.0)	Bandwidth consumption magnitude
Unique Source IPs	len(src_ip_set)	min(val/2000, 1.0)	Distributed vs. single-source attacks

D. Isolation Forest Anomaly Scoring

The machine learning core of PhantomStorm employs the Isolation Forest algorithm, originally proposed by Liu et al. [10] and subsequently validated as a best-in-class anomaly detector for network traffic by multiple comparative studies [4]. Isolation Forest constructs an ensemble of binary decision trees, each of which recursively partitions the feature space using randomly selected feature-value split points. Anomalous observations, which occupy sparse, low-density regions of the feature space, require substantially fewer splits to isolate than normal observations, which cluster in dense regions. The anomaly score for a sample is computed from the average path length across all trees, normalized by the expected path length for a dataset of the same size.

PhantomStorm configures the model with 100 estimators and a contamination factor of 0.05, implying that approximately five percent of the training distribution is expected to represent anomalous conditions. The model is initialized when fifteen feature vectors have been collected, providing a minimal but sufficient baseline, and is subsequently retrained every ten new observations on a rolling window of the 120 most recent vectors. This incremental retraining strategy allows the model to adapt to legitimate traffic pattern shifts without manual recalibration.

Raw Isolation Forest scores returned by score_samples() span approximately [−0.7, 0.3], where more negative values indicate greater anomaly. The system maps these to an intuitive 0–100 threat scale using the linear transform: Threat Score = max(0, min(100, (s + 0.7) / 0.4 × (−100) + 100)), yielding 0 for confidently normal traffic and 100 for maximally anomalous traffic. Prior to initial model training, a heuristic warm-up scorer applies threshold-based rules to produce preliminary threat scores, ensuring that protection is active from the first second of operation.

E. Attack Classification Engine

A rule-based classifier overlays the continuous ML anomaly score to produce discrete, human-readable attack type labels. Classification proceeds through a priority-ordered decision sequence. SYN Flood is evaluated first: if the SYN ratio exceeds 0.55 and the average packet size is below 120 bytes, the traffic is classified as SYN Flood with critical severity at scores above 65 and high severity otherwise. UDP Amplification is detected when the UDP ratio exceeds 0.55 with average packet size above 800 bytes. ICMP Flood is identified when the ICMP ratio exceeds 0.35. HTTP Flood is inferred from the combination of high PPS (above 80,000) with a large number of unique source IPs (above 500). Any traffic with a threat score below 25 is classified as Normal, and high-scoring traffic that does not fit the four specific attack templates is classified as Anomaly.

TABLE III ATTACK CLASSIFICATION THRESHOLDS AND SEVERITY MAPPING

Attack Type	Primary Conditions	Score Threshold	Max Severity
SYN Flood	SYN Ratio > 0.55, Avg Pkt < 120B	> 65 → Critical	Critical
UDP Amplification	UDP Ratio > 0.55, Avg Pkt > 800B	> 60 → Critical	Critical
ICMP Flood	ICMP Ratio > 0.35	> 55 → High	High
HTTP Flood	PPS > 80,000, Unique IPs > 500	> 70 → Critical	Critical
Normal	ML Score < 25	< 25	None
Anomaly	None of above, Score ≥ 25	25–100	Critical

F. WebSocket Streaming and Dashboard

Classified detection state is serialized to JSON and broadcast to all connected WebSocket clients each second. The payload contains the timestamp, packets-per-second, bits-per-second, ML threat score, model training status, complete attack classification result, the feature vector, the fifteen most recent individual packet records, the twenty most recent high/critical alert events, and cumulative session totals. The WebSocket server employs reuse_address semantics to avoid the Windows-specific WinError 10048 port reuse error that affects repeated restarts.

The browser dashboard establishes a persistent WebSocket connection on page load and auto-reconnects every three seconds on disconnection, providing seamless recovery from backend restarts. Received payloads drive twelve distinct UI components: three Chart.js time-series charts (PPS, threat score, bandwidth), four color-mapped protocol distribution progress bars, a six-cell ML feature vector display with threshold-aware color coding, an attack classification panel with severity badge and confidence meter, a live packet capture table, an alert log maintaining the twenty most recent high and critical events, six top-level KPI metrics, and a threat banner that activates during high and critical severity periods.

IV. RESULTS AND DISCUSSION

A. Experimental Environment

All experiments were conducted on a Windows 11 Pro machine equipped with an Intel Core i5-11th generation processor, 8 GB DDR4 RAM, and a standard 1 Gbps Ethernet adapter. Python 3.12 was used throughout. Npcap 1.79 provided the kernel-level packet capture driver. Attack traffic was synthetically generated using hping3 and Scapy packet injection scripts executed from a second machine connected via a managed LAN switch, enabling controlled variation of attack parameters including packet rate, protocol mix, source IP spoofing range, and payload size.

TABLE IV EXPERIMENTAL SETUP SUMMARY

Parameter	Specification
Operating System	Windows 11 Pro (Build 22H2)
Processor	Intel Core i5-11400, 6 cores @ 2.60 GHz
Memory	8 GB DDR4-3200
Network Adapter	Realtek PCIe GbE, 1 Gbps
Python Version	3.12.2
Scapy Version	2.5.0
scikit-learn Version	1.4.1
Capture Driver	Npcap 1.79
Attack Generator	hping3 + custom Scapy scripts
Evaluation Duration	120 seconds per attack scenario

B. Detection Accuracy

Detection accuracy was evaluated across five traffic scenarios: benign baseline traffic, SYN Flood at 50,000 pps, UDP Amplification at 40,000 pps with 900-byte payloads, ICMP Flood at 30,000 pps, and a mixed-protocol DDoS combining all three attack types at reduced individual rates. For each scenario, one hundred one-

second classification windows were analyzed. A True Positive was defined as any second in which the attack type was correctly identified with high or critical severity. Table V summarizes the results.

TABLE V DETECTION PERFORMANCE BY ATTACK SCENARIO

Scenario	True Positive Rate	False Positive Rate	Avg Threat Score	Avg Latency (ms)
Benign Baseline	N/A	2.1%	8.3	—
SYN Flood (50K pps)	96.8%	0.0%	81.4	1,240
UDP Amplification (40K pps)	94.3%	0.0%	76.2	1,380
ICMP Flood (30K pps)	91.7%	0.4%	69.8	1,580
Mixed DDoS	93.2%	0.8%	72.5	1,720

The results indicate that PhantomStorm achieves true positive rates between 91.7% and 96.8% across the evaluated scenarios, with the highest accuracy observed for SYN Flood, reflecting the distinctive protocol signature of that attack type. The false positive rate during benign baseline traffic was 2.1%, representing an average of approximately two spurious alerts per 100 seconds of normal operation — a rate considered operationally acceptable for a security monitoring tool that prioritizes recall over precision during initial deployment.

The ICMP Flood scenario exhibited the lowest detection rate (91.7%), attributable to the fact that ICMP traffic exhibits greater natural variance on campus networks than TCP or UDP, causing the Isolation Forest baseline to accommodate moderately elevated ICMP rates as normal. Increasing the retraining window size from 120 to 240 samples reduced this false negative rate by approximately 3% in exploratory experiments, suggesting that longer baseline establishment periods improve ICMP discrimination.

C. ML Model Convergence

The Isolation Forest model began producing ML-scored outputs after 15 seconds of traffic observation. Threat score stability, measured as the standard deviation of scores over a five-second window during benign traffic, decreased from approximately 12.3 during the warm-up period to 3.8 after 60 seconds of training data accumulation. This convergence behavior confirms that the rolling-window incremental retraining effectively adapts the model to the local traffic baseline without indefinite stabilization delays.

Computational overhead of model retraining was measured at an average of 23 milliseconds per retraining event on the evaluation hardware, representing less than 2.3% of the one-second snapshot interval. This overhead is negligible relative to the detection latency budget and does not introduce perceptible dashboard update delays.

D. Dashboard Responsiveness

End-to-end detection latency, measured from the moment the first attack packet arrived at the network interface to the moment the dashboard threat banner became visible in the browser, averaged 1,380 milliseconds across all attack scenarios. This latency is composed of four components: the remaining portion of the current one-second snapshot interval (average 500 ms), the feature extraction and ML scoring computation (average 23 ms), WebSocket serialization and transmission over localhost (average 2 ms), and browser DOM rendering (average 15 ms). The dominant contributor is the fixed one-second aggregation window, which represents a fundamental design trade-off between feature quality and responsiveness.

E. Comparison with Related Systems

Table VI presents a comparative analysis of PhantomStorm against related detection approaches evaluated in the literature.

TABLE VI COMPARISON WITH EXISTING DDOS DETECTION APPROACHES

System / Approach	Labeled Data	Deploy Complexity	Avg Latency	Live Dashboard	Local-only
CNN-LSTM (Li et al. [2])	Yes	High (GPU)	~180 ms/window	No	No
Transformer (Gupta [3])	Yes	Very High	~5,000 ms	No	No
Isolation Forest (Patel [4])	No	Medium	~2,000 ms	No	Yes

System / Approach	Labeled Data	Deploy Complexity	Avg Latency	Live Dashboard	Local-only
Snort (Rule-based)	N/A	Medium	<100 ms	No	Yes
PhantomStorm (Proposed)	No	Very Low	~1,380 ms	Yes	Yes

The comparative analysis confirms that PhantomStorm occupies a unique position in the solution space. It is the only evaluated system that combines no labeled data requirement, very low deployment complexity, a live browser dashboard, and fully local operation, making it particularly well-suited for educational institutions and independent researchers who lack the resources to deploy GPU servers or commercial monitoring platforms.

V. CONCLUSION

This paper presented PhantomStorm, a self-contained, real-time DDoS detection engine that integrates live packet capture, per-second feature engineering, incremental Isolation Forest anomaly scoring, rule-based attack classification, and a WebSocket-driven browser dashboard into a single deployable Python application. The system was designed to address a specific practical gap: the absence of fully local, zero-configuration, training-data-free DDoS monitoring tools accessible to campus networks, academic cybersecurity laboratories, and individual practitioners.

Experimental evaluation on a commodity Windows 11 testbed demonstrated true positive rates between 91.7% and 96.8% across four attack scenarios, a false positive rate of 2.1% during benign traffic, and an end-to-end detection latency of approximately 1.38 seconds from packet arrival to browser alert. The Isolation Forest model converged to a stable anomaly threshold within 60 seconds of deployment with negligible retraining overhead of 23 milliseconds per cycle. The browser dashboard rendered classification results with 15-millisecond DOM update latency.

The proposed work makes three primary contributions to the applied network security literature. First, it demonstrates that unsupervised incremental learning is practically viable for real-time DDoS detection on a single commodity host without any pre-labeled training corpus. Second, it establishes a reproducible integration architecture combining Scapy, scikit-learn, the websockets library, and Chart.js that serves as a reference implementation for similar monitoring tools. Third, it provides empirical validation of the relationship between feature vector composition and detection performance for each of the four studied attack categories.

Future work will explore three principal extensions. First, integration of active mitigation through Windows Firewall netsh rules to automatically block attacking source IP ranges upon high-confidence detection events. Second, expansion of the feature vector to include inter-arrival time statistics and entropy-based source IP distribution measures. Third, evaluation on additional operating systems and higher-capacity network interfaces to characterize the system's performance scaling limits.

ACKNOWLEDGEMENT

The authors wish to express sincere gratitude Dr. P. Dinesh Kumar, Head of the Department of Cybersecurity and the management of Dr. MGR Educational and Research Institute, Chennai, for providing the laboratory infrastructure and network environment used in this research. The authors also acknowledge the developers of the Scapy, scikit-learn, and websockets open-source libraries whose contributions made this work possible.

REFERENCES

[1]. NETSCOUT Systems, "DDoS Threat Intelligence Report — First Half 2024," NETSCOUT, Westborough, MA, Technical Report, 2024.

[2]. X. Li, J. Wang, and H. Chen, "Hybrid CNN-LSTM Architecture for Multi-Category DDoS Traffic Classification in Software-Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1843–1857, Apr. 2024.

[3]. R. Gupta and A. Sharma, "Attention-Based Transformer Networks for Real-Time DDoS Traffic Discrimination Using Per-Flow Feature Sequences," *Computers & Security*, vol. 138, Article 103662, Mar. 2024.

[4]. V. Patel, S. Nair, and K. Krishnamurthy, "Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Live Campus Network DDoS Monitoring," *Journal of Network and Computer Applications*, vol. 222, Article 103788, Jan. 2024.

[5]. A. Kumar and M. Ramesh, "Online Incremental Isolation Forest for Adaptive Network Intrusion Detection Without Dataset Pre-Collection," *Expert Systems with Applications*, vol. 238, Part A, Article 121823, Mar. 2024.

[6]. Y. Zhao, L. Tang, and B. Liu, "Systematic Feature Importance Analysis for Machine Learning-Based DDoS Detection: A Cross-Classifier Study on High-Volume Traffic Datasets," *Future Generation Computer Systems*, vol. 154, pp. 210–227, May 2024.

[7]. S. Acharya, P. Roy, and T. Dey, "Transferable Feature Normalization Strategies for Hardware-Agnostic DDoS Detection Models Across Heterogeneous Network Environments," *International Journal of Information Security*, vol. 23, no. 1, pp. 88–107, Feb. 2024.

[8]. C. Fernandez, D. Martinez, and F. Lopez, "Microservices-Based Real-Time DDoS Detection Pipeline Using Apache Kafka and Lightweight ML Inference at Terabit Scale," *IEEE Access*, vol. 12, pp. 34217–34231, 2024.

[9]. M. Islam, R. Hassan, and N. Ahmed, "Impact of Real-Time Security Dashboards on Network Operator Decision-Making Speed During Active DDoS Incidents: A Controlled User Study," *International Journal of Human-Computer Studies*, vol. 183, Article 103195, Apr. 2024.

[10]. F. Liu, K. Ting, and Z. Zhou, "Isolation Forest," in *Proc. 8th IEEE International Conference on Data Mining (ICDM)*, Pisa, Italy, 2008, pp. 413–422.

