# OpCode-Based Malware Classification Using Machine Learning and Deep Learning Techniques

Varij Saini, Rudraksh Gupta, Neel Soni
Students, Cybersecurity and Threat Intelligence
University of Guelph

April 1, 2025

## Executive Summary

This technical report presents a comprehensive analysis of malware classification using OpCode sequences. Two distinct approaches are evaluated: traditional machine learning using n-gram analysis with Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Decision Tree classifiers; and a deep learning approach employing a Convolutional Neural Network (CNN). The traditional machine learning approach establishes a baseline using handcrafted 1-gram and 2-gram features from disassembled malware samples. The deep learning methodology builds upon the work proposed in "Deep Android Malware Detection" by McLaughlin et al. and evaluates the performance of a CNN model trained to automatically extract features from raw OpCode data. Empirical results are compared using standard performance metrics (accuracy, precision, recall, and F1-score). While the SVM classifier outperforms other traditional techniques, the CNN model demonstrates competitive performance with the added benefit of automated feature extraction.

## 1 Introduction

In an era of increasingly sophisticated cyberattacks, the detection and classification of malicious software, or malware, remain central to safeguarding digital infrastructure. Among the most formidable threats are Advanced Persistent Threats (APTs), which employ stealthy and prolonged campaigns to exfiltrate sensitive data and compromise critical systems [1, 2]. Traditional signature-based antivirus mechanisms have shown declining efficacy against such advanced adversaries, as malware authors continually develop obfuscation techniques to evade static and dynamic detection engines. In response, the cybersecurity research community has increasingly turned to machine learning (ML) and deep learning (DL) techniques, aiming to detect malware by uncovering subtle, intrinsic patterns within executable code [3, 4].

One promising line of inquiry involves the analysis of OpCode sequences—low-level machine instructions executed by a CPU during program execution. OpCodes offer a more granular

and robust representation of a program's behavior than higher-level features, making them less susceptible to evasion through superficial code manipulation. This project builds upon this insight, proposing a comparative evaluation of traditional machine learning algorithms and a deep learning-based model for malware classification based solely on OpCode sequences extracted from disassembled malware binaries [5, 6].

The primary goal of this research is to assess the trade-offs between handcrafted n-gram features and automated feature learning. The traditional approach employs Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Decision Tree classifiers using 1-gram and 2-gram frequency distributions as input features. These models serve as a performance benchmark for the proposed deep learning method, which utilizes a Convolutional Neural Network (CNN) architecture to directly process raw OpCode sequences. By eliminating manual feature engineering, the CNN approach aims to streamline the malware detection pipeline while uncovering latent sequence patterns that may not be captured by fixed-length n-grams [7, 8].

To ensure methodological rigor, the models are trained and evaluated using standard classification metrics—accuracy, precision, recall, and F1-score—on a labeled dataset of APT malware families. Through this comparative study, we aim to provide empirical insights into the effectiveness of OpCode-based features, the scalability of automated learning in malware classification, and the practical implications for developing real-world threat detection systems. The research not only highlights the strengths and limitations of each approach but also establishes a reproducible framework for future work in OpCode-based malware analysis[9, 10].

## 1.1 Background and Motivation

Malware continues to pose a critical threat to organizations worldwide, with evasion techniques rapidly evolving to bypass signature-based detection systems [11]. Static analysis of executable code structures, particularly through OpCode sequences, has emerged as a robust method for malware family identification. OpCodes, the machine-level instructions executed by processors, provide distinctive patterns even in obfuscated malware samples. The extraction and classification of these sequences form the foundation for effective threat intelligence and risk assessment. This project is designed to document Advanced Persistent Threat (APT) Tactics, Techniques, and Procedures (TTPs) and develop both traditional machine learning and deep learning models for detecting malicious payloads used by APT groups[12, 13].

## 1.2 Research Objectives

The study aims to:

- Evaluate the effectiveness of OpCode n-gram analysis for malware classification.
- Compare the performance of traditional machine learning algorithms (SVM, KNN, Decision Tree) on 1-gram and 2-gram features.

- Implement and evaluate a CNN-based model for OpCode-based malware classification.
- Compare the performance of the CNN model with traditional classifiers to assess the trade-offs between manual feature engineering and automated deep learning feature extraction.
- Develop a reproducible methodology for OpCode-based malware classification.

### 1.3 Scope and Limitations

The research focuses on the static analysis of OpCode sequences extracted from disassembled malware samples. Dynamic analysis is beyond this study's scope [14, 15]. While the evaluation is limited to three traditional classifiers and a CNN, further advanced models may yield improvements. The work provides empirical evidence and establishes a methodological framework for future malware analysis[16, 17].

## 2 Methodology

This study employs both traditional machine learning and deep learning techniques to classify malware based on OpCode sequences extracted from disassembled APT malware samples. The dataset comprises files labeled by malware family, from which raw OpCodes are extracted while discarding operands. For traditional models, 1-gram and 2-gram frequency features are computed and transformed into normalized feature vectors. Class imbalance is addressed using RandomOverSampler. Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Decision Tree classifiers are trained and evaluated using standard metrics[18, 19].

In parallel, a Convolutional Neural Network (CNN) is implemented to process raw OpCode sequences without manual feature engineering. The CNN architecture includes two 1D convolutional layers, max pooling, ReLU activations, dropout regularization, and fully connected layers for classification. The model is trained using PyTorch with Adam optimization and learning rate scheduling. Performance of both approaches is evaluated using accuracy, precision, recall, and F1-score to assess effectiveness and generalization.

### 2.1 Dataset Acquisition and Preprocessing

The dataset comprises disassembled OpCode files extracted from malware samples belonging to various APT families. The files are organized by malware family using filenames formatted as `[malware_family]_[sample_id].opcode`, enabling automated labeling during preprocessing.

**Traditional Machine Learning Approach**

- **OpCode Extraction:** Files are parsed to extract OpCode sequences, ignoring operands.
- **N-gram Generation:** Both 1-gram (individual OpCodes) and 2-gram (sequential pairs) features are generated.

- **Feature Vector Creation:** Frequency distributions of n-grams are used to create feature vectors.
- **Data Normalization and Class Imbalance Handling:** StandardScaler is applied to normalize features, and RandomOverSampler is used to balance class distributions.

**Deep Learning Approach**

- Preprocessed OpCode sequences are loaded from previously generated numpy arrays.
- Label encoding converts string labels to integers, and data is converted into PyTorch tensors with DataLoader objects created for batch processing.

## 2.2 Feature Extraction and Representation

Feature extraction in this study is tailored to the nature of OpCode sequences derived from disassembled malware binaries. For the traditional machine learning approach, features are created by generating n-grams—specifically 1-grams (individual OpCodes) and 2-grams (consecutive OpCode pairs). These n-grams are counted to form frequency distributions, which are then transformed into numerical feature vectors representing each malware sample. The resulting vectors are normalized using StandardScaler to ensure consistent scaling across features [20, 21].

In contrast, the deep learning approach eliminates the need for manual feature engineering. Raw OpCode sequences are preprocessed and converted into numerical arrays, which are then encoded into integer labels and transformed into PyTorch tensors. The CNN model directly processes these sequences, learning hierarchical features through convolutional layers. This method captures complex spatial patterns within OpCode sequences, enabling automated extraction of semantic features critical for malware classification. Both representations aim to preserve structural characteristics essential for accurate detection.

**Traditional Approach:**

```
def generate_ngrams(opcodes, n):
    return [" ".join(opcodes[i:i+n]) for i in range(len(opcodes) - n + 1)]
```

Both unigram and bigram frequency counts are computed and merged for feature representation.

**Deep Learning Approach:** Unlike the explicit n-gram generation used previously, the CNN model processes raw OpCode sequence data directly, eliminating the need for manual feature engineering.

## 2.3 Classification Models

This research explores two categories of classification models: traditional machine learning algorithms and a deep learning-based Convolutional Neural Network (CNN). In the traditional pipeline, three classifiers are implemented: Support Vector Machine (SVM) with a linear kernel and regularization parameter C=1, K-Nearest Neighbors (KNN) with k=3, and

a Decision Tree limited to a maximum depth of 20. Additionally, a Voting Classifier ensemble is evaluated to assess the benefit of model aggregation.

The deep learning model is a 1D CNN architecture inspired by prior work in Android malware detection. It comprises two convolutional layers with kernel size 5, followed by max pooling, ReLU activations, and a dropout layer with a rate of 0.3 to mitigate overfitting. These layers feed into fully connected layers that output class probabilities. The CNN is trained using PyTorch with the Adam optimizer and dynamic learning rate scheduling to refine performance over multiple epochs. **2.3.1 Traditional Machine Learning Models**
Three classifiers are implemented:

- **SVM:** Linear SVM with $C = 1$.
- **KNN:** Implemented with $k = 3$.
- **Decision Tree:** A Decision Tree with a maximum depth of 20.

Additionally, a hard Voting Classifier ensemble is evaluated.

**2.3.2 Convolutional Neural Network (CNN) Model**
The CNN architecture follows the methodology of McLaughlin et al.:

- **Convolutional Layers:** Two 1D convolutional layers with kernel size 5, capturing local patterns.
- **Max Pooling:** Reduces dimensionality and allows representation of variable-length OpCode sequences.
- **Activation and Dropout:** ReLU activations and a dropout rate of 0.3 to mitigate overfitting.
- **Fully Connected Layers:** Transition from convolutional features to classification outputs matching the number of malware families.

**CNN Model Architecture (Excerpt):**

```
class MalwareCNN(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(MalwareCNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=64, kernel_size=5, stride=1,
        self.conv2 = nn.Conv1d(in_channels=64, out_channels=128, kernel_size=5, stride=1
        self.maxpool = nn.MaxPool1d(kernel_size=2, stride=2)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)
        with torch.no_grad():
            sample_input = torch.randn(1, 1, input_dim)
            sample_output = self._forward_features(sample_input)
            self.fc1_input_dim = sample_output.shape[1]
        self.fc1 = nn.Linear(self.fc1_input_dim, 128)
        self.fc2 = nn.Linear(128, num_classes)
    def _forward_features(self, x):
```

```
        x = self.relu(self.conv1(x))
        x = self.maxpool(x)
        x = self.relu(self.conv2(x))
        x = self.maxpool(x)
        x = torch.flatten(x, start_dim=1)
        return x
    def forward(self, x):
        x = x.unsqueeze(1)
        x = self._forward_features(x)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

## 2.4 Training and Evaluation

**Traditional Approach:**
Models are trained on 80% of resampled and scaled feature data using stratified sampling. Evaluation is conducted using accuracy, precision, recall, F1-score, and confusion matrices.
**Results Summary**

- SVM: Accuracy 66.37%, F1-score 64.04%
- KNN: Accuracy 63.65%, F1-score 61.02%
- Decision Tree: Accuracy 62.14%, F1-score 60.06%
- Voting Classifier: Accuracy 68.61% (but did not outperform SVM).

**Deep Learning Approach:**
The CNN is trained using PyTorch for 10 epochs with the Adam optimizer (lr=0.001) and a ReduceLROnPlateau scheduler. Evaluation metrics for the CNN are:

- Accuracy: 62.14%
- Precision: 64.49%
- Recall: 62.14%
- F1-score: 60.44%

Comparative analysis indicates that while the CNN reduces the need for manual feature engineering, SVM remains the top-performing model on this dataset.

# 3 Results and Analysis

## 3.1 Traditional Machine Learning Results

Performance metrics are tabulated for SVM, KNN, Decision Tree, and the Voting Classifier. Confusion matrix analysis reveals that SVM has the most balanced performance across malware families.

## 3.2 CNN Performance

The CNN model shows rapid initial convergence with diminishing improvements after the fifth epoch. Final performance registers an accuracy of 62.14%, with precision, recall, and F1-scores closely matching the Decision Tree classifier.

# 4 Discussion

## 4.1 Comparative Analysis

- **Traditional vs. CNN:** SVM outperforms other traditional models and remains superior in overall classification performance. The CNN model offers benefits in automated feature learning but is limited on smaller datasets.
- **Feature Engineering Trade-Offs:** Manual n-gram generation captures detailed sequence relationships, while CNN eliminates this need but may require larger datasets and further architectural refinement.

## 4.2 Practical Implications

- **Threat Intelligence Applications:** The ability to classify malware into distinct APT families aids in precise attribution and response.
- **Operational Integration:** Both approaches offer insights into building robust, automated detection systems, with SVM serving as a strong baseline and deep learning offering potential for future scaling.

# 5 Future Road Map

Building on the foundational work of OpCode-based malware classification, this project lays the groundwork for several promising research and development directions aimed at enhancing threat detection capabilities. As cyber threats continue to evolve, particularly from sophisticated Advanced Persistent Threat (APT) groups, it is imperative to develop adaptive, scalable, and intelligent systems that can offer real-time analysis and high-fidelity classification. The following roadmap outlines future milestones and key enhancements envisioned to improve the current methodology and expand its practical applicability [20].

1. Integration of Dynamic and Hybrid Analysis: While this project focused solely on static OpCode sequences, incorporating dynamic features—such as system call traces, network behavior, and runtime memory usage—can provide richer context and improve classification accuracy. A hybrid model that fuses static OpCode features with dynamic behavioral indicators could enable more robust malware detection, particularly for evasive or polymorphic threats.

2. Expansion of Dataset and Malware Families: The effectiveness of machine learning models is often tied to the diversity and size of the training dataset. In future iterations, the dataset will be expanded to include a broader range of malware samples from public repositories (e.g., VirusShare, VirusTotal) and threat intelligence platforms. Incorporating additional malware families, especially those from emerging APT groups, will improve the generalizability of the models and allow for real-world validation.

3. Advanced Deep Learning Architectures: To enhance classification accuracy, future work will explore more sophisticated deep learning architectures such as Recurrent Neural Networks (RNNs), Bidirectional LSTMs, and Transformer-based models that are better suited for capturing long-range dependencies in sequential data. Moreover, exploring lightweight architectures like MobileNet or quantized CNNs can facilitate deployment in resource-constrained environments such as edge devices and embedded security appliances.

4. Explainability and Model Interpretability: As AI models are increasingly integrated into security operations, explainability becomes essential. Future work will involve the use of explainable AI techniques—such as SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations)—to provide insight into the decision-making process of both traditional and deep learning models. This will help cybersecurity analysts trust and validate the classification outputs, and possibly identify new malicious OpCode patterns.

5. Transfer Learning and Pretrained Embeddings: Another strategic direction is the application of transfer learning, where models pre-trained on large corpora of assembly or OpCode sequences can be fine-tuned on smaller, task-specific malware datasets. This approach may significantly improve performance in low-data scenarios and accelerate training.

6. Real-Time Detection and Deployment: The long-term goal is to integrate the developed models into a real-time malware detection pipeline. This would involve optimizing inference time, reducing false positives, and ensuring compatibility with endpoint security solutions. Additionally, deploying the model within a cloud-native or containerized framework would enable scalable and distributed analysis of incoming files in enterprise environments.

In summary, the future roadmap aims to advance this research from a controlled experimental setting toward a practical, enterprise-ready malware detection solution. By leveraging advanced ML/DL models, dynamic threat signals, and explainable AI, this research will contribute to the development of next-generation cybersecurity defenses tailored for APT detection and beyond.

# 6 Conclusion and Future Work

This research presents a comparative study of OpCode-based malware classification using both traditional machine learning algorithms and a deep learning-based CNN. By leveraging static analysis techniques, specifically focusing on OpCode sequences extracted from disassembled malware samples, we investigated the effectiveness of both manually engineered features (n-grams) and automated feature learning. Among the traditional classifiers, the SVM demonstrated the highest accuracy and overall performance, with an F1-score of 64.04%, outperforming K-Nearest Neighbors and Decision Tree classifiers. The SVM model's robustness can be attributed to its ability to find optimal hyperplanes in high-dimensional feature spaces, making it particularly suitable for n-gram frequency data. The Voting Classifier ensemble offered modest performance gains but did not surpass SVM alone. The CNN model, while slightly underperforming compared to the SVM in terms of accuracy (62.14%), offers significant advantages in terms of automation and scalability. By eliminating the need for manual feature engineering, the CNN streamlines the malware classification pipeline and has the potential to uncover deeper, hierarchical patterns in OpCode sequences. However, its performance was likely constrained by dataset size and model capacity, suggesting that deeper architectures or larger training datasets may improve results. This study establishes that OpCode-based static analysis remains a viable approach for malware detection and classification, especially when paired with efficient feature engineering or deep learning architectures. Both traditional and deep learning methods have demonstrated their utility in classifying malware families associated with Advanced Persistent Threats (APTs), offering practical insights for security analysts and automated defense systems. For future work, several avenues can be pursued. First, the exploration of more advanced deep learning architectures, such as Transformers or hybrid CNN-LSTM models, could improve classification performance and better capture long-range dependencies in OpCode sequences. Second, integrating dynamic analysis features (e.g., API calls, system behavior traces) could enhance detection accuracy by combining both static and behavioral perspectives. Third, leveraging transfer learning or pre-trained embeddings for OpCodes may offer improvements in generalization, especially across diverse malware families. Moreover, incorporating explainable AI (XAI) techniques—such as SHAP or LIME—could improve transparency in decision-making, aiding analysts in understanding which features or OpCode patterns are most influential in classification. Expanding the dataset and including real-world samples from emerging threats would further validate the robustness of the proposed models.

# Additional Information

- **Implementation Details:** Full code implementations are available in the accompanying files for both traditional machine learning pipelines and the CNN model training process.
- **References:** Key literature including McLaughlin et al. (2017), Bilar (2007), among others, supports the methodologies used.

# References


[1] A. Yazdinejad, A. Dehghantanha, R. M. Parizi, and G. Epiphaniou, "An optimized fuzzy deep learning model for data classification based on nsga-ii," *Neurocomputing*, vol. 522, pp. 116–128, 2023.

[2] J. Sakhnini, H. Karimipour, A. Dehghantanha, A. Yazdinejad, T. R. Gadekallu, N. Victor, and A. Islam, "A generalizable deep neural network method for detecting attacks in industrial cyber-physical systems," *IEEE Systems Journal*, vol. 17, no. 4, pp. 5152–5160, 2023.

[3] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, Q. Zhang, and K.-K. R. Choo, "An energy-efficient sdn controller architecture for iot networks with blockchain-based security," *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp. 625–638, 2020.

[4] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, H. Karimipour, G. Srivastava, and M. Aledhari, "Enabling drones in the internet of things with decentralized blockchain-based security," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6406–6415, 2020.

[5] B. Zolfaghari, A. Yazdinejad, A. Dehghantanha, J. Krzciok, and K. Bibak, "The dichotomy of cloud and iot: Cloud-assisted iot from a security perspective," *arXiv preprint arXiv:2207.01590*, 2022.

[6] M. Aledhari, R. Razzak, M. Rahouti, A. Yazdinejad, R. M. Parizi, B. Qolomany, M. Guizani, J. Qadir, and A. Al-Fuqaha, "Safeguarding connected autonomous vehicle communication: Protocols, intra-and inter-vehicular attacks and defenses," *Computers & Security*, p. 104352, 2025.

[7] A. Yazdinejad, A. Dehghantanha, R. M. Parizi, M. Hammoudeh, H. Karimipour, and G. Srivastava, "Block hunter: Federated learning for cyber threat hunting in blockchain-based iiot networks," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 11, pp. 8356–8366, 2022.

[8] H. Nazari, A. Yazdinejad, A. Dehghantanha, F. Zarrinkalam, and G. Srivastava, "P3gnn: A privacy-preserving provenance graph-based model for autonomous apt detection in software defined networking," in *Proceedings of the Workshop on Autonomous Cybersecurity*, 2023, pp. 34–44.

[9] F. Nelles, A. Yazdinejad, A. Dehghantanha, R. M. Parizi, and G. Srivastava, "A federated learning approach for multi-stage threat analysis in advanced persistent threat campaigns," *arXiv preprint arXiv:2406.13186*, 2024.

[10] A. Yazdinejad, A. Dehghantanha, R. M. Parizi, G. Srivastava, and H. Karimipour, "Secure intelligent fuzzy blockchain framework: Effective threat detection in iot networks," *Computers in Industry*, vol. 144, p. 103801, 2023.

[11] D. Namakshenas, A. Yazdinejad, A. Dehghantanha, and G. Srivastava, "Federated quantum-based privacy-preserving threat detection model for consumer internet of things," *IEEE Transactions on Consumer Electronics*, 2024.

[12] Y. Hailemariam, A. Yazdinejad, R. M. Parizi, G. Srivastava, and A. Dehghantanha, "An empirical evaluation of ai deep explainable tools," in *2020 IEEE Globecom Workshops (GC Wkshps*. IEEE, 2020, pp. 1–6.

[13] A. Yazdinejad, R. M. Parizi, A. Bohlooli, A. Dehghantanha, and K.-K. R. Choo, "A high-performance framework for a network programmable packet processor using p4 and fpga," *Journal of Network and Computer Applications*, vol. 156, p. 102564, 2020.

[14] A. Yazdinejad, A. Bohlooli, and K. Jamshidi, "Efficient design and hardware implementation of the openflow v1. 3 switch on the virtex-6 fpga ml605," *The Journal of Supercomputing*, vol. 74, pp. 1299–1320, 2018.

[15] J. Sakhnini, H. Karimipour, A. Dehghantanha, A. Yazdinejad, T. R. Gadekallu, N. Victor, and A. Islam, "A generalizable deep neural network method for detecting attacks in industrial cyber-physical systems," *IEEE Systems Journal*, vol. 17, no. 4, pp. 5152–5160, 2023.

[16] A. Yazdinejad, E. Rabieinejad, A. Dehghantanha, R. M. Parizi, and G. Srivastava, "A machine learning-based sdn controller framework for drone management," in *2021 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2021, pp. 1–6.

[17] A. Yazdinejad, A. Dehghantanha, and G. Srivastava, "Ap2fl: Auditable privacy-preserving federated learning framework for electronics in healthcare," *IEEE Transactions on Consumer Electronics*, 2023.

[18] A. Yazdinejad, A. Dehghantanha, G. Srivastava, H. Karimipour, and R. M. Parizi, "Hybrid privacy preserving federated learning against irregular users in next-generation internet of things," *Journal of Systems Architecture*, vol. 148, p. 103088, 2024.

[19] A. Yazdinejad, A. Dehghantanha, H. Karimipour, G. Srivastava, and R. M. Parizi, "A robust privacy-preserving federated learning model against model poisoning attacks," *IEEE Transactions on Information Forensics and Security*, 2024.

[20] T. Crawford, S. Duong, R. Fueston, A. Lawani, S. Owoade, A. Uzoka, R. Parizi, and A. Yazdinejad, "Ai in software engineering: A survey on project management applications. arxiv," *arXiv preprint arXiv:2307.15224*, 2023.

[21] A. Yazdinejad, "Secure and private ml-based cybersecurity framework for industrial internet of things (iiot)," Ph.D. dissertation, University of Guelph, 2024.