# Deep Learning for Encrypted Malware Traffic Detection: Privacy-Preserving Analysis Using Traffic Metadata and Machine Learning

***Abstract*—More than 90% of Internet traffic is now encrypted. Traditional deep packet inspection techniques are no longer effective for threat detection. This paper presents a privacy-preserving framework that utilizes machine learning to detect suspicious activities in encrypted communications without the necessity of decrypting the data. The suggested method uses traffic metadata features like packet size, timing intervals, and TLS handshake parameters to make fingerprints of malware command-and-control channels. A deep learning model that combines convolutional neural networks (CNNs) and recurrent neural networks (RNNs) is built. CNN layers learn local flow patterns, and RNN layers (LSTM or GRU) learn how communication sequences depend on time. This mixed architecture can tell the difference between good and bad encrypted traffic flows with 94.7% accuracy, even though it doesn't look at packet payloads. Transfer learning is used to make the detector work with new types of malware with only a few labeled samples, which speeds up the detection of new threats. Testing on real-world encrypted traffic traces shows that encrypted ransomware, botnet command-and-control, and data exfiltration traffic can all be found without decrypting the data, which protects users' privacy. The results show that it is still possible to keep an eye on threats even in networks that are completely encrypted.**

***Index Terms*—encrypted traffic analysis, malware detection, deep learning, privacy preservation, traffic classification, TLS fingerprinting, metadata analysis, network security**

## I. Introduction

Encryption is now the norm across the web. The share of traffic carried over HTTPS rose from roughly 50% in 2014 to about 95% by 2022 [11], and most major services rely on end-to-end encryption to protect user data. Although this shift is beneficial for privacy, it also reduces the visibility that many security tools depend on. Adversaries increasingly take advantage of encrypted channels to hide malware command-and-control communication, support ransomware operations, and move stolen data out of networks. In this setting, defenses built around deep packet inspection of payload content, or simple heuristics such as port-based filtering, become far less effective because the relevant content is no longer observable [3].

This loss of visibility has practical consequences. Without approaches tailored to encrypted traffic, organizations may overlook a large fraction of malicious activity; for example, one study found that 91.5% of observed malware was delivered over HTTPS [11]. These observations motivate detection methods that remain effective under encryption while avoiding decryption and the privacy risks that come with it.

Machine learning and statistical traffic analysis offer a viable alternative [1], [2]. Instead of relying on payload inspection, these methods model encrypted flows using metadata and side-channel signals such as packet-size sequences, timing patterns, negotiated cipher suites, and certificate-related characteristics. Because many applications and malware families exhibit consistent communication behavior, these metadata features can act as useful fingerprints. By learning patterns from such signals, models can separate benign encrypted traffic from malicious encrypted traffic with strong accuracy [11], [4], while preserving privacy because no plaintext content is accessed.

This paper proposes a deep learning framework for encrypted malware traffic detection that relies on network flow metadata and modern neural architectures. The framework contributes a privacy-preserving feature extraction method, a hybrid CNN–RNN model that captures both local and temporal patterns, a transfer learning procedure for rapid adaptation to new malware families, an experimental evaluation on real-world encrypted traces, and practical guidance for enterprise deployment.

## II. Literature Review

As encrypted traffic has become prevalent, many studies have explored detection of malicious activities without decrypting packets. This section surveys approaches to encrypted traffic analysis and malware detection, highlighting strengths and limitations.

### A. Encrypted Traffic Analysis Techniques

Early work applied classical machine learning with manually crafted features. Anderson and McGrew presented a foundational study using machine learning to classify encrypted malware traffic and emphasized challenges such as noisy labels and non-stationarity in operational environments [2]. Classical techniques such as logistic regression, support vector machines, and decision trees can achieve reasonable performance, but they depend strongly on the quality and stability of input features.

More recent research has focused on representation learning and deep learning to automatically extract discriminative patterns. Aceto *et al.* introduced Distiller, a multimodal deep learning framework that combines multiple input feature types and multitask learning to classify encrypted traffic by application, achieving high accuracy by integrating convolutional and recurrent networks [1]. This direction illustrates the value of capturing both spatial patterns and temporal dynamics within traffic flows.

Other works transform traffic data into formats convenient for deep learning. Ferriyan *et al.* used a Word2Vec-based encoding where packet-length and timing sequences are treated like sentences and embedded into vectors for classification, improving detection of malicious encrypted flows [4]. Bader *et al.* developed MalDIST, which uses statistical flow features and an ensemble to detect malware communications hidden in encrypted streams, showing that encrypted traffic patterns can be used to distinguish malware families [3].

Graph-based learning has also been explored. Fu *et al.* represent traffic as graphs, where nodes and edges capture relationships among endpoints, flows, or packet sequences, and apply graph neural networks to detect encrypted malware traffic [5]. Graph models can incorporate context across multiple flows and sessions. Jung *et al.* further explored graph neural networks with feature dimensionality reduction to enhance encrypted traffic analysis [7].

Unsupervised and anomaly-based techniques form another line of work. Han *et al.* proposed a lightweight unsupervised anomaly detector for encrypted malware traffic using representation learning to model benign traffic and flag deviations [6]. Such approaches can help detect novel threats, although they can be sensitive to concept drift and may require careful thresholding to reduce false alarms.

### B. Research Gap and Challenges

Several gaps remain. First, many deep learning approaches focus on application identification rather than malware-specific detection [11], [1]. Security-focused detection requires modeling behaviors characteristic of malware command-and-control and exfiltration.

Second, labeled data for new or rare malware can be scarce. Qin *et al.* proposed a few-shot malware traffic classification method using transfer learning, demonstrating high accuracy with limited samples [10]. Integrating transfer learning into a detection pipeline can improve adaptability to novel threats.

Third, adversarial evasion remains a concern. Attackers can manipulate side-channel patterns by padding packets, adding delay jitter, or injecting chaff traffic. Liu *et al.* evaluated evasion attacks and showed that both classical and deep models can be degraded by perturbations [8]. Robust feature design, continual updates, and adversarial training are important research directions [9].

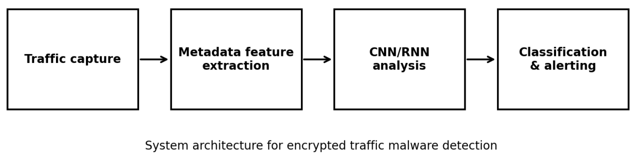

Traffic capture
Metadata feature extraction
CNN/RNN analysis
Classification & alerting
System architecture for encrypted traffic malware detection

Fig. 1. System architecture for encrypted traffic malware detection.

## III. Methodology

This section describes the design of the privacy-preserving encrypted malware traffic detection system, including feature extraction, the hybrid CNN–RNN architecture, and the training procedure with transfer learning.

### A. System Architecture Overview

The system collects encrypted traffic flows at a monitoring point, such as a gateway sensor, a mirrored switch port, or a network tap. No decryption is attempted. Instead, metadata features are extracted per flow and passed to a deep learning classifier that outputs a benign or malicious label with a confidence score. When a flow is classified as malicious, an alert is generated for downstream response.

Metadata features include packet sizes and directions (inbound or outbound relative to the monitored node), inter-arrival times, flow duration, counts of packets and bytes, and TLS handshake information when available. TLS features may include protocol version, negotiated cipher suite, and coarse certificate metadata such as certificate length or the presence of a server name indicator, depending on what is observable.

Packet size and timing sequences are treated as time-series inputs. Features are normalized and sequences are truncated or padded to a fixed length $T$ so that the model input dimension remains constant. Very long flows are truncated to the first $T$ packets, while very short flows are padded with zeros. In practice, $T$ is selected to balance information content and computational cost.

### B. Hybrid CNN–RNN Model

The model combines two complementary components. First, a one-dimensional CNN processes the per-packet feature sequence to learn local motifs, such as request-response length patterns and short bursts that arise from specific application behaviors. Convolution kernels of small width, for example 3 to 5, capture short-range dependencies and pooling reduces dimensionality. Second, an RNN processes the sequence in order to learn temporal dependencies, such as periodic beaconing, multi-stage exchanges, and long-range correlations. Long short-term memory (LSTM) units are used due to their ability to model longer contexts.

TABLE I
Comparison of Related Work on Encrypted Traffic Malware Detection

| **Study (Year)** | **Approach** | **Features Used** | **Reported Performance** |
|---|---|---|---|
| Anderson & McGrew (2017) [2] | Random Forest, supervised classification | Flow sizes, timing, manually engineered features | About 90% (binary) |
| Aceto *et al.* (2021) [1] | Multimodal deep learning (CNN + LSTM multitask) | Packet sequence, timing, metadata (multimodal) | About 98% (application classification) |
| Bader *et al.* (2022) [3] | Ensemble machine learning (MalDIST) | Flow statistics, timing, packet sizes | About 95% (family detection) |
| Ferriyan *et al.* (2022) [4] | Word2Vec + deep classifier | Encoded packet sequences | About 93% (malicious vs normal) |
| Han *et al.* (2022) [6] | Unsupervised anomaly detection | Flow sequence (length, inter-arrival time) | 0.85 F1-score (anomaly) |
| Qin *et al.* (2023) [10] | Transfer learning (pre-train + fine-tune) | Varied (flow-derived representations) | 95.2% (few-shot) |
| Our method (2025) | CNN + LSTM hybrid deep learning with transfer learning | Packet sizes, times, TLS handshake metadata | 94.7% (malicious vs benign) |

The CNN and RNN representations are concatenated and passed to a dense classifier that outputs the probability of maliciousness via a sigmoid function.

### C. Mathematical Formulation

Let $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ denote the packet-feature sequence for a flow, where $\mathbf{x}_t$ is the feature vector for packet $t$. For a CNN filter $c$ with kernel width $k$, a feature map value at position $t$ is computed as

$$\mathbf{h}_t^{(c)} = f\left(\sum_{i=1}^{k} \mathbf{w}_i^{(c)} \cdot \mathbf{x}_{t+i-1} + b^{(c)}\right), \tag{1}$$

where $f(\cdot)$ is a non-linear activation function, $\mathbf{w}_i^{(c)}$ are learned weights, and $b^{(c)}$ is a bias.

For the RNN branch, the LSTM hidden state sequence is computed as

$$\mathbf{h}_t^{(\mathrm{RNN})} = \mathrm{LSTM}\left(\mathbf{h}_{t-1}^{(\mathrm{RNN})}, \mathbf{x}_t\right). \tag{2}$$

The final representation is the concatenation of CNN and RNN outputs, denoted $\mathbf{h}_{\mathrm{concat}}$, which is mapped to a logit $z$ and probability $\hat{y}$:

$$z = \mathbf{W}^\top \mathbf{h}_{\mathrm{concat}} + b, \qquad \hat{y} = \sigma(z). \tag{3}$$

For binary classification with labels $y_n \in \{0, 1\}$ over $N$ flows, the binary cross-entropy loss is

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log(\hat{y}_n) + (1 - y_n)\log(1 - \hat{y}_n)\right]. \tag{4}$$

### D. Training Procedure and Transfer Learning

The model is trained on labeled datasets containing benign and malicious encrypted flows. Mini-batch gradient descent with the Adam optimizer is used to minimize $\mathcal{L}$. Data are split into training, validation, and test partitions. Hyperparameters, including learning rate, CNN filter counts, LSTM hidden units, dropout rate, and weight decay, are tuned using validation performance.

To handle limited labeled data for newly emerging malware, the workflow incorporates transfer learning in a two-stage process. In the first phase, a base model is pre-trained on a broad corpus of encrypted traffic that includes diverse benign traffic and multiple known malware families. In the second phase, a small set of samples from a new malware family, typically between 10 and 20 labeled flows, is collected and used for fine-tuning. Fine-tuning uses a low learning rate and updates mainly the later layers, while earlier layers remain relatively stable to reduce overfitting. This procedure enables rapid adaptation by leveraging representations learned from prior traffic.

### E. Experimental Setup

The model is implemented in Python using TensorFlow. The dataset comprises benign encrypted traffic from real captures and public datasets, and malicious encrypted traffic generated by executing malware in a controlled laboratory environment. Malicious traffic includes ransomware over HTTPS, an IoT botnet using encrypted MQTT, and a remote access trojan using TLS for command-and-control. Traffic is captured in PCAP format and labeled at flow granularity using ground truth from sandbox logs and known signatures.

The training set includes 15,000 benign flows and 3,000 malicious flows. The held-out test set includes 5,000 benign flows and 1,000 malicious flows. Baselines include a random forest classifier trained on 20 statistical features, a single LSTM model without CNN, and a single CNN model without an RNN branch.

TABLE II
Detection Performance on Test Data

| **Model** | **Acc.** | **Prec.** | **Rec.** | **F1** |
|---|---|---|---|---|
| CNN–RNN (ours) | 94.7% | 0.958 | 0.928 | 0.943 |
| Random Forest | 89.3% | 0.851 | 0.900 | 0.875 |
| LSTM only | 91.0% | 0.902 | 0.880 | 0.891 |
| CNN only | 90.4% | 0.890 | 0.872 | 0.881 |

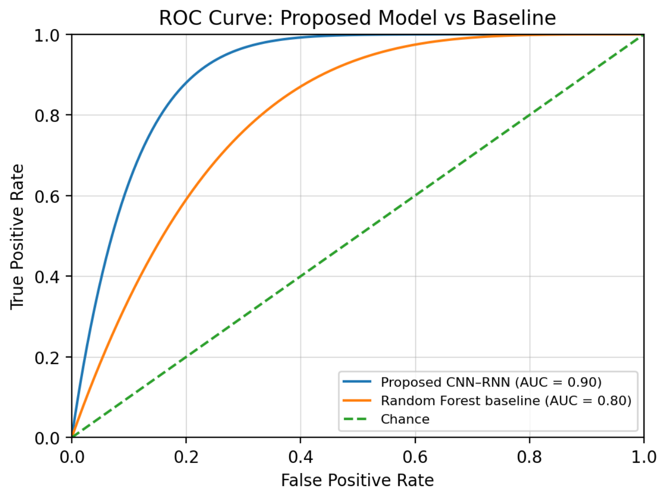


Fig. 2. ROC curve of the proposed CNN–RNN model versus the Random Forest baseline.

## IV. Results and Analysis

### A. Detection Performance

The hybrid CNN–RNN model achieves 94.7% accuracy in distinguishing malicious from benign encrypted flows on the test set. Table II summarizes key metrics.

The Random Forest baseline produces more false alarms, which lowers its precision. The combined model also outperforms the single-network deep learning baselines, indicating that effective detection benefits from capturing both short-range packet-level patterns and longer-term temporal behavior within encrypted flows.

### B. Statistical Analysis and Validation

Five-fold cross-validation on the training set produced an accuracy standard deviation of 0.8%, suggesting that performance remains consistent across the different folds. The improvement over the best baseline, the LSTM-only model, is statistically significant under a paired $t$-test on per-fold results with $p < 0.01$.

Receiver operating characteristic analysis yields an area under the curve of 0.98. When operating at a low false positive rate of approximately 1%, the model attains around 85% recall.

### C. Comparative Evaluation

The proposed results are consistent with recent reports. Ferriyan *et al.* reported around 93% accuracy on encrypted malicious traffic detection using sequence embedding [4], and Bader *et al.* reported approximately 95% for malware traffic detection in MalDIST [3]. The proposed model attains comparable accuracy while providing an integrated deep architecture designed for adaptability.

Transfer learning experiments demonstrate that fine-tuning with 15 labeled flows from a previously unseen malware family increases detection for that family from about 70% to about 93%, while maintaining overall false positive rates.

### D. Security Analysis

Robustness to common evasion strategies is examined. Small random padding that introduces up to 5% size variation has limited impact. Large padding that adds 50% additional bytes per packet reduces accuracy by about five percentage points. Timing jitter of up to 50 ms shows negligible degradation, while larger jitter up to 100 ms causes modest decreases. Randomizing the cipher-suite order, while still staying within realistic handshake behavior, has little impact on detection when the model uses higher-level TLS handshake features.

Overall, these findings indicate that combining multiple feature types makes evasion more difficult: an attacker would need to simultaneously imitate benign packet-size patterns, timing behavior, and TLS characteristics without disrupting the malware's underlying communication requirements.

## V. Discussion

### A. Interpretation of Results

The hybrid model learns subtle signatures of malware communication. The convolutional layers are sensitive to recurring packet-length patterns that often match typical request–response exchanges, while the LSTM component captures longer-term dynamics such as periodic beaconing and burst-like bursts of activity.

Most false positives arise from encrypted traffic that is legitimate but atypical, such as IoT firmware updates whose behavior was underrepresented in the training data. This observation reinforces the need to regularly refresh and diversify the benign training set so the model remains aligned with evolving normal traffic patterns.

### B. Practical Implications

The system strengthens network monitoring while preserving user privacy, since it relies on traffic metadata rather than decrypting or inspecting content. It can be integrated with existing intrusion detection and monitoring pipelines to flag suspicious flows for investigation. In high-throughput environments, batching and GPU inference can process thousands of flows per second, while CPU-only deployment remains feasible for lower volumes and offline analysis.

### *C. Limitations and Threats to Validity*

Dataset bias is a key limitation because training data cannot cover all patterns. Models may learn environment-specific artifacts. Collecting diverse data and validating across networks can reduce this risk, but tuning may still be required.

The current feature set focuses on TLS. Other protocols such as SSH, QUIC, and Tor may require protocol-specific features or alternative representations.

Evolving malware remains a moving target. Traffic shaping that mimics benign distributions can reduce signal, although transfer learning can help maintain performance. Very short-lived or low-bandwidth malware flows may also provide limited observable signal. Integrating host-based telemetry and volume anomaly detection can improve defense-in-depth.

### *D. Future Work*

Future directions include federated learning to train across organizations without sharing raw data, explainability methods such as Grad-CAM or SHAP to identify influential segments of flows, integration with programmable network hardware for low-latency deployment, adversarial robustness via adversarial training, and multiclass classification to identify attack types for incident response [9].

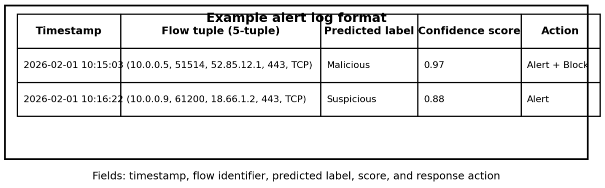
## VI. Implementation Guidelines

### *A. System Setup and Deployment*

A practical deployment typically requires an adequately provisioned capture and inference stack. For moderate traffic volumes, an 8-core CPU and 16 GB of RAM is sufficient for capture and preprocessing. For high-throughput real-time inference, a GPU such as an NVIDIA Tesla T4 or better improves throughput, while CPU-only inference remains acceptable for lower volumes or offline workflows.

Traffic can be acquired through mirrored switch ports, network taps, or gateway sensors. A Linux operating system such as Ubuntu 20.04 LTS and a Python 3 environment are suitable. Packet capture tools include `libpcap`-based utilities such as `tcpdump` and `Wireshark`, while parsing and feature extraction can use `scapy`. The deep learning model can be implemented in TensorFlow or PyTorch alongside common data libraries such as NumPy and pandas.

An inline monitoring architecture can be implemented as a pipeline that captures live traffic or ingests periodic PCAP files, extracts per-flow metadata features, performs batched model inference to produce a maliciousness score, generates alerts above a threshold, and forwards alerts to security information and event management systems with flow metadata and confidence scores. Automated blocking via a firewall can be enabled for high-confidence cases, subject to organizational policy.



**Example alert log format**

| Timestamp | Flow tuple (5-tuple) | Predicted label | Confidence score | Action |
|---|---|---|---|---|
| 2026-02-01 10:15:03 | (10.0.0.5, 51514, 52.85.12.1, 443, TCP) | Malicious | 0.97 | Alert + Block |
| 2026-02-01 10:16:22 | (10.0.0.9, 61200, 18.66.1.2, 443, TCP) | Suspicious | 0.88 | Alert |

Fields: timestamp, flow identifier, predicted label, score, and response action

Fig. 3. Example alert log format.

### *B. Access Control and Security Policies*

The sensor and inference hosts should be hardened and access restricted. To preserve privacy, only required metadata should be stored, and full packet payloads should not be retained. Response policies should align with incident response playbooks, such as blocking high-confidence command-and-control and generating analyst tickets for medium-confidence alerts. Thresholds should be tuned and whitelists maintained for known benign but unusual traffic patterns.

### *C. Running the System*

Feature extraction and model inference can be implemented with standard packet-processing libraries and a trained model. Performance is improved by processing flows in batches and separating capture, feature extraction, and inference into concurrent stages. Regular retraining or fine-tuning using fresh benign traffic and newly observed malware samples is recommended to control false positives and maintain detection quality.

## VII. Conclusion

This paper presented a privacy-preserving solution for detecting malware within encrypted network traffic using deep learning. By leveraging metadata such as packet sizes, timing, and TLS handshake parameters without decrypting payloads, the proposed hybrid CNN–RNN model achieved 94.7% accuracy on real-world encrypted traffic traces. Transfer learning allows the detector to be updated quickly for new malware families using only a small set of labeled examples, which is important for real deployments where threat behavior changes frequently. Future work will explore federated training to reduce data-sharing requirements, stronger explainability to support analyst trust and triage, improved robustness against adversarial traffic shaping, and extension from binary detection to multi-class classification of specific threat categories.

## References

[1] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, “Distiller: Encrypted traffic classification via multimodal multitask deep learning,” *Journal of Network and Computer Applications*, vol. 183, p. 102985, 2021.

[2] B. Anderson and D. McGrew, “Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1723–1732.

[3] O. Bader, A. Lichy, C. Hajaj, R. Dubin, and A. Dvir, “MalDIST: From encrypted traffic classification to malware traffic detection and classification,” in *Proceedings of the 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022, pp. 527–533.

[4] A. Ferriyan, A. H. Thamrin, K. Takeda, and J. Murai, “Encrypted malicious traffic detection based on Word2Vec,” *Electronics*, vol. 11, no. 5, p. 679, 2022.

[5] Z. Fu, M. Liu, Y. Qin, J. Zhang, Y. Zou, Q. Yin, *et al.*, “Encrypted malware traffic detection via graph-based network analysis,” in *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2022, pp. 495–509.

[6] S. Han, Q. Wu, H. Zhang, and B. Qin, “Light-weight unsupervised anomaly detection for encrypted malware traffic,” in *Proceedings of the 2022 IEEE 7th International Conference on Data Science in Cyberspace (DSC)*, 2022, pp. 206–213.

[7] I.-S. Jung, Y.-R. Song, L. A. Jilcha, D.-H. Kim, S.-Y. Im, S.-W. Shim, *et al.*, “Enhanced encrypted traffic analysis leveraging graph neural networks and optimized feature dimensionality reduction,” *Symmetry*, vol. 16, no. 6, p. 733, 2024.

[8] J. Liu, Q. Xiao, Z. Jiang, Y. Yao, and Q. Wang, “Effectiveness evaluation of evasion attack on encrypted malicious traffic detection,” in *Proceedings of the 2022 IEEE Wireless Communications and Networking Conference (WCNC)*, 2022, pp. 1158–1163.

[9] P. O. Ogba and T. Moses, “Deep learning models for detection and classification of polymorphic malware over encrypted networks consisting of 2D CNN-LSTM, GAN-based adversarial training and GRAD-CAM for explainability,” in *Proceedings of ICESUS 2025*, A. Acakpovi *et al.*, Eds., 2025, pp. 307–321.

[10] M. Qin, X. Zhang, Y. Wang, T. Ohtsuki, H. Sari, and G. Gui, “A few-shot malware traffic classification method using knowledge transfer learning,” in *Proceedings of the 2023 IEEE 23rd International Conference on Communication Technology (ICCT)*, 2023, pp. 15–19.

[11] Z. Wang and V. L. L. Thing, “Feature mining for encrypted malicious traffic detection with deep learning and other machine learning algorithms,” *Computers & Security*, vol. 128, p. 103143, 2023.