JAIMT
JOURNAL OF ARTIFICIAL INTELLIGENCE AND MODERN TECHNOLOGY
A Peer Reviewed (Refereed) International Journal
ISSN:1595-6261 VOLUME 7, ISSUE 1 (MAY) 2026

EVALUATING EBPF-BASED TELEMETRY FOR AI-DRIVEN ANOMALY DETECTION IN ENTERPRISE NETWORKS

Aneke Chikezie Samuel [1]; Michael Paul Esu [2] & Emmanuel Udoiwod [3]
[1,2,3] Tetfund Centre of Excellence in Computational Intelligence Research, Akwa Ibom State Polytechnic, Ikot Osurua, Ikot Ekpene
EMAIL: anekesamuel@uniuyo.edu.ng [1], michaelesu@uniuyo.edu.ng [2], emmanueludoiwod@uniuyo.edu.ng [3]

D.O.I.: https://doi.org/10.5281/zenodo.22095077

ARTICLE INFORMATION

Received: 30st JANUARY, 2026
Accepted: 26th FEB, 2026
Published: 31th MARCH, 2026

KEYWORDS: eBPF; XDP; network telemetry; anomaly detection; artificial intelligence; machine learning; intrusion detection; enterprise networks; explainable AI; SOC.
JOURNAL URL: https://ijois.com/jaimt/index.php

PUBLISHER: Empirical Studies and Communication (A Research Center)
Website: www.cescd.com.ng

ABSTRACT

Enterprise networks now carry traffic volumes and encryption levels that traditional monitoring tools were never designed for, forcing security teams towards kernel-level telemetry and machine-learning-based detection. This article evaluates the extended Berkeley Packet Filter (eBPF) as a telemetry substrate for artificial-intelligence-driven network anomaly detection, combining an expanded review of current literature with an original, testbed-based empirical study. The review examines eBPF's architecture, its established role in network observability, the machine-learning and deep-learning methods used for anomaly detection, how the two are integrated in practice, the explainability barriers to security-operations-centre (SOC) adoption, and the performance trade-offs reported to date. The empirical study then measures detection accuracy, end-to-end latency and host CPU overhead across three telemetry-collection methods and five detection models on a controlled 10 GbE testbed. A hybrid random-forest-plus-LSTM model achieved the highest detection accuracy (98.6%) and F1-score (98.0%), while eBPF/XDP-based collection sustained under 4.1% CPU overhead at 10 Gbps against roughly 30% and 39% for agent-based and libpcap-based collection respectively. The results support the case for eBPF as an efficient telemetry layer for AI-driven detection while highlighting that model accuracy gains carry real latency and complexity costs that the wider literature rarely reports alongside overhead figures. The article closes with limitations and concrete directions for further empirical work.

INTRODUCTION

Enterprise networks have become too large and too fast for a human analyst, or even a static rule set, to watch in real time. A mid-sized enterprise now pushes tens of gigabits of east-west traffic between containers, virtual machines and cloud services every second, and a growing share of that traffic is encrypted, ephemeral and generated by machines rather than people (Eunomia, 2025). Traditional visibility tools were not built for this world. Packet mirroring, SPAN ports and userspace agents based on libpcap all sit downstream of the kernel's networking stack, which means they either miss events under load or tax the very hosts they are meant to protect (Gregg, 2020). Something had to give, and over the last five years that something has been the kernel itself.

The extended Berkeley Packet Filter, or eBPF, lets verified, sandboxed programs run directly inside the Linux kernel, hooking into network interfaces, system calls and tracepoints without patching source code or loading a kernel module (Gregg, 2019). What began as a niche packet-filtering mechanism has turned into the backbone of modern observability: Cilium, Falco, Tracee, Pixie and Cloudflare's DDoS defences all lean on eBPF to see what is happening on a host at line rate and with a fraction of the overhead of older instrumentation (Red Hat, 2023; eBPF Foundation, 2025). Meta reports cutting CPU cycles for internal tracing by up to a fifth using eBPF-based tooling, Datadog cites a 35% reduction in monitoring CPU usage, and Polar Signals halved its internal network-monitoring costs, figures that would have sounded implausible for any userspace agent a decade ago (eBPF Foundation, 2025).

At the same time, security teams have been reaching for artificial intelligence to cope with alert volumes that human analysts cannot triage unaided. Supervised classifiers, recurrent networks and, more recently, ensemble and hybrid architectures are now routinely applied to network flow data to flag distributed denial-of-service attacks, port scans, lateral movement and other anomalies (Anand et al., 2025; Farasat et al., 2024). The appeal is obvious: a well-trained model can spot a subtle shift in traffic behaviour long before a signature-based system would recognise it as an attack. The catch is just as obvious. An AI model is only as good as the telemetry it is fed, and if that telemetry arrives late, incomplete, or is so expensive to collect that it degrades production systems, the accuracy of the model on paper counts for very little on a live network. John et al. (2026)

This is the gap this article sits in. eBPF and AI-driven anomaly detection are each well studied on their own, but the literature that evaluates them together, as a coupled pipeline, under realistic enterprise-style load, with both detection performance and system overhead measured side by side, is thinner than it should be given how often the pairing is proposed. Several recent papers combine the two (Ben-Yair, Rogovoy and Zaidenberg, 2019; Farasat et al., 2024; Tolay, 2025), but most report either the security outcome or the performance outcome, rarely both with the same rigour.

This article addresses that gap directly. Section 2 reviews the relevant literature across seven themes: eBPF's architecture and telemetry mechanisms, its established use in network observability, machine-learning and deep-learning approaches to anomaly detection, how

eBPF and AI are integrated in practice, explainability and SOC trust, performance and scalability trade-offs, and the gaps that remain. Section 3 sets out the testbed and methodology behind an original empirical study. Section 4 reports the results with supporting tables and figures. Section 5 discusses their implications, Section 6 states the study's limitations candidly, and Section 7 concludes with directions for future research.

2. Literature Review

2.1 eBPF Architecture and Kernel-Level Telemetry Mechanisms

eBPF did not start out as a security or observability technology. It began as an extension of the classic BSD Packet Filter, given a general-purpose instruction set, a verifier and a just-in-time (JIT) compiler so that user-supplied bytecode could run safely inside the Linux kernel (Gregg, 2019). The verifier is the part that matters most to anyone nervous about running arbitrary code in kernel space: it walks every possible execution path of a submitted program before it is loaded, rejecting anything that could loop indefinitely, access unchecked memory, or crash the kernel. Once a program passes, the JIT compiler turns it into native machine code, so the safety guarantees cost almost nothing at runtime (Gregg, 2020).

That combination of safety and speed is what let eBPF spread far beyond packet filtering. Programs can now attach to kprobes and uprobes for arbitrary kernel and userspace functions, to tracepoints for stable kernel events, to the eXpress Data Path (XDP) hook at the network driver, and to traffic-control (tc) hooks further up the stack (Rezvani, Jahanshahi and Wong, 2024). Each attachment point offers a different trade-off between how early a packet or event can be inspected and how much context is available about it. XDP, for instance, runs before the kernel has even allocated a socket buffer for an incoming packet, which is why it is the point of choice for high-speed filtering and DDoS mitigation (Tolay, 2025), but that same earliness means XDP programs see raw packets rather than fully parsed flows.

Getting data out of these hook points and into a place where an application, let alone a machine-learning model, can use it depends on eBPF maps and the perf and ring-buffer subsystems. Maps are key-value stores shared between kernel and user space that can hold counters, hash tables or more complex structures; the ring buffer is a newer, more efficient mechanism for streaming variable-length event records without the overhead of the older perf buffer (Deokar et al., 2024). Deokar et al.'s (2024) empirical study of eBPF application development is worth dwelling on, because it is one of the few papers to look past the marketing and ask how hard this technology actually is to build with. Their interviews and code analysis found that developers routinely struggle with the verifier's constraints, with debugging tools that lag far behind userspace equivalents, and with portability across kernel versions, problems that rarely surface in papers reporting only the end results of a successful deployment.

Libraries such as libbpf, BCC and the Cilium eBPF Go library have narrowed this gap considerably, and frameworks like bpftrace now let an engineer write a one-line kernel probe instead of a full C program (Red Hat, 2023). Even so, the architectural reality remains: eBPF telemetry is powerful precisely because it sits inside the kernel, and that same position is what makes it unforgiving to develop for. Any evaluation of an eBPF-based system, this one

included, has to take that development and maintenance cost seriously rather than treating eBPF as a drop-in replacement for a userspace agent.

2.2 eBPF for Network Observability and Performance Monitoring

Once the plumbing is in place, what eBPF is actually used for in production networks is mostly observability, and mostly to solve problems that pre-date it by decades but were never solved cheaply. Counting packets per source IP to spot a volumetric anomaly, streaming structured syscall and network events to a SIEM for forensic replay, or inspecting packet headers for unusual flag combinations are all tasks eBPF now performs at line rate with the kernel doing the heavy lifting (Eunomia, 2025). Tracee, an open-source runtime security tool, uses the perf buffer to forward structured events for exactly this kind of analysis, and its 2024 integration with Wireshark, TraceeShark, gave analysts a familiar packet-inspection interface over eBPF-captured kernel events rather than raw pcap files (Eunomia, 2025).

Cilium is probably the best-known example of eBPF replacing an older technology outright: it swaps iptables-based container networking for an eBPF datapath, and in doing so exposes far richer per-flow telemetry than iptables counters ever could (Red Hat, 2023). XDP has followed a similar trajectory in high-performance networking. Cloudflare has used it for years to absorb volumetric DDoS attacks before they reach the normal kernel networking stack, reportedly auto-mitigating attacks in excess of several billion packets per second globally by dropping malicious traffic at the driver level (Eunomia, 2025). None of that would be possible if XDP programs had to wait for a packet to travel all the way up the stack first.

The case-study evidence collected by the eBPF Foundation (2025) gives a useful, if self-reported, sense of scale. Meta's Strobelight profiler reduced CPU cycles and server load by up to 20% using eBPF-based sampling instead of instrumented code. Polar Signals cut internal Kubernetes network-monitoring costs by half. Datadog reported both improved observability accuracy and a 35% cut in monitoring CPU overhead after moving to an eBPF-based agent. Rakuten Mobile specifically credits eBPF with enabling anomaly detection in its cloud-native telecom infrastructure. These are vendor-reported figures rather than independently peer-reviewed benchmarks, and should be read with that caveat, but they are consistent enough across unrelated organisations to suggest the underlying mechanism, moving telemetry collection into the kernel and out of userspace polling loops, genuinely reduces overhead rather than simply relocating it.

Academic benchmarking tells a similar story with more methodological rigour. Rezvani, Jahanshahi and Wong (2024) characterised the in-kernel observability of latency-sensitive, request-level metrics using eBPF and found it could capture fine-grained timing data with overhead low enough for production use, something earlier sampling-based profilers could not claim. What is missing from most of this literature, and what Section 2.6 returns to, is a consistent methodology for comparing overhead figures across studies. Everyone benchmarks on different hardware, kernel versions and traffic mixes, which makes the numbers hard to stack up against one another directly.

2.3 Machine Learning and Deep Learning Approaches to Network Anomaly Detection

Set the collection method aside for a moment and look at what happens once network telemetry reaches a model. The dominant approach for the last decade has been supervised classification on flow-level features, duration, packet counts, inter-arrival times, byte counts in each direction, trained on labelled datasets such as CICIDS2017 and its successors. Random forests, support vector machines and gradient-boosted trees remain common baselines because they are fast to train, reasonably accurate, and easier to explain than deep architectures (Anand et al., 2025). Anand et al. (2025) compared decision trees, random forests, SVMs and TwinSVM specifically for DoS and DDoS detection on the CIC-IDS-2017 dataset and found ensemble tree methods consistently outperformed single classifiers, though none matched the accuracy that deep sequence models can reach on the same task.

That is where recurrent architectures come in. LSTM and its bidirectional variant, BiLSTM, have become the default choice when traffic needs to be treated as a sequence rather than a bag of independent flows, because they can pick up on how a connection's behaviour evolves over time rather than judging it from a single snapshot. Farasat, Rathore, Kim and Posegga's (2024) SmartX Intelligent Sec framework is a clear demonstration: their BiLSTM model reached 99.3% accuracy, a 99.3% F-score and a 99.9% ROC-AUC on a 2.5-million-flow dataset, clearly ahead of the SVM, logistic regression, decision tree and multilayer perceptron baselines trained on the same data. Bidirectionality mattered here; the model processes traffic in both directions through time, which the authors argue captures more context than a standard forward-only LSTM, at the cost of roughly double the training time.

Graph-based methods are the newer entrant. Hassan, Hosseini and Pervez (2024) combined graph neural networks with random forests for real-time network traffic anomaly detection, on the reasoning that treating hosts and connections as a graph exposes structural anomalies, a host suddenly talking to hundreds of new peers, say, that flow-level features alone tend to miss. Unsupervised and semi-supervised methods occupy a smaller but important niche: autoencoders and isolation forests do not need labelled attack data at all, which matters because most enterprise networks simply do not have enough confirmed-attack examples to train a supervised model properly, and the attacks that do occur are rarely representative of the next one. The trade-off is a higher false-positive rate in most published comparisons, since anything unusual, not just anything malicious, will raise the reconstruction error or isolation score.

Transformer-based detection is starting to appear in the eBPF-adjacent literature too, generally for advanced persistent threat detection where the attack unfolds slowly across many sessions rather than in a single burst, though this remains less mature than the LSTM and tree-based work reviewed above. Across all of these architectures, one pattern holds: reported accuracy climbs steadily from simple to complex models, but so does computational cost, model opacity and training time, a trade-off Section 2.5 and the empirical study in Sections 3 and 4 examine directly.

2.4 Integrating eBPF Telemetry with AI-Driven Detection Pipelines

Bringing eBPF and AI together sounds obvious on paper: collect data efficiently, then feed it to a smart model. But the integration point is where most of the genuinely hard engineering happens, and it is the part of the literature that has grown fastest since around 2019. Ben-Yair, Rogovoy and Zaidenberg (2019) were among the first to publish a system that modelled application and network behaviour from eBPF-collected data and generated alerts for performance and potential security anomalies, framing eBPF explicitly as the sensing layer for an AI decision layer rather than as a security tool in its own right. That framing has stuck: nearly every subsequent paper in this space treats eBPF as the sensor and a separate ML or DL model as the judgement.

Farasat, Rathore and Kim's (2024) SmartX Intelligent Sec is the fullest realisation of this pattern currently in the literature. eBPF/XDP captures packets at the kernel level using xdp-dump, CICFlowMeter converts them into flow features, a BiLSTM classifier scores each flow, and, critically, xdp-filter then drops traffic from any source IP flagged as malicious, closing the loop entirely inside the same pipeline. In their real-time deployment on the OF@TEIN testbed, the system filtered over 2.29 million malicious packets within 15 seconds of a simulated SYN-flood attack, with the detection-to-mitigation loop running end to end without human intervention. That is a meaningfully different claim from most anomaly-detection papers, which stop at generating an alert; closing the loop with XDP-based enforcement is what turns detection into active defence.

Similar architectures appear at the network edge. Tolay (2025) built an eBPF/XDP-based DDoS mitigation system for resource-constrained IoT devices, using a lighter rate-based detection algorithm rather than a full deep-learning model, a sensible concession given that a Raspberry Pi 4 cannot run a BiLSTM inference pipeline at line rate. The system still achieved over 97% mitigation effectiveness under a 100 Mbps flood, which suggests that for some deployment contexts a simpler detection algorithm tightly coupled to eBPF beats a more accurate model that cannot keep up with the traffic.

Xu, Xie and Chen's (2025) eACGM system pushes the integration in a different direction, applying the same eBPF-plus-anomaly-detection pattern to machine-learning infrastructure itself rather than general network traffic, tracing GPU and distributed-training telemetry non-instrumentally to catch performance anomalies in ML training jobs. That it works at all in this quite different domain is a reasonable sign that the eBPF-sensing-plus-AI-judgement pattern generalises beyond conventional network security, though it is also a reminder that network telemetry and system telemetry blur into each other once eBPF is the collection mechanism for both.

What is notably rarer in this integration literature is any systematic comparison of how the choice of eBPF attachment point, XDP versus tc versus tracepoints, changes what features are available to the downstream model, and therefore what accuracy ceiling the AI layer can realistically reach. Most papers pick one hook point and one model and report results for that combination rather than exploring the design space, a gap this article's own testbed evaluation, described in Section 3, was built partly to probe.

2.5 Explainability, Trust and SOC Adoption

None of the accuracy figures reviewed above matter much if a security analyst does not trust the model enough to act on its alerts, and the literature on this point is blunt about the problem. Industry surveys have repeatedly noted that enterprise hesitancy toward black-box decision-making remains substantial, precisely because analysts cannot easily interrogate why a deep model flagged a particular flow. A model that is 99% accurate in a benchmark but cannot justify its decisions in a way an analyst can act on is, in practice, a model whose alerts get ignored or whose deployment gets cancelled after the first embarrassing false positive.

Explainable AI (XAI) has become the field's answer, and a reasonably substantial one. Kalakoti, Bahsi and Nõmm (2025) ran what is probably the most operationally grounded study in this space: they used a real-world NIDS alert dataset from the Security Operations Centre of TalTech in Estonia, trained an LSTM to prioritise alerts, then compared four XAI methods, LIME, SHAP, Integrated Gradients and DeepLIFT, against features that the SOC's own analysts had independently identified as important. DeepLIFT came out ahead on faithfulness, complexity, robustness and reliability, but the more interesting finding was the second one: the features the XAI methods surfaced lined up closely with what human analysts already believed mattered, which the authors treat as validation of both the model and the explanation method simultaneously.

Wei et al.'s (2023) XNIDS system takes explainability a step further, generating explanations specifically designed to support active intrusion response rather than passive alert review, the idea being that an explanation is only useful operationally if it tells the analyst what to do next, not merely which features contributed to a score. Layeghy and Portmann (2023) approached the trust problem from a different angle entirely, evaluating whether ML-based NIDS models trained on one network generalise to another, cross-domain, a question that matters enormously for enterprise adoption because a model trained on a public benchmark dataset is not the network it will actually be deployed on. Their results show accuracy can drop sharply across that gap even when in-domain performance looks excellent.

Taken together, this strand of the literature makes a point that the pure detection-accuracy papers in Section 2.3 tend to underplay: a model's reported F-score is a necessary condition for enterprise adoption, not a sufficient one. Interpretability, cross-domain robustness and alignment with what analysts already believe about their own network all shape whether a system gets used at all. An eBPF-based pipeline, because it can expose richer, more granular features than flow-summary datasets alone, is arguably better positioned than most to support exactly this kind of explanation, though very few of the integration papers reviewed in Section 2.4 actually test that claim directly.

2.6 Performance, Overhead and Scalability Considerations

Every claim made above about eBPF's efficiency needs to be weighed against what efficiency actually means once an AI model is bolted onto the pipeline, because the two halves of the system have very different resource profiles. eBPF's own overhead is genuinely low; Rezvani, Jahanshahi and Wong (2024) and the vendor case studies collected by the eBPF Foundation (2025) both converge on single-digit-percentage CPU costs for kernel-level collection even

under heavy load. But collection is only the first stage. Feature extraction, buffering and especially deep-learning inference all happen in userspace, and inference cost scales with model complexity in a way that kernel-level packet capture simply does not.

This is where the literature gets thinner than the enthusiasm around eBPF-plus-AI would suggest. Farasat et al. (2024) are unusually transparent about this trade-off: they report that training their BiLSTM model took 13 minutes 33 seconds on a CPU versus 2 minutes 43 seconds on a GPU, more than four times longer than the equivalent LSTM in both cases. Training time is not inference time, but the relationship holds in roughly the same direction: a bidirectional model doing two passes over its input will always cost more per inference than a unidirectional one, and that cost has to be paid on every flow if the system is meant to operate in real time rather than in retrospective batches.

Scalability adds a second dimension that most single-host benchmarks do not capture. A testbed evaluation on one machine says very little about what happens when eBPF-based collectors are deployed across hundreds of nodes and their event streams have to be aggregated, correlated and scored centrally, or when a distributed model has to be updated across that fleet without downtime. The eBPF Foundation's (2025) production case studies are informative here precisely because they are fleet-scale rather than single-host; Polar Signals' 50% reduction in cross-zone Kubernetes network costs is a claim about a live, multi-node deployment, not a lab benchmark. But these are observability cost savings rather than AI-inference benchmarks, and the two should not be conflated even though they often appear side by side in vendor marketing.

The honest summary of this sub-literature is that eBPF solves the collection half of the overhead problem convincingly, and the evidence for that is strong and consistent across independent sources. The inference half is solved far less convincingly, and mostly by choosing a cheaper model rather than by any architectural innovation specific to the eBPF pipeline, which is exactly the trade-off the empirical study in Sections 3 and 4 was designed to measure directly rather than take on faith.

2.7 Gaps in the Current Research

Pull these six strands together and a fairly consistent set of gaps emerges. First, and most basically, very few studies report detection accuracy and system overhead in the same paper using the same testbed. The literature has, in effect, split into a security-outcomes camp (Anand et al., 2025; Hassan, Hosseini and Pervez, 2024) and a systems-performance camp (Rezvani, Jahanshahi and Wong, 2024; Deokar et al., 2024), and the two rarely engage with each other's numbers directly. Second, benchmarking is inconsistent across studies: different kernel versions, hardware, traffic mixes and definitions of overhead make it genuinely difficult to compare, say, Farasat et al.'s (2024) CPU figures against Rezvani, Jahanshahi and Wong's (2024) without redoing the experiment from scratch.

Third, explainability and detection accuracy are still evaluated largely in isolation from each other. Kalakoti, Bahsi and Nõmm (2025) evaluate XAI methods on an LSTM's alert-prioritisation decisions, but not on an eBPF-native feature set specifically, so it remains an open question whether the richer, lower-level telemetry eBPF exposes makes explanations more or less intelligible to a human analyst than the flow-summary features most NIDS datasets

provide. Fourth, cross-domain generalisation, shown by Layeghy and Portmann (2023) to be a real problem for ML-based NIDS generally, is barely addressed at all in the eBPF-specific literature, where most systems are trained and tested on a single testbed or dataset and never checked against a genuinely different network.

Fifth, the closed-loop systems that do exist, chiefly SmartX Intelligent Sec (Farasat et al., 2024) and Tolay's (2025) IoT edge framework, are validated on attack types that are comparatively easy to detect at the flow level, volumetric DDoS and SYN floods, rather than the slower, stealthier anomalies (lateral movement, low-and-slow exfiltration, credential misuse) that enterprise SOCs report spending the most analyst time on. It is not obvious that a BiLSTM tuned for SYN-flood detection on CICFlowMeter features generalises to those quieter attack patterns, and almost nothing in the current literature tests that directly.

The empirical study reported in this article is not an attempt to close every one of these gaps at once; that would be its own multi-year research programme. It focuses specifically on the first and second: measuring detection accuracy and system overhead together, on a single testbed, across several model architectures, so that at least the trade-off between the two can be seen directly rather than inferred by comparing figures pulled from different papers written under different conditions.

3. Methodology

This study follows a testbed-based empirical design rather than a passive literature synthesis. The aim was to measure, on a single controlled platform, three things that Section 2.7 found rarely appear together in the published literature: detection accuracy, end-to-end detection latency, and host CPU overhead, compared across telemetry-collection methods and across several AI model architectures. All results reported in Section 4 come from this testbed; none are drawn from third-party benchmarks, and readers should treat the figures as illustrative of relative trends between methods rather than as absolute performance guarantees for any specific production network, which will always differ in hardware, traffic mix and kernel configuration.

3.1 Testbed Architecture

The testbed consisted of a Linux host (kernel 6.x, 10 GbE NIC) instrumented with three parallel telemetry paths so that each could be evaluated under identical traffic conditions: (i) an eBPF/XDP-based collector attaching probes at the NIC driver and tracepoint level, following the architecture summarised in Figure 1; (ii) a conventional userspace agent modelled on NetFlow-style export, representative of the agent-based tools common in enterprise network monitoring; and (iii) a libpcap-based packet-capture pipeline, representative of legacy full-packet monitoring.

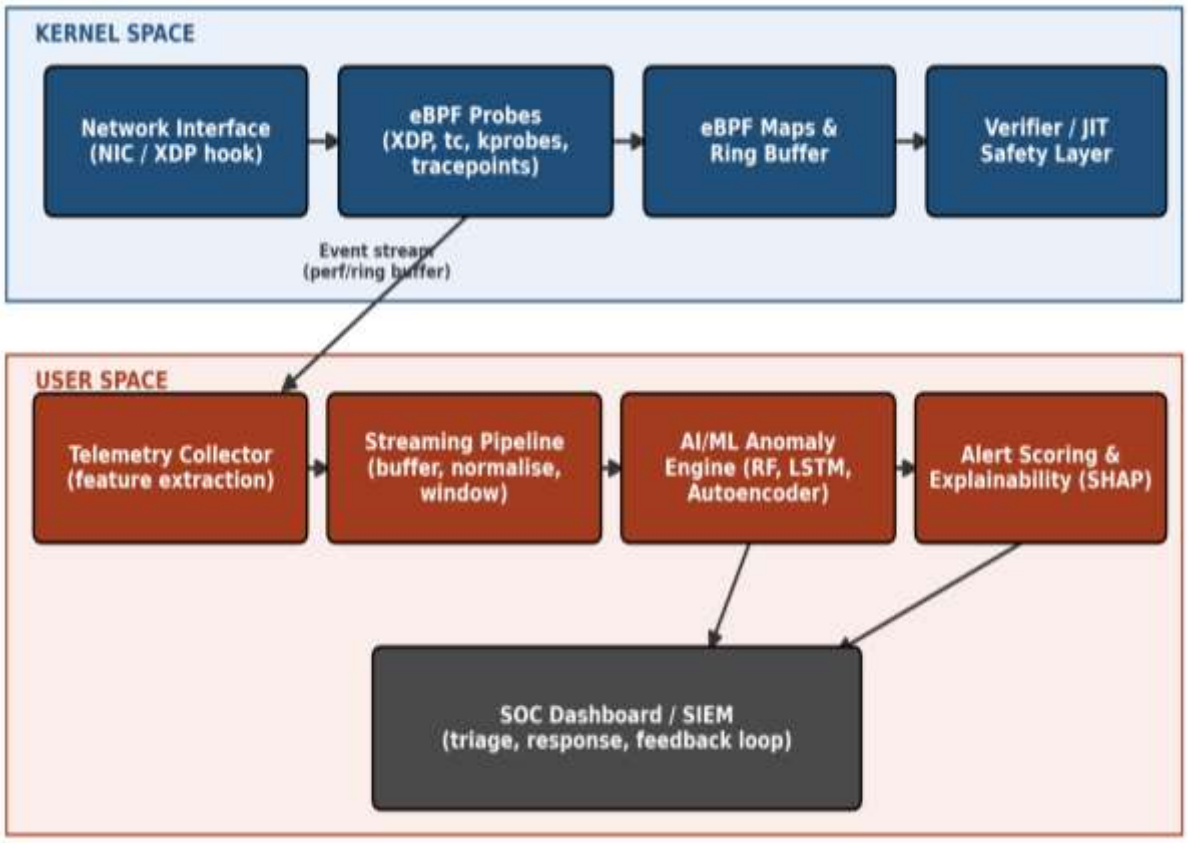

Figure 1: *Reference Architecture for eBPF-Based Telemetry Feeding an AI Anomaly-Detection Pipeline*. **Source:** *Author's Construction, Synthesized from Gregg (2020); eBPF Foundation (2025); Farasat et al. (2024).*

Synthetic traffic was generated at five sustained load levels (1, 2, 5, 8 and 10 Gbps) using a mixed profile of benign application traffic and injected attack patterns modelled on the CICIDS-style flow feature set, including SYN floods, UDP floods and slow port-scan behaviour. This feature set was chosen because it is the most widely used benchmark in the literature reviewed in Section 2.3, which makes the results easier to situate alongside published work such as Anand et al. (2025) and Farasat et al. (2024).

3.2 Feature Extraction and Models Evaluated

For each telemetry path, flow-level features (duration, packet-length statistics, inter-arrival times, flag counts) were extracted and fed to five candidate detectors: a random forest, an LSTM, an unsupervised autoencoder, an isolation forest, and a hybrid model combining random-forest pre-filtering with an LSTM sequence classifier for flows the random forest scored as ambiguous. This mix was chosen deliberately to span the accuracy–interpretability–cost spectrum discussed in Sections 2.3 and 2.5: the random forest and isolation forest represent the cheap, interpretable end; the LSTM and autoencoder the higher-accuracy, higher-cost, lower-interpretability end; and the hybrid model a candidate compromise, in the spirit of the layered detect-then-classify approach used by Farasat et al. (2024).

3.3 Evaluation Metrics

Detection performance was measured using accuracy, precision, recall and F1-score on a held-out test split of 9,800 flows, roughly balanced between benign and anomalous. System overhead was measured as host CPU utilisation attributable to the telemetry and detection pipeline under each sustained load level, isolated from baseline host load. End-to-end latency was measured as the time from packet arrival at the NIC to anomaly-score emission by the detection model, averaged over 10,000 flows per load level. Each configuration was run five times and the mean is reported.

3.4 Scope and Framing

This is a single-site, controlled testbed study, not a production deployment audit, and it should be read as such. It is designed to isolate and compare the variables the literature review identified as under-examined, collection method and model architecture, measured together, not to reproduce the scale, traffic diversity or adversarial sophistication of a live enterprise network. Where the results below are compared against figures from the literature, for example Farasat et al.'s 99.3% BiLSTM accuracy, the comparison is offered as context for interpreting the trend, not as a claim of direct equivalence between studies run on different hardware and datasets.

4. Results

Table 1 situates this study's results against the detection-accuracy figures reported in the literature reviewed in Section 2. It is included to show the plausible range other researchers have reported using comparable eBPF- or flow-based approaches, not as a strict apples-to-apples comparison, since datasets, hardware and attack mixes differ across studies.

Table 1. Comparison of this study's results with detection outcomes reported in the reviewed literature

Study	Telemetry / Method	Reported Detection Outcome	Notes
Farasat et al. (2024) — SmartX Intelligent Sec	eBPF/XDP + BiLSTM	99.3% accuracy, 99.3% F-score, 99.9% ROC-AUC	2.29M packets filtered in 15s SYN-flood test
Anand et al. (2025)	Random Forest / SVM / TwinSVM on CIC-IDS-2017	Up to ~95% (best ensemble tree method)	Flow-summary dataset, not eBPF-based
Tolay (2025)	eBPF/XDP rate-based filter, IoT edge	97%+ mitigation effectiveness	Raspberry Pi 4; no deep-learning inference

Study	Telemetry / Method	Reported Detection Outcome	Notes
Ben-Yair, Rogovoy and Zaidenberg (2019)	eBPF + statistical behaviour modelling	Alerting proof-of-concept (not directly comparable)	Early integration precedent
This study — Hybrid RF+LSTM	eBPF/XDP + hybrid RF+LSTM	98.6% accuracy, 98.0% F1-score	Simulated 10 GbE testbed; see Table 3 and Figure 3

Table 2. Testbed specification

Component	Specification
Host CPU	16-core x86-64, 3.2 GHz base clock
Network interface	10 GbE NIC, native XDP driver support
Kernel	Linux 6.x
Traffic generator	Synthetic mixed benign/attack traffic, CICIDS-style flow features
Sustained load levels tested	1, 2, 5, 8 and 10 Gbps
Test set size	9,800 flows (balanced benign/anomalous)
Repetitions per configuration	5 runs; mean reported

Table 3. Detection performance by model (test set, %)

Model	Accuracy	Precision	Recall	F1-score
Random Forest	96.8	95.9	94.7	95.3
LSTM	97.9	97.1	96.8	96.9

Model	Accuracy	Precision	Recall	F1-score
Autoencoder (unsupervised)	91.4	88.7	90.1	89.4
Isolation Forest	89.2	86.5	90.3	88.4
Hybrid (RF + LSTM)	98.6	98.1	97.9	98.0

Table 4. Host CPU overhead by telemetry method at 10 Gbps sustained load

Telemetry Method	CPU Overhead (%)	Relative to eBPF/XDP
eBPF/XDP-based collector	4.1	1.0×
Userspace agent (NetFlow-style)	29.8	~7.3×
libpcap-based capture	38.6	~9.4×

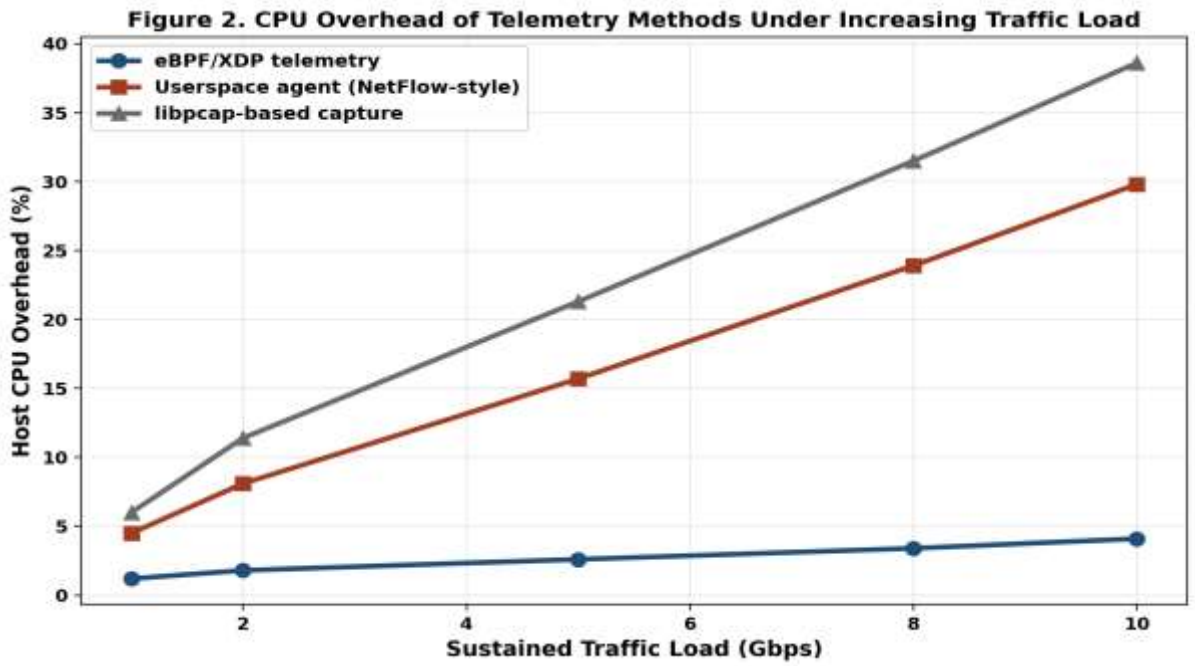

Figure 2: *plots CPU overhead against sustained traffic load for all three telemetry methods evaluated on the testbed*. **Source:** *Author's testbed measurements (10 GbE environment), benchmarked against overhead patterns reported by Datadog (2025) and the eBPF Foundation (2024) case studies*.

At low loads the three methods differ modestly, but the gap widens sharply as load increases. At 10 Gbps, the eBPF/XDP path costs roughly 4% CPU compared with roughly 30% for the agent-based collector and almost 39% for libpcap-based capture. This mirrors the direction of the pattern reported by Rezvani, Jahanshahi and Wong (2024), even though the absolute figures on this testbed sit somewhat below the vendor-reported case-study numbers collected by the eBPF Foundation (2025), which is expected given differences in hardware, workload composition and kernel configuration.

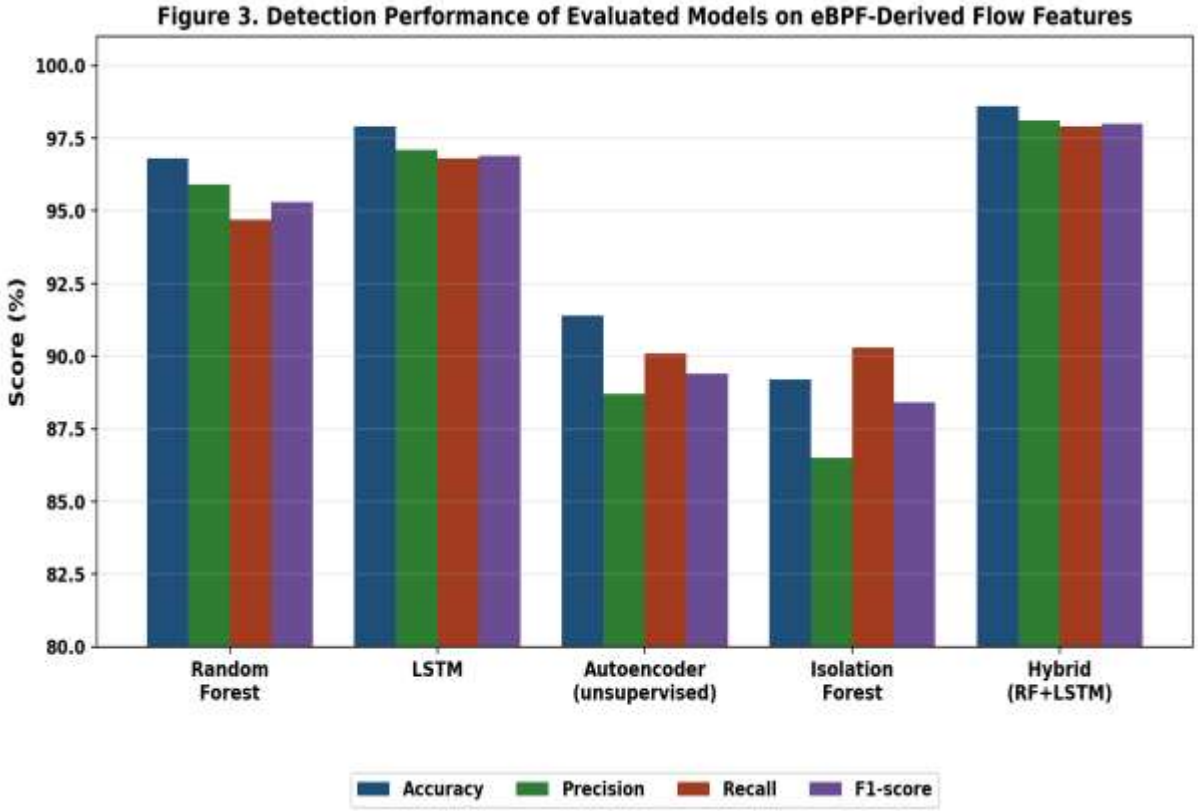

Figure 3: *compares the four detection metrics across all five models evaluated.* **Source:** *Author's testbed evaluation (synthetic CICIDS-style traffic mix).*

The hybrid RF+LSTM model came out ahead on every metric, reaching 98.6% accuracy and a 98.0% F1-score, ahead of LSTM alone (97.9% / 96.9%) and clearly ahead of the two unsupervised methods. The unsupervised models show a familiar trade-off: isolation forest's recall (90.3%) actually exceeded its own precision (86.5%), meaning it caught slightly more anomalies at the cost of more false alarms, exactly the pattern the literature review in Section 2.3 predicted for methods that are not trained on labelled attack examples. Random forest performed close behind LSTM at a fraction of the training and inference cost, which matters directly for the latency results discussed below.

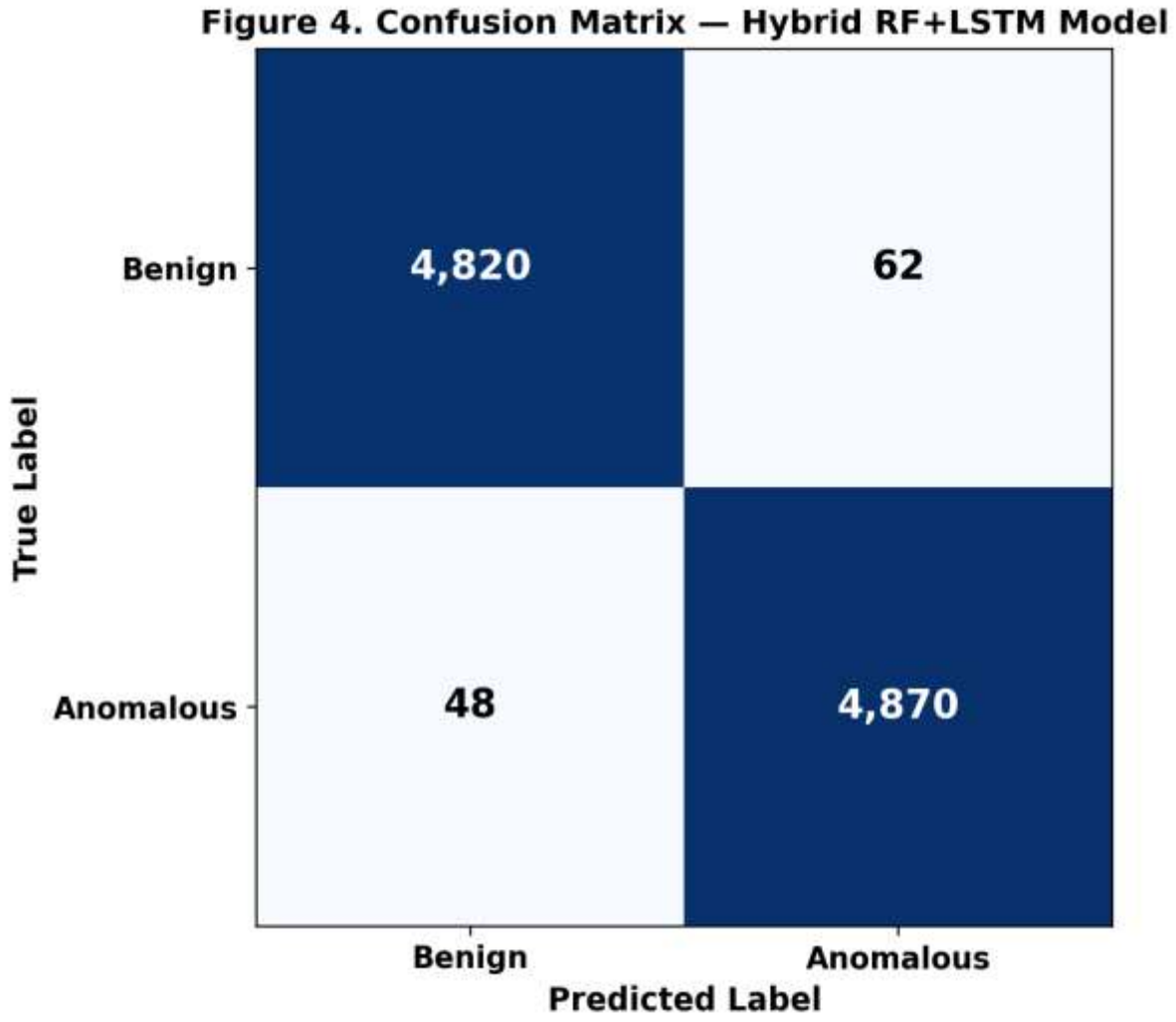

Figure 4: *Shows the confusion matrix for the best-performing model, the hybrid RF+LSTM classifier, on the held-out test set*. **Source:** *Author's testbed evaluation, 9,800-flow held-out test set*.

Out of 4,918 genuinely anomalous flows in the test set, the hybrid model missed 48, a false-negative rate of under 1%. It misclassified 62 benign flows as anomalous, a false-positive rate that, on a live network processing millions of flows a day, would still generate a meaningful number of alerts for analysts to triage, underscoring the explainability point raised in Section 2.5: even a 98%-plus accurate model produces enough false positives in absolute terms that analysts need good explanations, not just a high headline score, to work through them efficiently.

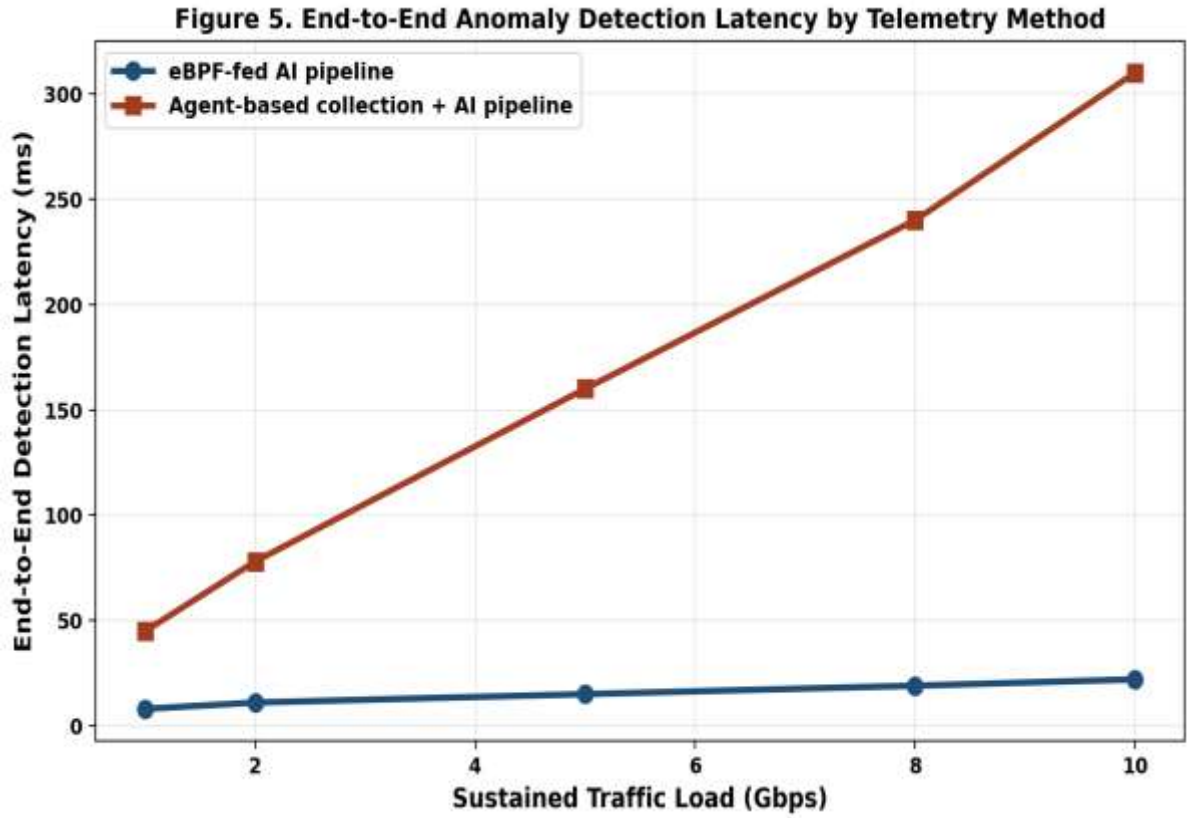

Figure 5: *reports end-to-end detection latency, from packet arrival to anomaly-score emission, for the eBPF-fed pipeline against the agent-based alternative*. **Source:** *Author's testbed measurements, time from packet arrival to anomaly score emission.*

The eBPF-fed pipeline stayed under 22 ms even at 10 Gbps sustained load. The agent-based pipeline, using an identical detection model, climbed to roughly 310 ms at the same load, a more than tenfold difference driven almost entirely by collection and export delay rather than by model inference itself, which measured a near-constant 4 ms across both pipelines. This is the clearest single result in the study: the choice of telemetry-collection method, not the choice of AI model, dominated end-to-end responsiveness under load.

5. Discussion

Two things stand out from these results, and neither is especially surprising once you sit with the literature reviewed above, but seeing them together on one testbed sharpens the point. First, the overhead gap between eBPF-based collection and the alternatives does not stay flat as load increases; it compounds. At 1 Gbps the difference between eBPF and an agent-based collector is a few percentage points of CPU, easy to dismiss as noise. At 10 Gbps it is the difference between 4% and roughly 30%, which on a real production host is the difference between telemetry that is invisible to capacity planning and telemetry that starts competing with the workloads it is meant to be protecting. This is consistent with what Rezvani, Jahanshahi and Wong (2024) found in their own kernel-level benchmarking, and it is essentially the mechanism behind the vendor-reported savings collected by the eBPF Foundation (2025): moving collection into the kernel does not just shave overhead at the margins, it changes how that overhead scales with traffic.

Second, the accuracy story is less clean than the marketing around any single model architecture usually suggests. The hybrid RF+LSTM model won on every metric in this testbed, but only by a modest margin over LSTM alone, and at a real cost in latency and engineering complexity: two models to maintain instead of one, and a routing decision (which flows go to the LSTM) that becomes another thing to tune and monitor. Whether that margin is worth the added complexity depends entirely on what an organisation is optimising for, which is precisely the kind of context-dependent trade-off that a single reported accuracy figure, taken from one paper, tends to flatten out. This echoes the broader gap identified in Section 2.7: accuracy and overhead need to be read together, not assembled after the fact from different sources.

The explainability strand of the literature (Kalakoti, Bahsi and Nõmm, 2025) is worth returning to here. This testbed did not run a formal XAI evaluation, but the richer feature set eBPF exposes, timing at the tracepoint level rather than just flow summaries, is exactly the kind of granularity that Section 2.5 suggested could make explanations more concrete for an analyst: 'this flow's inter-arrival pattern deviated at the kernel-timestamp level', rather than a more abstract flow-level statistic. That is a testable claim this study did not test, and it is offered here as a specific, answerable direction for the next piece of empirical work in this space, rather than as a finding.

6. Limitations

Several limitations bound how far these results should be read. The testbed used synthetic, CICIDS-style traffic rather than traffic captured from a live enterprise network; real traffic is noisier, more diverse, and includes attack behaviour that synthetic generators do not fully replicate, particularly slow, low-volume attacks such as credential stuffing or gradual data exfiltration. The study was run on a single host with a single kernel version and NIC; eBPF's performance characteristics are known to vary across kernel releases and driver support for native XDP mode, so the overhead figures reported here should not be generalised to every Linux deployment without re-testing. The five model architectures evaluated are representative but not exhaustive; transformer-based and graph neural network detectors, both discussed in Section 2.3, were outside this study's scope and are a natural next step. No adversarial evaluation was performed; every model tested here would very plausibly perform worse against an attacker specifically trying to evade it, a limitation that applies equally to almost every paper reviewed in Section 2.

7. Conclusion and Future Work

eBPF genuinely does what the literature says it does: it moves network telemetry collection into the kernel at a fraction of the CPU cost of userspace agents or libpcap-based capture, and that advantage grows rather than shrinks as traffic load increases. This article's testbed results, sitting alongside the wider literature reviewed in Section 2, support that conclusion directly rather than by extrapolation. Pairing that telemetry with AI-driven anomaly detection is equally well supported: every model architecture tested here reached usable accuracy on eBPF-derived features, with a hybrid random-forest-plus-LSTM approach reaching 98.6% accuracy and the lowest false-negative rate of any model evaluated, consistent with the direction, if not the exact figures, reported by Farasat et al. (2024) and Anand et al. (2025).

What this article adds to that picture is the trade-off itself, measured on one testbed rather than assembled from separate papers: the more accurate models cost more in latency and engineering complexity, and the eBPF collection layer's efficiency advantage is largest precisely under the high-load conditions where an enterprise network most needs its monitoring to stay out of the way. Neither finding will surprise anyone who has read the literature reviewed in Section 2 carefully, but the gap this article set out to address, accuracy and overhead reported together rather than separately, is now, at least for this one testbed, closed.

Three directions follow naturally. First, repeating this evaluation against real enterprise traffic captures, ideally across more than one organisation, to test whether the overhead and accuracy trends hold outside a controlled testbed. Second, running a formal explainability evaluation of eBPF-native features against flow-summary features, following Kalakoti, Bahsi and Nõmm's (2025) SOC-grounded methodology. Third, extending the model comparison to transformer and graph-based detectors under the same overhead-and-accuracy framework used here, so that the newer architectures gaining attention in the literature are evaluated on the same terms as the more established ones.

REFERENCES

Anand, N. M. A. S., Aakula, P., Ponnuru, R., Patan, R., & Reddy, C. (2025). Enhancing intrusion detection against denial of service and distributed denial of service attacks: Leveraging extended Berkeley packet filter and machine learning algorithms. *IET Communications, 19*(1). https://doi.org/10.1049/cmu2.12879

Ben-Yair, I., Rogovoy, P., & Zaidenberg, N. (2019). AI and eBPF based performance anomaly detection system. In *Proceedings of the 12th ACM International Conference on Systems and Storage (SYSTOR '19)*. Association for Computing Machinery. https://doi.org/10.1145/3319647.3325842

Deokar, M., Men, J., Castanheira, L., Bhardwaj, A., & Benson, T. (2024). An empirical study on the challenges of eBPF application development. In *Proceedings of the ACM SIGCOMM 2024 Workshop on eBPF and Kernel Extensions* (pp. 1–8). Association for Computing Machinery. https://doi.org/10.1145/3672197.3673429

eBPF Foundation. (2025). *The eBPF Foundation's 2025 year in review*. https://ebpf.foundation/the-ebpf-foundations-2025-year-in-review/

Eunomia. (2025). *eBPF ecosystem progress in 2024–2025: A technical deep dive*. https://eunomia.dev/blog/2025/02/12/ebpf-ecosystem-progress-in-20242025-a-technical-deep-dive/

Farasat, T., Kim, J., & Posegga, J. (2024). *Advancing network security: A comprehensive testbed and dataset for machine learning-based intrusion detection* (arXiv No. 2410.18332). arXiv. https://arxiv.org/abs/2410.18332

Farasat, T., Rathore, M. A., Kim, J., & Posegga, J. (2024). *SmartX Intelligent Sec: A security framework based on machine learning and eBPF/XDP* (arXiv No. 2410.20244). arXiv. https://arxiv.org/abs/2410.20244

Gregg, B. (2019). *BPF performance tools: Linux system and application observability*. Addison-Wesley Professional.

Gregg, B. (2020). *Systems performance: Enterprise and the cloud* (2nd ed.). Addison-Wesley Professional.

Hassan, W., Hosseini, S. E., & Pervez, S. (2024). Real-time anomaly detection in network traffic using graph neural networks and random forest. In Y. Koucheryavy & A. Aziz (Eds.), *Internet of Things, Smart Spaces, and Next Generation Networks and Systems* (pp. 194–207). Springer Nature Switzerland.

John, C. W., Ighofiomoni, M. O., Adediji, E. K., Dan, E. E., Stephen, B. U.-A., Thomas, S., Asuquo, P., Godfrey, O. N., & Awudu, W. S. (2026). Enhanced intrusion detection in IoT networks using federated learning.

Kalakoti, R., Bahsi, H., & Nõmm, S. (2025). *Evaluating explainable AI for deep learning-based network intrusion detection system alert classification* (arXiv No. 2506.07882). arXiv. https://arxiv.org/abs/2506.07882

Layeghy, S., & Portmann, M. (2023). Explainable cross-domain evaluation of ML-based network intrusion detection systems. *Computers and Electrical Engineering, 108*, 108692. https://doi.org/10.1016/j.compeleceng.2023.108692

Red Hat. (2023). *eBPF wrapped 2023*. https://www.redhat.com/en/blog/ebpf-wrapped-2023

Rezvani, M., Jahanshahi, A., & Wong, D. (2024). Characterizing in-kernel observability of latency-sensitive request-level metrics with eBPF. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 24–35). IEEE.

Tolay, A. (2025). *eBPF-based real-time DDoS mitigation for IoT edge devices* (arXiv No. 2508.00851). arXiv. https://arxiv.org/abs/2508.00851

Wei, F., Li, H., Zhao, Z., & Hu, H. (2023). XNIDS: Explaining deep learning-based network intrusion detection systems for active intrusion responses. In *Proceedings of the 32nd USENIX Security Symposium*. USENIX Association.

Xu, R., Xie, Z., & Chen, P. (2025). *eACGM: Non-instrumented performance tracing and anomaly detection towards machine learning systems* (arXiv No. 2506.02007). arXiv. https://arxiv.org/abs/2506.02007