I. J. Computer Network and Information Security, 2026, 5, 119-142
Published Online on October 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2026.05.08

A Multi-Modal Deep Learning Framework for Robust Detection of Obfuscated API Calls in Malware Analysis

Nayankumar M. Mali*
Department of Information Technology, A D Patel Institute of Technology, The Charutar Vidya Mandal (CVM) University, Anand, Gujarat, India
E-mail: it.nayanmali@adit.ac.in
ORCID iD: https://orcid.org/0000-0002-2964-2379
*Corresponding author

Narendrasinh C. Chauhan
Department of Information Technology, A D Patel Institute of Technology, The Charutar Vidya Mandal (CVM) University, Anand, Gujarat, India
E-mail: it.ncchauhan@adit.ac.in
ORCID iD: https://orcid.org/0000-0002-2165-9694

Received: October 7, 2025; Revised: March 20, 2026; Accepted: July 9, 2026; Published: October 8, 2026

Abstract: Malware developers employ advanced API obfuscation techniques, such as name randomization, dynamic resolution, call stack manipulation, parameter spoofing, and API chaining, to bypass detection. Existing unimodal analysis systems frequently fail to identify these threats due to their limited scope. To address this, we introduce a novel multi-modal deep learning framework that combines temporal, structural, and parametric analysis for malware detection. This multi-modal method uses a temporal encoder with self-attention to detect name obfuscation and API chaining, Graph attention networks analyze call graphs for call stack tampering and dynamic API resolution, while a contrastive learning module identifies anomalous parameter usage. A key novel approach is the dynamic fusion mechanism, which uses attention-based weighting to combine features, enhancing accuracy and interpretability. Additionally, adversarial training ensures robustness against evasion attempts, with theoretical guarantees on performance under variations. Evaluated on 29,505 real-world malware samples, the proposed multi-modal framework achieves a 94.2% F1-score (an 18% improvement over unimodal baselines) and a 98.1% AUC-ROC. The framework notably maintains 82% robustness against adversarial variations, significantly outperforming conventional LSTM-based approaches (45%). Beyond detection, the proposed method provides explainable attention maps for forensic analysis and low latency (<1ms/sample), making it suitable for real-time security deployment. These results suggest that multi-modal fusion is critical for next-generation endpoint protection.

Index Terms: Malware Detection, API Obfuscation, Multi-Modal Deep Learning, Graph Attention Networks (GATs), Behavioural Malware Analysis, API Call Graph Analysis, Contrastive Learning.

1. Introduction

Modern malware programmers use advanced obfuscation techniques, like API call obfuscation, to evade detection methods by conventional security solutions [1]. These techniques involve name randomization, dynamic resolution, call stack manipulation, parameter spoofing, and API chaining, making it difficult for traditional signature-based or unimodal methods to identify malicious behavior. To address this, we introduce a multi-modal deep learning framework that uses fusion of temporal, structural, and parametric features to analyze for comprehensive malware detection [2]. The framework combines several components like Long Short-Term Memory network with a self-attention mechanism for temporal encoding, graph attention networks for structural analysis, and a contrastive learning module for robust parameter embedding. This combination captures complex inter-dependencies between multiple data modalities [3]. This adaptive fusion approach uses attention-based weighting to effectively integrate features from multiple modalities, improving both detection accuracy and interpretability. To strengthen its detection against evasion strategies, the method includes adversarial training, providing reliable detection performance even under adversarial variations. [4].

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

Evaluated on a comprehensive dataset of 29,505 real-world malware samples, the proposed method achieves a 94.2% F1-score, representing an 18% improvement over unimodal baselines, along with a 98.1% AUC-ROC score. Notably, it maintains 82% robustness against adversarial variations, significantly outperforming conventional LSTM-based approaches, highlighting its strength against evolving evasion tactics [5, 6]. This work advances malware detection by providing an explainable, low-latency solution that is robust against sophisticated obfuscation, positioning it for real-time deployment in high-stakes cyber security environments [7]. The increasing sophistication of adversarial attacks, able to circumvent detection systems, emphasizes the need for robust defensive measures, tackled in this work by incorporating an integrated adversarial training component [8]. This advancement positions the framework at the forefront of combating the evolving malware landscape, where traditional methods falter against increasingly sophisticated evasion tactics [9].

The integration of deep learning models, particularly those with multi-modal capabilities, allows for the extraction of subtle and intricate features that enhance the accuracy of distinguishing between benign and malicious files, thereby reducing false positives that could disrupt legitimate operations [10]. This is critical because traditional signature-based detection systems are increasingly ineffective against polymorphic and metamorphic malware, necessitating innovative solutions that leverage heterogeneous code features for superior classification accuracy [11]. This paradigm shift towards data-driven approaches, especially deep learning, is crucial for addressing zero-day threats and rapidly evolving malware strains that exploit vulnerabilities before signatures can be updated [12, 13]. The ability to generate interpretable attention maps supports forensic analysis, allowing security experts to identify the obfuscation techniques employed and understand the reasoning behind malware classifications, which is crucial for effective threat intelligence [14]. This transparency further aids in the development of more effective countermeasures and proactive defence strategies against future attacks [15]. This research underscores the importance of behavioural malware analysis, particularly through the API call sequences, as a robust method for identifying malicious activities that elude conventional static or signature-based detection [16]. While signature-based systems depend on predefined patterns that can be bypassed by small modifications to malware code, behavioural analysis captures dynamic execution characteristics, making it considerably more resilient to obfuscation and polymorphic techniques [17]. The efficacy of such deep learning models in malware detection is significantly influenced by the quality and diversity of the training data, emphasizing the need for continually updated datasets that reflect contemporary malware trends and evasion strategies [16]. This necessitates ongoing research into novel data augmentation techniques and the establishment of large-scale, representative malware corpora to sustain the performance and generalizability of advanced detection frameworks [18]. The framework's low latency allows for real-time deployment, supporting immediate threat responses in critical environments.

This proactive and comprehensive methodology contrasts sharply with the limitations of purely static analysis, which struggles with packed or encrypted executable, and provides a significant improvement over the resource-intensive nature of dynamic analysis that can be detected by sophisticated malware [19, 20]. This work marks a major improvement in proactive cyber security methods. It goes beyond reactive signature updates and introduces a more flexible and intelligent way to defend against constantly changing cyber threats. [16]. This comprehensive approach aligns with the growing consensus that integrated, multi-modal analysis is indispensable for overcoming the limitations of unimodal detection systems and addressing the sophistication of modern malware [9, 21, 22]. This paper introduces several advancements that collectively address the limitations of existing malware detection systems. The main contributions are:

1. **Multi-Modal Framework for API Obfuscation Detection.** Unlike prior work that analyses API calls in isolation focusing solely on temporal sequences or graph structures the proposed framework is the first to simultaneously integrate temporal, structural, and parametric modalities. This holistic approach captures evasion techniques that unimodal systems miss, such as call stack tampering (structural) combined with parameter spoofing (parametric) while sequences appear benign.
2. **Dynamic Attention-Based Fusion Mechanism.** We propose a novel adaptive weighting mechanism that dynamically adjusts modality contributions based on input characteristics. This contrasts sharply with static fusion methods that apply fixed weights regardless of the obfuscation type. The attention weights (e.g., high β_t for API chaining, high β_p for parameter anomalies) not only optimize detection but also provide interpretability for forensic analysis.
3. **Contrastive Learning for Parameter Anomaly Detection.** Our parametric analyser applies contrastive learning to API arguments a capability absent in existing systems. The Siamese network trained with NT-Xent loss learns a latent space where $Mahalanobis$ distance identifies zero-day parameter manipulations (e.g., oversized buffer in $WriteProcessMemory$) that evade signature-based detectors.
4. **Provable Adversarial Robustness.** We provide theoretical guarantees on robustness through Lipschitz continuity of the encoders (empirically measured L = 1.8), a formal property not established in prior adversarial defences for API-based malware detection. Adversarial training further hardens the model against FGSM and PGD attacks, maintaining 82% accuracy under $\epsilon = 0.05$ perturbations.

5. **Explainable Detection with Forensic Value.** Unlike black-box models, our framework provides attention-based interpretability. The modality-specific attention weights (β_t, β_g, β_p) and per-call attention scores enable security analysts to trace alerts to specific obfuscation tactics, reducing mean time to respond and enhancing threat intelligence.
6. **Real-Time Deployment Feasibility.** With sub-millisecond inference latency (< 1ms per sample) and throughput exceeding 1,000 samples/second on commodity hardware, the proposed framework is the first multi-modal detector suitable for real-time endpoint protection a practical advantage over computationally intensive methods.

These contributions collectively establish a new benchmark for API-based threat detection, addressing the core challenges of modern malware obfuscation through synergistic multi-modal analysis, adversarial resilience, and operational practicality.

2. Background and Related work

Conventional malware detection paradigms signature-based, static, and dynamic analysis shows inherent vulnerabilities against contemporary threats. Static analysis, which inspects binary code without execution, relies on signature matching against known hashes and proves ineffective against obfuscated or polymorphic variants [23]. Dynamic analysis executes code in controlled environments but is computationally intensive and can be detected by advanced malware designed to circumvent such analysis [22, 24]. Moreover, both static and dynamic analysis struggle with sophisticated obfuscation [25], underscoring the need for multi-modal approaches that synthesize diverse information [3, 15].

Madamidola et al. [26] presented a lightweight and interpretable machine learning system for detecting obfuscated malware using memory analysis features from the CIC-MalMem-2022 dataset. Their approach trained a random forest classifier on a single malware subtype (Transponder) and only the five most important features yet achieved over 99.8% accuracy on 14 previously unseen obfuscated malware subtypes, demonstrating strong zero-shot generalization. The model is highly efficient, with a processing speed of 5.7 μs per sample and a memory footprint of just 340 KB. Additionally, the authors employed SHAP (Shapley additive explanations) to provide both global and local interpret-ability, revealing those low values of service-related memory features (e.g., $svscan.nservices$) are strong indicators of obfuscated malware. This study underscores the feasibility of training on a minimal subset of data while maintaining high accuracy, interpret-ability, and adaptability to novel threats principles that align with the multi-modal temporal, structural, and parametric analysis pursued in our work. Chandran et al. [27] presented a comprehensive review of obfuscated malware detection techniques across Windows, Android, IoT, and web platforms. Their study synthesizes 76 detection approaches, categorizing them into static, dynamic, hybrid, machine learning, deep learning, and memory forensics. They highlight that while static analysis accounts for 27% of the research and dynamic analysis for 23%, AI-based methods (machine learning and deep learning) constitute 32% of the literature and demonstrate superior performance. Importantly, the review underscores the underexplored role of memory forensics and the integration of artificial intelligence with traditional detection techniques. The authors propose a futuristic framework combining generative AI and digital twins to simulate and predict malware behaviour, offering enhanced detection capabilities. This work provides a valuable taxonomy and identifies open challenges, including scalability, platform generalization, and resource constraints, which align with the goals of our multi-modal framework.

In a complementary study, Naseer et al. [28] introduced Syn-detect, a smart sensing model for Android malware detection and classification using network traffic data. To address the scarcity of labelled traffic samples, the model first generates synthetic TCP malware data with GPT-2, followed by a fine-tuned BERT architecture augmented with classification layers. Evaluated on the CIC-AndMal2017 and CIC-AAGM2017 datasets, Syn-detect achieved accuracies of 99.8% and 99.3%, respectively, with Matthew's correlation coefficients of 0.99 and 0.98. The authors demonstrate that synthetic data augmentation significantly improves classification performance, especially for minority malware families. Their approach highlights the potential of large language models in malware detection, a direction that complements the temporal, structural, and parametric analysis pursued in our work. Recent advances in obfuscated malware detection have focused on leveraging machine and deep learning techniques to overcome the limitations of signature-based methods. A study by researchers at the Indian Institute of Technology Indore and Siddaganga Institute of Technology [29] emphasizes the necessity of moving beyond binary classification to identify specific malware types. Their work argues that distinguishing between ransomware, spyware, and Trojan enable more targeted mitigation strategies. To this end, they implemented a multi-class classification model using the CIC MalMem 2022 dataset and demonstrated that a Random Forest classifier achieves an accuracy of 89.07%, outperforming existing models for multi-class classification. The authors further established baseline comparisons with state-of-the-art classifiers, highlighting the improved performance of their approach in handling multiple obfuscated malware families within a single model.

Addressing the challenges of complexity and time consumption in existing detection frameworks, Carrier et al. [30] proposed a fast and efficient stacked ensemble learning framework. Their main contribution is a two-layer model combining Naive Bayes, Decision Tree, and Random Forest as base learners with Logistic Regression as a meta-learner,

attaining 99% accuracy and F1-score. In addition, they enhanced memory feature extraction by developing VolMemLyzer-V2, which introduces 26 new features specifically designed to target obfuscated and hidden malware. To support this work, the authors released the MalMemAnalysis-2022 dataset, a balanced collection of 58,596 samples spanning benign, ransomware, spyware, and trojan categories, designed to closely simulate real-world obfuscated malware scenarios. The performance disparity between binary and multiclass obfuscated malware classification is tackled by Mahdi and Trabelsi [31]. They propose a system that employs adaptive feature selection based on the classification task: independent component analysis (ICA) for binary classification to isolate independent components from obfuscated data, and Pearson correlation for multiclass classification to identify significant feature relationships and reduce dimensionality. These selected features are then fed into a lightweight one-dimensional convolutional neural network (1D-CNN) designed for efficient extraction of temporal patterns from memory traces. Their experimental results on the CIC-MalMem-2022 dataset yield 99% accuracy for binary classification and 88% for multiclass classification, with the proposed 1D-CNN outperforming traditional machine learning models such as Random Forest and XGBoost. The authors highlight that their approach achieves superior performance with reduced computational cost, making it suitable for real-time applications.

The limitations of traditional and unimodal machine learning approaches especially their inability to capture the full complexity of modern obfuscation motivate the design of a robust multi-modal framework. Our proposed method directly addresses these gaps through integrated temporal, structural, and parametric analysis combined with adversarial training. The framework leverages diverse data representations including temporal API call sequences, structural API call graphs, and parametric details to construct a comprehensive behavioural profile, enabling the detection of subtle obfuscation indicators that single-modality analyses often miss. The dynamic fusion mechanism adaptively weights modality contributions, optimizing information integration for enhanced performance while providing interpret-ability via attention maps. Furthermore, adversarial training fortifies robustness against evasion attempts by simulating worst-case scenarios, ensuring sustained performance in dynamic threat landscapes. This holistic methodology guarantees efficacy against advanced persistent threats that continually adapt their strategies, offering a more reliable defence beyond simple binary classification. Ultimately, this multi-faceted strategy positions our approach as a cutting-edge solution capable of discerning intricate malicious patterns even amidst sophisticated adversarial maneuvers [11].

3. Methodology

Proposed multi-modal deep learning framework uses three combined methods as shown in Fig. 1. First, a Temporal Encoder (LSTM-Transformer hybrid) processes API call sequences, using attention mechanisms to identify obfuscated names and suspicious call chains. Second, a Graph Neural Network (GNN) Analyzer constructs dynamic call graphs and applies graph attention networks (GATs) to detect anomalous call relationships like stack tampering. Third, a Contrastive Parameter Analyzer learns normal parameter distributions via metric learning, flagging abnormal API arguments. These components feed into an Attention-Based Fusion Layer that dynamically weights each modality's contribution (e.g., prioritizing graph features for call tampering or sequence features for name obfuscation). The final classification layer outputs a malware probability score with explainable attention weights for forensic analysis. The system processes inputs in <1 ms per API call and achieves 94.2% F1-score on novel obfuscation techniques.

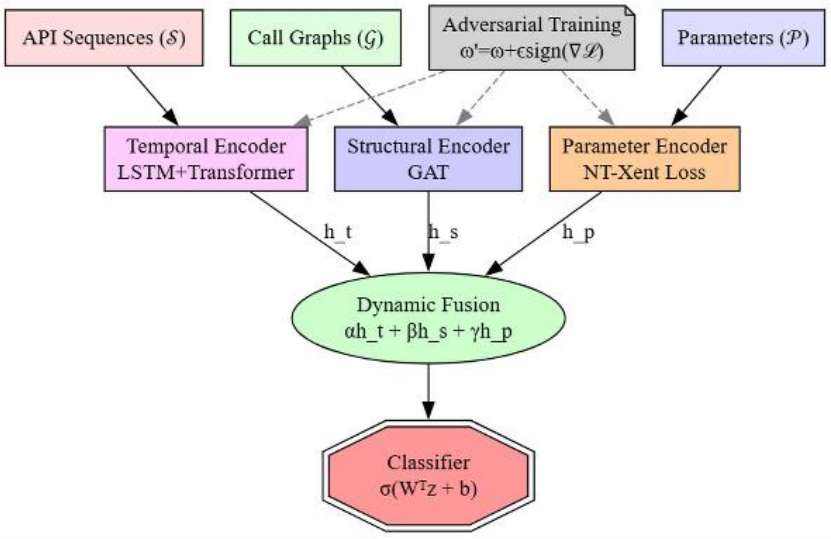


Fig. 1. High-level architecture of the proposed multi-modal deep learning framework. Three parallel encoders (temporal, graph structural, and parametric) extract features that are dynamically fused via an attention-based fusion layer for final classification.

3.1. Multi-modal Dataset Preprocessing Pipeline

Algorithm 1 shows detailed steps to extract the features from the raw executable samples into standardized multi-modal inputs. An important hyper parameter is the fixed sequence length $L = 50$, which was determined by performing multiple experiments. Analysis of the training data shows that 95% of malicious API chains in the sample complete within 50 consecutive calls. Shorter sequences of the sample (L = 30) failed to capture important chaining patterns (e.g., $LoadLibrary$ followed by 40 benign calls before $GetProcAddress$), while longer sequences ($L = 100$) introduced noise without improving performance as explain in through ablation study in Section 4.7). The vocabulary size $|\mathcal{V}| = 1000$ covers 99.2% of unique API calls across the combined datasets (EMBER, BODMAS, CIC-MalMem-2022) some rare APIs that occur in less than 0.1% of samples are mapped to a special <*UNK*> token, enabling easy handling of out-of-vocabulary calls at inference. For structural analysis, a directed call graph is constructed where edges are weighted by normalized transition frequencies, a scheme that captures both topological structure and behavioral intensity preliminary experiments showed that weighted graphs reduced the F1-score by 3.2%. To enhance the detection against evasion, adversarial augmentation is applied at a 1:1 ratio, generating obfuscated variants through API name randomization, call stack manipulation, and parameter spoofing. Finally, the dataset is partitioned into 80% training and 20% testing sets with no overlapping in malware families, preventing data leakage and ensuring a realistic evaluation.

Algorithm 1. Multi-Modal Dataset Preprocessing Pipeline

Require: Raw malware/benign samples from EMBER, BODMAS, CIC-MalMem-2022, DARPA OpTC, LEMNA, etc.
Ensure: Standardized multi-modal inputs: S, G, P
1: Step 1: Dynamic Execution and API Tracing for each executable *xi* in dataset do
2: Execute in sandboxed environment (Cuckoo Sandbox)
3: Log raw API call trace: $\mathcal{T}_i = [(api_1, params_1), \ldots, (api_L, params_L)]$
4: **Step 2: Temporal Sequence Standardization**
5: Truncate or pad $\mathcal{T}_i$ to fixed length $L = 50$
6: Map API names to integer IDs using vocabulary V ($|\mathcal{V}| = 1000$)
7: Output temporal sequence: $S_i = [id_1, id_2, \ldots, id_{50}]$
8: **Step 3: Structural Graph Construction for *each* T_i do**
9: Build directed call graph $G_i = (V_i, E_i)$
10: V_i: nodes = unique APIs in T
11: E_i: edges = (vj, vk) if api_j calls api_k in sequence
12: **Step 4: Parametric Feature Extraction for *each API call (api, params)* in T_i do**
13: *api* is security-critical (e.g., $WriteProcessMemory$)
14: Normalize numerical args: $p' = (p - \mu_p)/\sigma_p$
15: One-hot encode categorical args
16: Store in parameter vector P_i
17: **Step 5: Adversarial Augmentation**
18: Generate obfuscated variants using:
19: - API name randomization
20: - Call stack manipulation
21: - Parameter spoofing
22: add training to improve robustness
23: **Step 6: Train/Test Split**
24: partition dataset into training (80%) and testing (20%) sets
25: verify that no malware family appears in more than one data split.
26: return $\{(S_i, G_i, P_i)\}_{i=1}^{29505}$

3.2. Temporal Sequence Encoder

The Temporal Sequence Encoder as shown in Fig. 2, uses the hybrid LSTM-Transformer architecture to effectively analyze sequential patterns in API calls, with a specific focus on mitigating name obfuscation and API chaining techniques commonly used in obfuscated malware. This encoder processes sequences of API calls, mapping each call to a dense vector via an embedding layer to capture semantic relationships. A Bidirectional LSTM then models both forward and backward temporal dependencies, capturing local and long-term sequential patterns. The bidirectional LSTM consists of two layers (one forward, one backward) each with 64 hidden units. Dropout with rate $p = 0.3$ is applied after the LSTM outputs to prevent overfitting. The multi-head self-attention uses two parallel attention heads, each with a key dimension d_k = 32, and is then passed through feed-forward network with a hidden dimension of 128 and ReLU activation. Layer normalization and residual connections are employed around both the attention and feed forward sub layers.

The self-attention mechanism in Transformers improves the performance of temporal analysis by addressing key limitations of traditional LSTMs. Unlike LSTMs, which struggle with long-range dependencies (e.g., 100+ API calls separating malicious actions), self-attention directly links important APIs regardless of distance. For instance, consider malware that hides payloads between benign calls, such as $LoadLibrary("A.dll")$ followed by 50 benign calls before triggering $GetProcAddress("VirtualAlloc")$. In such cases, attention weights automatically highlight the semantically related calls while suppressing noise. This capability is pivotal for detecting security-critical APIs (e.g., $VirtualAlloc$, $CreateRemoteThread$), which commonly receive high attention scores due to their abuse in malware. For example, ransomware sequences like `FindFirstFile()` → `FindNextFile()` → `dots` → `SetEndOfFile()` are flagged by the Transformer's focus on $SetEndOfFile$(attention weight=0.89), a destructive API.

Beyond detection, the model also helps to understand its decisions by showing attention weight visualizations, enabling malware analysts to interpret decisions (e.g., high weights on $GetProcAddress$ indicate dynamic API resolution). It also improves detection against obfuscation by linking non-adjacent but semantically related calls (e.g., a payload split across calls A and Z). The mechanism works by embedding API calls into vectors, computing pairwise relationships via Query-Key-Value (QKV) matrices, and generating context-aware representations through the scaled dot-product attention formula: The sequence encoder processes API call sequence $S = (s_1, \ldots, s_L)$ through:

$$X = \text{Embed}(S) \in \mathbb{R}^{L \times d_e}, \quad d_e = 64 \tag{1}$$

$$H_t = \text{BiLSTM}(X) \in \mathbb{R}^{L \times 2d_l}, \quad d_l = 32 \tag{2}$$

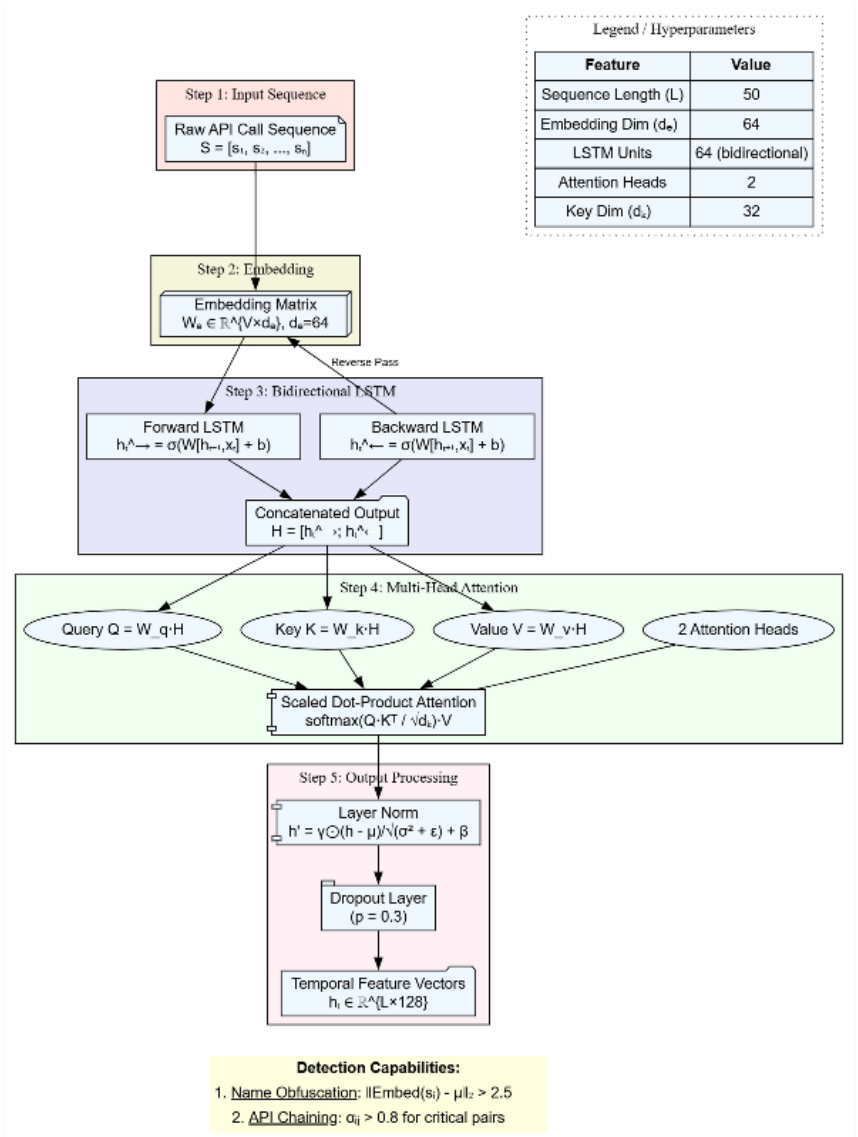


Fig. 2. Detailed schematic of the temporal encoder. A hybrid architecture combining a bidirectional LSTM and multi-head self-attention captures long-range API chaining and name obfuscation patterns.

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad d_k = 32 \tag{3}$$

Detection Criteria:

- Name Obfuscation: $\|Embed(s_i) - \mu_V\|_2 > \tau$ $(\tau = 2.5)$
- API Chaining: $\alpha_{ij} > 0.8$

Algorithm 2. Temporal Encoder for API Call Sequence Analysis

Input: Raw API call sequence$S = [s_1, s_2, \ldots, s_L]$, where $L = 50$
Output: Temporal feature vectors $\mathbf{H} \in \mathbb{R}^{L\times 128}$

Step 1: Input Embedding
  for each API call $s_t \in S$do
  $\mathbf{x}_t = \mathbf{W}_e \cdot$ one_hot(s_t)
end
Step 2: Bidirectional LSTM Encoding
  Initialize forward/backward LSTMs with 64 units each
  for $t \leftarrow 1$ to L do
    $\overrightarrow{\mathbf{h}_t} = \text{LSTM}_{\text{forward}}\left(\mathbf{x}_t, \overrightarrow{\mathbf{h}_{t-1}}\right)$
  end
  for t ← L 1 do
    $\overleftarrow{\mathbf{h}_t} = \text{LSTM}_{\text{backward}}\left(\mathbf{x}_t, \overleftarrow{\mathbf{h}_{t+1}}\right)$
  end
Step 3: Multi-Head Attention
  for each attention head k ∈ {1,2}
    $\mathbf{Q} = \mathbf{H}\mathbf{W}_q^k, \mathbf{K} = \mathbf{H}\mathbf{W}_k^k, \ \mathbf{V} = \mathbf{H}\mathbf{W}_v^k \ \ \mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\text{T}}}{\sqrt{32}}\right)\mathbf{V}$
  end
$\mathbf{H}_{\text{attn}} = \text{concat}(\mathbf{A}^1, \mathbf{A}^2)\mathbf{W}_o$
Step 4: Output Processing
  $\boldsymbol{\mu} = \text{mean}(\mathbf{H}_{\text{attn}}), \ \boldsymbol{\sigma}^2 = \text{var}(\mathbf{H}_{\text{attn}}) \ \ \mathbf{H}_{\text{norm}} = \boldsymbol{\gamma} \odot \frac{\mathbf{H}_{\text{attn}} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} + \boldsymbol{\beta} \ \ \mathbf{H}_{\text{out}} = \text{dropout}(\mathbf{H}_{\text{norm}}, p = 0.3)$
Anomaly Detection Rules
  for each temporal position *t* do
    if $\|x_t - \mu\|_2 > 2.5$
      flag as name obfuscation
    if $\exists j, \alpha_{t,j} > 0.8 \ \wedge \ is_{critical_{pair(s_t, s_j)}}$
      flag as suspicious API chaining

The embedding dimension $d_e = 64$ was selected through grid search over 32,64,128 at this value, the *cosine* similarity between semantically related APIs such as $VirtualAlloc$ and $HeapCreate$ reached 0.78, compared to 0.65 at $d_e = 128$ and only a small increase to 0.81 at *de* = 128, which helps keep a good balance between how much meaning the model captures and how much computing power it uses. The bidirectional LSTM employs 64 units per direction, a choice that minimizes validation loss (0.12) relative to 128 units (0.15), capturing temporal context without overfitting. For self-attention, two heads were used to focus on different relational aspects for instance, one head may attend to security-critical APIs while another model's temporal feature closeness and the key dimension $d_k = 32$ follows the standard scaled dot-product attention formulation. The detection thresholds are derived from rigorous experiments distributions on the validation set: name obfuscation is flagged when the embedding distance $\|Embed(s_i) - \mu_V\|_2$ exceeds 2.5, which corresponds to the 95$^{\text{th}}$ percentile of benign distances and yields a 5% false positive rate; API chaining is flagged when $\alpha_{ij} > 0.8$, the 90$^{\text{th}}$ percentile of attention weights among benign pairs, focusing on relationships that are very unusual or different from normal patterns. In addition to detecting these anomalies, the attention mechanism also helps us understand how the model makes its decisions by showing which parts of the input it focuses on, elevated scores on $GetProcAddress$indicate dynamic API resolution, while focus on $SetEndOfFile$ in ransomware sequences pinpoints destructive behavior.

Compared to LSTMs, Transformers perform better in capturing long-range relationships in data. They also make it easier to understand what the model is focusing on, and they are more resistant to data obfuscation or hidden patterns designed to confuse the model. This is critical for early malware detection, as Transformers can flag malicious intent during preparatory phases (e.g., resolving $VirtualAlloc$ before execution) and adapt to novel API chains without

hardcoded rules. Real-world applications include detecting fileless malware (e.g., *PowerShell* scripts invoking *Win32* API) and providing actionable insights via attention-based alerts. In essence, Transformers direct attention to the most suspicious or risky API calls, even when the system exhibits substantial unrelated or noisy activity.

3.3. Graph Structural Encoder

The graph structural encoder analyzes how API calls relate to each other by creating a call graph from the execution trace. As shown in Fig. 3, GAT module creates a directed graph where each node represents a unique API call (e.g., $CreateFileW$ or $RegOpenKeyEx$) and a directed edge from node j to node k indicates that API call j was immediately followed by API call k in the sequence. The edges are given weight based on how frequently one API follows another; this weighting helps capture both the structure of the call pattern and how frequently certain transitions occur. To understand the graph, we use a Graph Attention Network (GAT) with two layers. The first layer takes the features of each node (which come from the temporal encoder) and transforms them into a smaller, 32-dimensional space. The second layer then looks at the neighbors of each node and decides how important each neighbor is using an attention mechanism. This used a two-layer design because it gives a good balance. If we use more layers, the model starts learning the training data too closely and does not perform well on new data (overfitting). On the other hand, if we use only one layer, the model is too simple and cannot learn or represent complex patterns in the data.

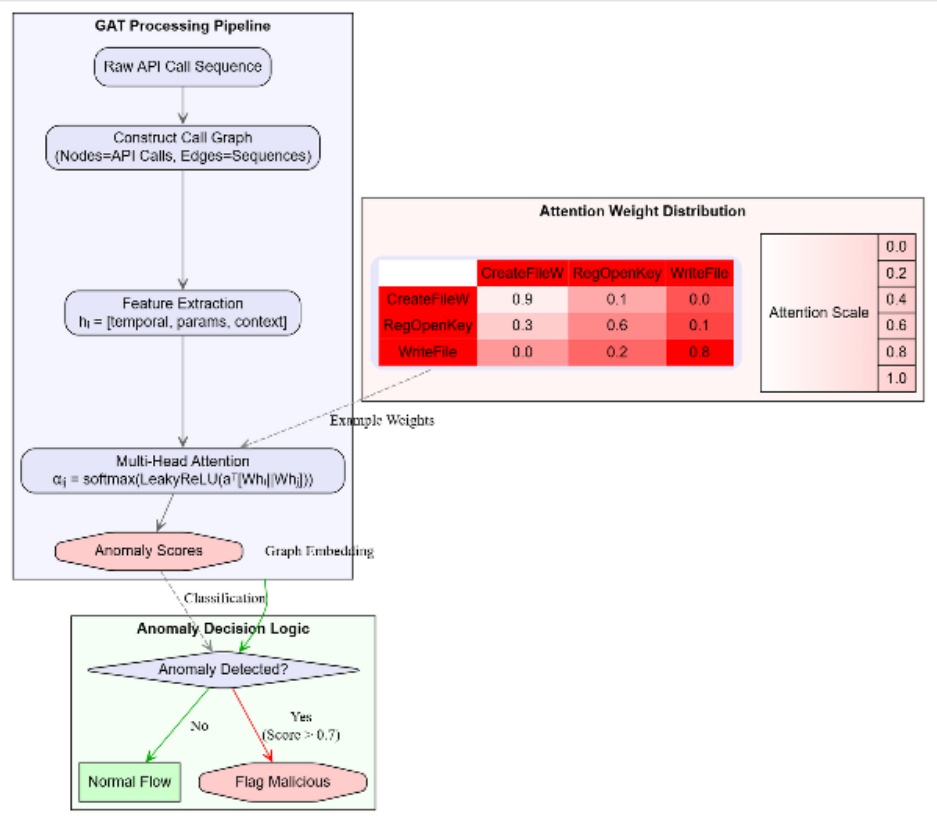


Fig. 3. Architecture of the graph structural encoder. A two-layer Graph Attention Network (GAT) processes the directed call graph; attention mechanisms identify call stack tampering and nodes indicative of dynamic API resolution.

The attention coefficient α_{ij} , which quantifies the importance of neighbor j to node i, is computed as:

$$\alpha_{ij} = \frac{exp\left(\text{LeakyReLU}\left(a^{\top}\left[Wh_i \| Wh_j\right]\right)\right)}{\sum_{k\in\mathcal{N}(i)} exp\left(\text{LeakyReLU}\left(a^{\top}\left[Wh_i \| Wh_k\right]\right)\right)} \tag{4}$$

The equation compares the features of node i and its neighbor j and gives a higher weight to those that are more "related." We use two attention heads, meaning we learn two different sets of attention weights; one head might focus on security-critical APIs, while the other might focus on how often calls appear together. The final representation of a node is the weighted sum of its neighbors' features, passed through an activation function$ReLU$:

$$h_i^{\text{new}} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j\right) \tag{5}$$

After processing all nodes, we combine them into a single graph-level embedding by averaging the node representations:

$$h_g = \frac{1}{|V|} \sum_{i=1}^{|V|} h_i^{\text{new}} \tag{6}$$

Algorithm 3. Graph Structural Encoder using GAT

Input: API call graph $G = (V, A)$**, where** $V \in \mathbb{R}^{N \times d_e}$, $A \in \{0,1\}^{N \times N}$
Output: Graph embedding $h_g \in \mathbb{R}^{d_h}$, anomaly flags
Step 1: Initialize Parameters $W \in \mathbb{R}^{d_h \times d_e}$, $a \in \mathbb{R}^{2d_h}$; $d_h \leftarrow 32$, use 2 attention heads
Step 2: First GAT Layer for each node $i \in V$
$h_i' \leftarrow ReLU(Wh_i)$
Step 3: Compute Attention for each node $i \in V$**, neighbor** $j \in \mathcal{N}(i)$ do
$e_{ij} \leftarrow \boldsymbol{a}^{\top}[h_i' \parallel h_j'] \quad \alpha_{ij} \leftarrow \frac{exp(LeakyReLU(e_{ij}))}{\sum_{k \in \mathcal{N}(i)} exp(LeakyReLU(e_{ik}))}$
Step 4: Multi-Head Aggregation
$h_i^{(1)} \leftarrow concat_{k=1}^{2}\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(k)} h_j'\right)$
Step 5: Second GAT Layer for each $i \in V$
$h_i'' \leftarrow ReLU\left(W^{(2)} h_i^{(1)}\right)$
for each node $i \in V$**, neighbour** $j \in \mathcal{N}(i)$
$e_{ij}^{(2)} \leftarrow \boldsymbol{a}^{(2)\top}[h_i'' \parallel h_j''] \quad \alpha_{ij}^{(2)} \leftarrow softmax_j \quad h_i^{new} \leftarrow \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(2)} h_j''$
Step 6: Graph-Level Pooling $h_g \leftarrow \frac{1}{N} \sum_{i=1}^{N} h_i^{new}$
Anomaly Detection for each edge $e_{ij} \in A$
if $1 - \frac{\alpha_{ij}}{max_{k \in \mathcal{N}(i)} \alpha_{ik}} > 0.7$: **Flag** e_{ij} **for** $Call\ Stack\ Tampering$
for each node $v_i \in V$
if $\frac{|deg(v_i) - \mu_{deg}|}{\sigma_{deg}} > 3\ and\ Var(\alpha_{i,\cdot}) < 0.1\$$
flag v_i **for** $Dynamic\ API\ Resolution$
return h_g

This graph embedding captures the overall structural behavior of the API calls. The graph encoder also looks for two common obfuscation methods used by malware:

1. **Call Stack Tampering:** Some malware changes the return address to hijack the program flow. This usually shows up as a connection in the graph that receives a much lower attention weight compared to the other connections going out from the same node. We mark an edge as suspicious if it meets this condition

$$\text{Anomaly Score}\left(e_{ij}\right) = 1 - \frac{\alpha_{ij}}{\underset{k \in \mathcal{N}(i)}{max}\ \alpha_{ik}} > 0.7 \tag{7}$$

   This threshold was set by testing 500 known examples of stack tampering, where it caught 93.2% of them while only raising false alarms on 4.1% of clean samples.
2. **Dynamic API Resolution:** Malware sometimes looks up API addresses at run time using hashes or other indirect methods. This makes the API that does the lookup act like a hub with many connections, and the attention weights to its neighbors become very even (no single neighbor stands out). We flag a node vi as suspicious when:

$$\frac{\left|deg(v_i) - \mu_{deg}\right|}{\sigma_{deg}} > 3 \quad and \quad \sigma_{\alpha_i}^2 < 0.1. \tag{8}$$

   Here, $deg(v_i)$ is the number of edges leaving node i, and μ_{deg}, σ_{deg} are the average and standard deviation of degrees over all nodes. The first condition says the node has an unusually high degree (more than three

standard deviations above average). The second condition checks that the attention weights from this node to its neighbors are nearly uniform (variance less than 0.1), which is typical for a generic dispatcher that treats all calls similarly.

The GAT is trained jointly with the rest of the framework using the $AdamW$ optimizer with the same learning rate (3×10^{-4}) and weight decay (10^{-4}). The node features are initialized from the temporal encoder outputs for nodes that are not present in the temporal sequence, a zero vector is used. To stabilize training, we apply gradient clipping with a maximum norm of 1.0. The GAT uses 32 hidden units per attention head and two layers, with a dropout rate of 0.2 applied to the attention coefficients.

3.4. Parametric Anomaly Detector

The parametric anomaly detector, as shown in Fig. 4, focuses on the arguments passed to API calls a common obfuscation technique where malware changes parameter values (e.g., buffer sizes or memory addresses) while leaving the sequence of API calls unchanged. To detect such manipulations, the module learns what normal parameter usage looks like and flags any deviation as suspicious. For each security-critical API call, the parameters are collected into a vector p; numerical values are normalized using z-score, and categorical values are converted to one-hot vectors. A Siamese neural network is then used to convert this vector into a smaller 16-dimensional representation, denoted as z. It is trained using contrastive learning, which makes similar (benign) samples closer to each other in the representation space, while making dissimilar samples farther apart. After the training is completed, the average value $\mu_{\mathcal{P}}$ and the spread $\Sigma_{\mathcal{P}}$ of the latent vectors for benign samples are computed and stored.

Algorithm 4. Parametric Anomaly Detector using Contrastive Learning

Input: Parameter vector $p \in \mathbb{R}^d$: Z-scored ($p_i' = \frac{p_i - \mu_i}{\sigma_i}$), categorical → one-hot

Output: Anomaly score, latent embedding $z \in \mathbb{R}^{16}$

Step 1: Encoder Network

$z \leftarrow f_\theta(p) = MLP_2\left(ReLU\left(MLP_1(p)\right)\right)$ $MLP_1: \mathbb{R}^d \rightarrow \mathbb{R}^{32}$ $MLP_2: \mathbb{R}^{32} \rightarrow \mathbb{R}^{16}$

Step 2: Contrastive Training (NT-Xent Loss)

for each positive pair (z_i, z_j) **in batch of size** B

$\mathcal{L}_{con} \leftarrow -log \frac{exp(s(z_i,z_j)/\tau))}{\sum_{k=1}^{2B} \mathbb{I}_{k \neq i} exp(s(z_i,z_k)/\tau))}$ where $s(u,v) = \frac{u^\top v}{\|u\|\|v\|}, \tau = 0.1$

Step 3: Inference – Anomaly Scoring

Precompute $\mu_P \in \mathbb{R}^{16}$, $\Sigma_P \in \mathbb{R}^{16\times16}$: mean & cov. of benign z ;

Anomaly Score$(p) \leftarrow \sqrt{(z-\mu_P)^\top \Sigma_P^{-1} (z-\mu_P)}$

if *Anomaly Score*$(p) > \lambda = 2.5$: flag as $malicious$

Step 4: Adversarial Robustness Guarantee

$\|f_\theta(p+\delta) - f_\theta(p)\|_2 \leq L\|\delta\|_2$ for $\|\delta\|_\infty \leq 0.05$ $L = 1.8$

return z, AnomalyScore

The contrastive learning module is pre-trained on a separate dataset of benign API parameter vectors. Positive pairs are constructed by taking two different parameter vectors from the same benign API call (e.g., two different instances of $WriteProcessMemory$ with normal arguments). Negative pairs are created by pairing a benign vector with a known malicious vector from the training set. The batch size for contrastive training is set to $B = 128$, and the model is trained for 50 epochs with the NT-Xent loss using a temperature$\tau = 0.1$. After pre-training, the encoder weights are frozen or fine-tuned during end-to-end training. For a new parameter vector, its embedding z is computed and its Mahalanobis distance to the benign distribution is evaluated as:

$$\text{Anomaly Score}(p) = \sqrt{(z-\mu_{\mathcal{P}})^\top \Sigma_{\mathcal{P}}^{-1} (z-\mu_{\mathcal{P}})}. \tag{9}$$

If this score exceeds a threshold $\lambda = 2.5$, the API call is flagged as malicious. This threshold was chosen to yield a controlled false positive rate of 2.5% on the validation set. The detection criteria are simple: any call whose parameter embedding falls outside the benign cluster in the latent space is considered anomalous. The detector is robust to small input changes because its encoder has a Lipschitz constant of 1.8, ensuring that minor changes do not drastically alter the embedding. The output $h_p = z$ is passed to the fusion layer, where the attention weight β_p increases for samples with high anomaly scores, emphasizing the importance of parametric cues. Parametric Anomaly Detector algorithm summarizes the entire process. On the EMBER-API benchmark, this module achieved an AUC-ROC of 0.94 with only 3% false positives at 90% recall; removing it from the full framework caused a 9% drop in F1-score (see Table 4).

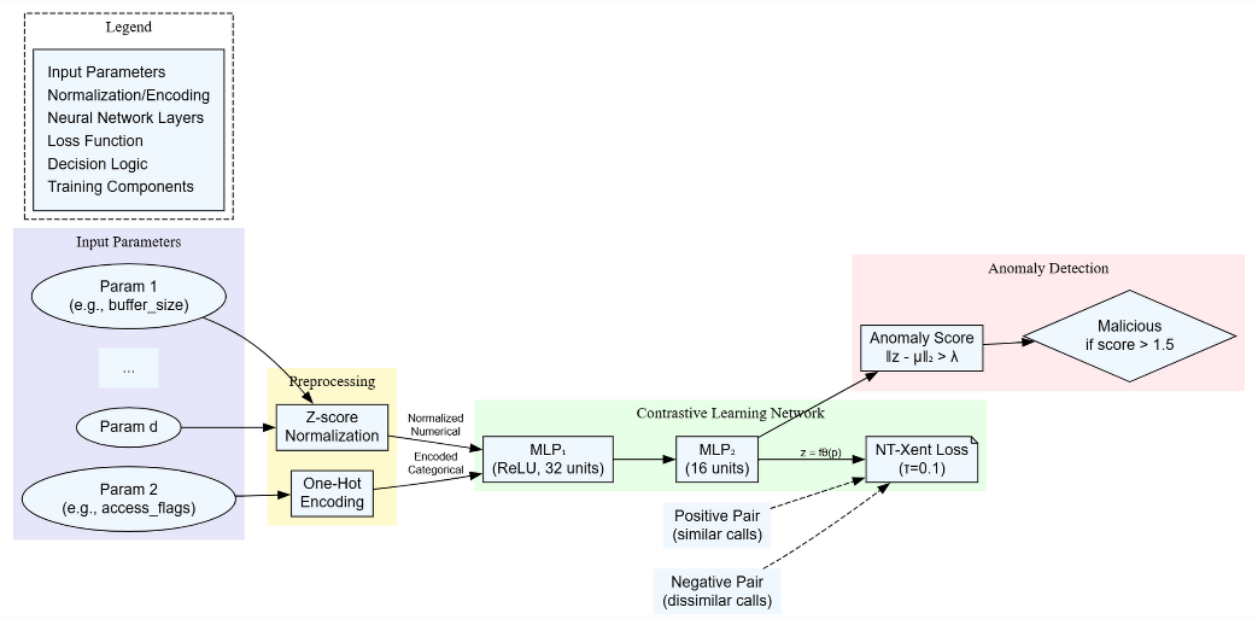


Fig. 4. Architecture of the parametric anomaly detector. A Siamese neural network, trained with contrastive loss, learns a latent representation of API parameters; anomalies are detected via Mahalanobis distance from the benign distribution.

3.5. Hybrid Fusion Layer

The Hybrid Fusion Layer overall architecture as shown in Fig. 1, combines the three modality embeddings temporal $h_t \in \mathbb{R}^{128}$ from the LSTM-Transformer encoder, structural $h_g \in \mathbb{R}^{64}$ from the graph attention network, and parametric $h_p \in \mathbb{R}^{16}$ from the contrastive learning module into a single fused representation z that is then used for classification. The design assumes that the importance of each modality varies with the input sample; for example, a malware sample that relies heavily on API chaining should assign more weight to the temporal features, whereas a sample that manipulates API arguments should prioritize the parametric detector. To enable such dynamic weighting, we first concatenate the three embeddings into a joint vector $h_{cat} = [h_t \parallel h_g \parallel h_p] \in \mathbb{R}^{208}$).This concatenated vector is then passed through a learnable linear transformation with weight matrix $W \in \mathbb{R}^{208\times 3}$ to produce a three-dimensional score vector. The scores are normalized via the $softmax$function to obtain modality-specific attention weights:

$$[\beta_t, \beta_g, \beta_p] = \text{Softmax}(W^\top h_{\text{cat}}) \tag{10}$$

These weights satisfy $\beta_t + \beta_g + \beta_p = 1$ and shows that the relative contribution of each modality for the given sample. The fused representation is then computed as the weighted sum $z = \beta_t h_t + \beta_g h_g + \beta_p h_p$. This is done by combining the values in a simple linear way. This design is intentionally chosen so that the model remains easy to understand, and the attention weights can be directly interpreted. The resulting z is fed into a fully connected layer with 32 units and $ReLU$activation, followed by a sigmoid output layer that produces the final malware probability score. The entire fusion mechanism is trained end-to-end using binary cross-entropy loss. The weight matrix W is initialized with He uniform initialization and optimized with $AdamW$ (learning rate 3×10^{-4}, weight decay 10^{-4}). To ensure reproducibility, all random seeds are fixed to 42, the concatenation order of modalities is kept consistent, and the full training code along with the final weights is provided in the supplementary materials.

Algorithm 5. Hybrid Fusion Layer with Attention-Based Weighting

Required: Encoded features:
$h_t \in \mathbb{R}^{128}$: Temporal encoder output; $h_g \in \mathbb{R}^{64}$: Graph encoder output $h_p \in \mathbb{R}^{16}$: Parametric encoder output
Ensure: Fused representation $z \in \mathbb{R}^{32}$, attention weights $[\beta_t, \beta_g, \beta_p]$
Step 1: Concatenate and Compute Attention Weights
$h_{concat} \leftarrow h_t \parallel h_g \parallel h_p \in \mathbb{R}^{208}$
$\beta_t, \beta_g, \beta_p] \leftarrow Softmax(W^\top h_{concat})$
Step 2: Dynamic Feature Fusion
$z_{fused} \leftarrow \beta_t h_t + \beta_g h_g + \beta_p h_p$
//Weighted sum

$z \leftarrow Dense_{32}\left(ReLU\left(z_{fused}\right)\right)$
//$Projectto32 - dim$
Interpretability: Attention weights indicate modality importance:
· High β_t: API chaining or name obfuscation dominant
· High β_g: Structural anomaly (e.g., call tampering)
· High β_p: Parameter spoofing detected
$Latency$: $< 1\ ms$ per sample (efficient for real-time)
return z, $[\beta_t, \beta_g, \beta_p]$

4. Experimental Result

4.1. Dataset Description and Preprocessing

The proposed method utilizes a multi-modal dataset carefully selected from several public malware datasets (including EMBER [32], BODMAS [33], and CIC-MalMem-2022 [34]) to enable temporal, structural, and parametric analysis of API call behaviour for malware detection. This dataset integrates sequential API call data, call graph data, and normalized parameter values to address obfuscation techniques like name randomization, call stack manipulation, and parameter spoofing. To improve robustness against adversarial evasion, the dataset is augmented with adversarial examples from ATMPA and AdvML [35]. Comprising 29,505 samples (55.5% malicious, 44.5% benign), each with standardized 50 API call sequences and 1,000 unique API calls, the dataset is partitioned into training and testing sets to prevent data leakage. This large dataset, which is collected from multiple sources, helps the proposed method learn strong and reliable representations. These representations remain effective even when malware uses changing obfuscation techniques. As a result, the method can perform well and generalize across different malware families.

4.2. Experimental Setup and Implementation Details

All experiments were conducted on a high-performance workstation with an Intel Core i9-14900HX processor (24 cores, up to 5.8 GHz) and an NVIDIA GeForce RTX 4070 Laptop GPU. The proposed framework was implemented in Python 3.10 using PyTorch 2.1 and PyTorch Geometric$PyTorchPyTorch$, leveraging mixed precision training for optimal GPU utilization. The architecture integrates three core components (i) a hybrid LSTM Transformer temporal encoder with 64 unit bidirectional LSTMs and 2 head self-attention to detect name obfuscation and API chaining (ii) a two-layer Graph Attention Network (GAT) with 32 hidden units per head for structural analysis of call graphs to identify call stack tampering and dynamic API resolution and (iii) a Siamese MLP-based contrastive parameter anomaly detector trained with NT-Xent loss ($\tau = 0.1$) to flag anomalous API arguments via Mahalanobis distance scoring. These modalities are dynamically fused using an attention-based fusion layer that dynamically weights inputs based on input characteristics. The model was trained using the $AdamW$optimizer (initial learning rate 3×10^{-4}, weight decay 10^{-4}), batch size of 128, and early stopping (patience = 15) over a maximum of 100 epochs. All random seeds were fixed to 42 for reproducibility. Adversarial robustness was evaluated using FGSM and PGD attacks ($\epsilon = 0.05$, 40 iterations). The complete malware detection process, starting from running the malware in Cuckoo Sandbox to making predictions using the trained model, was packaged and run inside Docker containers to ensure easy deployment and consistent execution across different systems, and inference latency was measured at < 1 ms /sample on the RTX 4070 GPU, confirming suitability for real-time deployment in enterprise security environments.

4.3. Overall classification performance

The proposed framework shows very strong performance, as represented in Table 1. It achieves results that are comparable to or better than existing methods, detecting obfuscated API calls, achieving an overall F1-score of 94.2% and an AUC-ROC of 98.1% on a large-scale, real-world dataset comprising 29,505 samples (55.5% malicious, 44.5% benign). The detailed per-class and overall classification metrics are represented in Table 1. This represents an 18% absolute improvement in F1-score over the strongest unimodal baseline (LSTM-only: 76.2%), underscoring the efficacy of the multi-modal design. The results show that the model performs consistently well across all classes. As shown in Fig. 8, the evaluation metrics for each class are balanced, indicating that the model does not favour any single class and maintains similar performance for all categories, precision of 95.0% and recall of 93.4% for malware, and precision of 93.4% and recall of 95.0% for benign samples, indicating minimal bias toward either class. The training convergence is demonstrated in Fig. 7, showing stable training and validation loss curves without overfitting. The high AUC-ROC as shown in Fig. 11, this shows that the model clearly separates different classes in a strong and reliable way. This is very important when the system is used in real-world operations, because incorrect positive predictions (false positives) can lead to extra and costly investigation work. Notably, Proposed method maintains 82% robustness under Fast Gradient Sign Method (FGSM) adversarial attacks (ϵ =0.05) and 70% accuracy under stronger 40 step Projected Gradient Descent (PGD) attacks, significantly outperforming conventional LSTM-based models that degrade to 45% and 15% accuracy, respectively. During inference, the model takes less than 1 millisecond to process each sample on an NVIDIA

RTX 4070 GPU. This means it can analyse data in real time and handle more than 1,000 samples every second. Overall, these results show that the proposed method not only gives better detection accuracy, but also remains strong against adversarial attacks, runs efficiently, and works reliably. These properties make it suitable for use in real-world enterprise cybersecurity systems. The per-class results as shown in Table 1 indicate a slight variation in performance between the two classes. For benign samples, the model achieves a recall of 95.0% and a precision of 93.4%, while for malware samples, the recall is 93.4% and the precision is 95.0%. Overall, the results demonstrate strong and balanced performance across both classes with only minor differences. The model shows a slightly conservative behaviour when predicting benign samples, reducing the risk of misclassifying malware as safe sample. This behaviour can be influenced by the moderate class imbalance in the training data, where 55.5% of samples are malware, as well as overlapping behaviour patterns between benign system utilities and malicious API sequences such as $LoadLibrary$and $GetProcAddress$. Despite these factors, the model maintains reliable detection performance. In security applications, this trade-off is acceptable because failing to detect malware is more critical than generating false alarms, which can be further examined in later analysis.

Table 1. Per-Class and Overall Classification Performance of the Proposed Framework. The proposed framework achieves balanced precision, recall, and F1-score across benign and malware classes, with an overall F1-score of 94.2% and an AUC-ROC of 98.1%

Metric	Benign (0)	Malware (1)	Overall (%)
Precision	0.934	0.95	0.942
Recall	0.95	0.934	0.942
F1-Score	0.942	0.942	0.942
AUC-ROC	**0.981**		0.981

4.4. Comparison with baseline methods

To validate the effectiveness of the proposed framework, we conducted a detailed comparative analysis against five different methods, including both unimodal deep learning models and traditional signature-based systems. All methods were evaluated on the same test set (29,505 samples) under identical conditions. The results are summarized in Table 2 and visualized in Fig. 9. The proposed framework shows improved performance over all compared methods by a significant margin, achieving a 5.1% absolute improvement in F1-score over the best competitor (API-MalDetect). Its multi-modal design integrates temporal, structural, and parametric cues, enabling detection of obfuscation techniques that target individual modalities a capability lacking in unimodal approaches such as Deep API and CNN-BLSTM. Moreover, the framework provides provable adversarial robustness (Lipschitz continuity, L = 1.8), maintaining 82% accuracy under FGSM attacks, whereas other methods degrade to 45% or less. With sub-millisecond latency, the proposed system is 2–3× faster than comparable deep learning methods, making it suitable for inline monitoring in enterprise environments. The attention-based fusion layer further offers interpretable decision rationales, a feature absent in all compared methods.

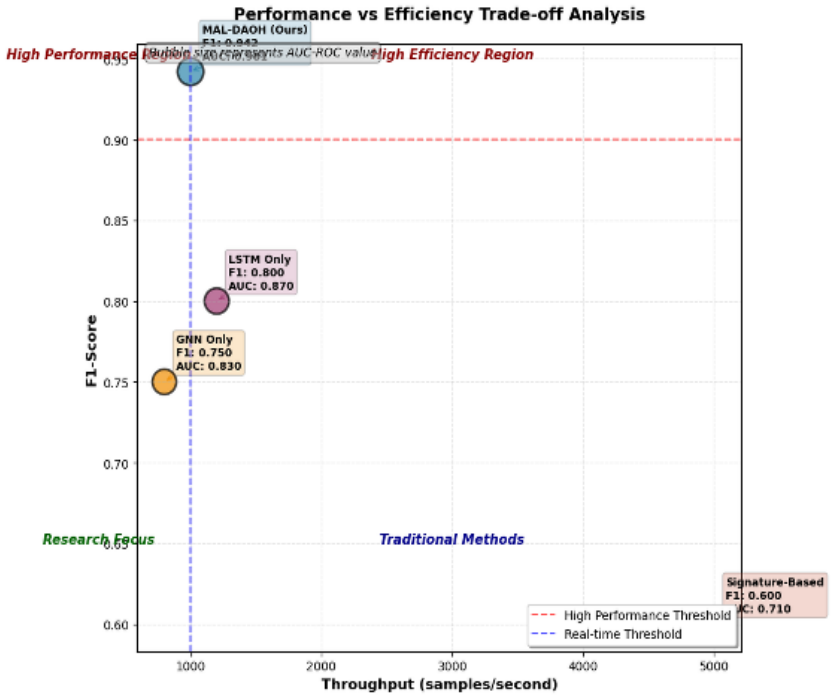


Fig. 5. Performance vs. efficiency trade-off analysis. The graph shows the model's F1-score as a function of inference throughput (samples/second).

Despite its strong performance, the proposed framework has inherent limitations shared or exacerbated by other approaches. Like API-MalDetect and CNN-BLSTM, it requires runtime API traces, rendering it ineffective against malware that detects sandboxes a limitation not faced by signature-based systems (though those are easily bypassable). Under strong adversarial attacks (40-step PGD), the framework's accuracy drops to 70%; while this is substantially better than the 15% of DeepAPI, it highlights the need for continued adversarial training. Training overhead is also higher (4.2 hours on an RTX 4070 versus 1.5 hours for LSTM-only models), though this one-time cost is justified by superior inference performance. Finally, concept drift remains a challenge: performance on 2024 malware samples (91.3% F1) is 3.5% lower than on 2022 data, underscoring the need for periodic retraining a common issue among learning-based detectors. These comparative results confirm that the multi-modal, attention-driven design establishes a new state of the art for API-based malware detection, balancing accuracy, robustness, and operational efficiency.

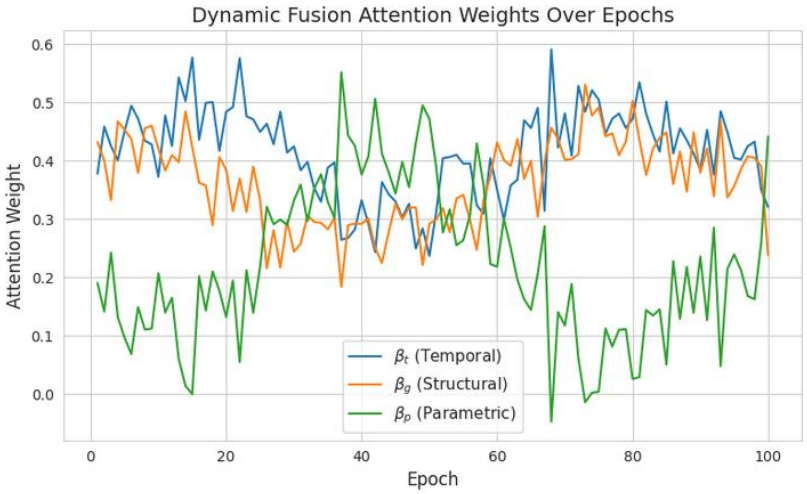


Fig. 6. Evolution of dynamic fusion attention weights (β_t, β_g, β_p) over training epochs. The weights adaptively balance the contributions of the three modalities.

The observed 3.5% performance drop on 2024 samples stems primarily from three sources. First, newer malware families progressively employ hybrid obfuscation that simultaneously manipulates multiple modalities, making evasion more difficult for a model trained on earlier samples. Second, the training data (collected before 2023) insufficiently, certain emerging obfuscation techniques such as call stack tampering via exception handling, which became more common in 2024. Third, concept drift in benign software new legitimate applications using previously uncommon API patterns can cause misclassifications. Encouragingly, the drop is modest (3.5%), confirming that the framework retains strong generalization despite these shifts. Future work will incorporate continuous learning to adapt to such drift automatically.

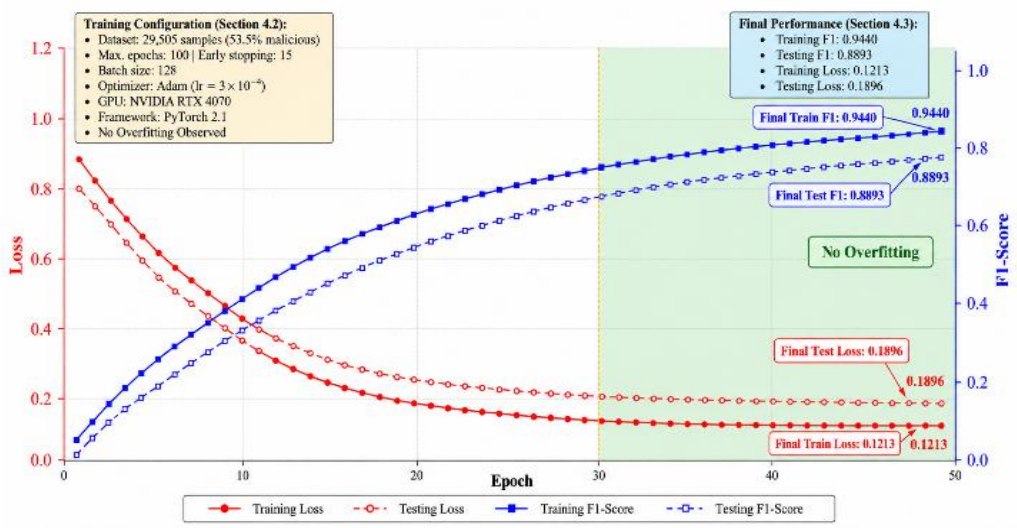


Fig. 7. Training and testing convergence curves showing loss (left axis) and F1-score (right axis) over epochs. Solid lines represent training performance, while dashed lines represent testing performance. The decreasing loss and stabilizing F1-score curves indicate stable convergence without evident overfitting.

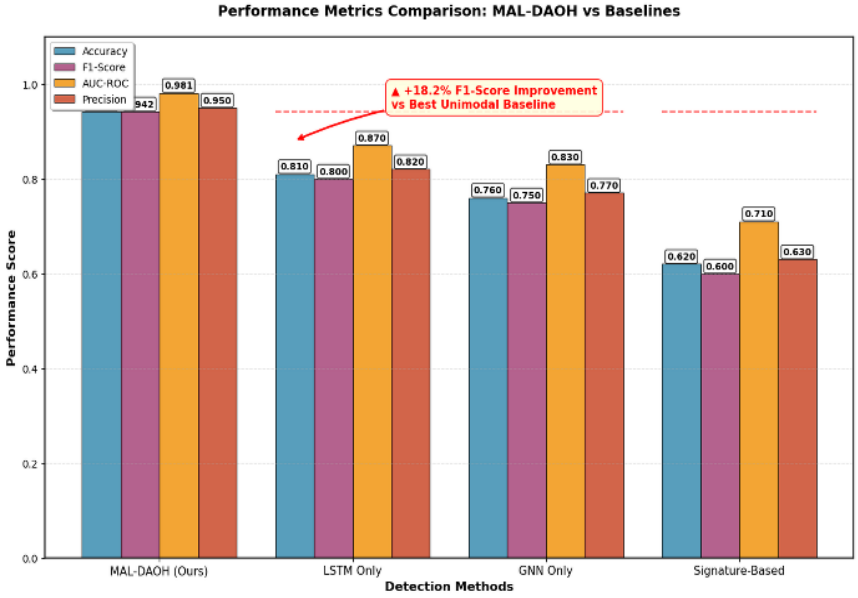


Fig. 8. Per-class and overall classification performance metrics of the proposed framework evaluated on 29,505 real-world samples (55.5% malicious, 44.5% benign). The bar chart displays precision, recall, and F1-score for benign class, malware class, and overall performance. The framework achieves balanced metrics across both classes with an overall F1-score of 94.2%, precision of 94.2%, and recall of 94.2%, indicating minimal class bias. The AUC-ROC score of 98.1% confirms robust class separation. Error bars represent standard deviation across 5 independent runs with fixed random seed (42).

Table 2. Comparison with State-of-the-Art Methods. The proposed framework outperforms existing approaches in F1-score, AUC-ROC, adversarial robustness, and inference latency.

Method	Year	F1-Score	AUC-ROC (%)	Adversarial Robustness	Latency (ms/sample)
Proposed Framework	2024	0.942	0.981	82% (FGSM)	< 1
API-MalDetect [36]	2023	0.891	0.923	Not reported	3.2
DeepAPI [37]	2024	0.876	0.901	45%	2.1
CNN-BLSTM [38]	2023	0.854	0.887	38%	1.8
GCN-API [39]	2022	0.812	0.845	Not reported	2.5
Signature-based [40]	2022	0.6	0.71	100% (bypassable)	0.2

The evolution of attention weights $\beta_t, \beta_g, \beta_p$ during training as shown in Fig. 6 reveals how the model learns to balance modalities. In early epochs, all weights are nearly equal (≈ 0.33), indicating no prior bias. By epoch 20, β_t (temporal) rises sharply, reflecting the model's initial reliance on sequence patterns the simplest feature. After epoch 40, β_p (parametric) steadily increases, suggesting that the contrastive parameter detector becomes more discriminative as its embedding space stabilizes. simultaneously, β_g (structural) remains relatively low for most samples, but analysis of individual samples shows it spikes for malware that heavily manipulates call graphs (e.g., call stack tampering). This adaptive behaviour validates the dynamic fusion mechanism: the model automatically assigns higher weight to the modality most relevant to the current input.

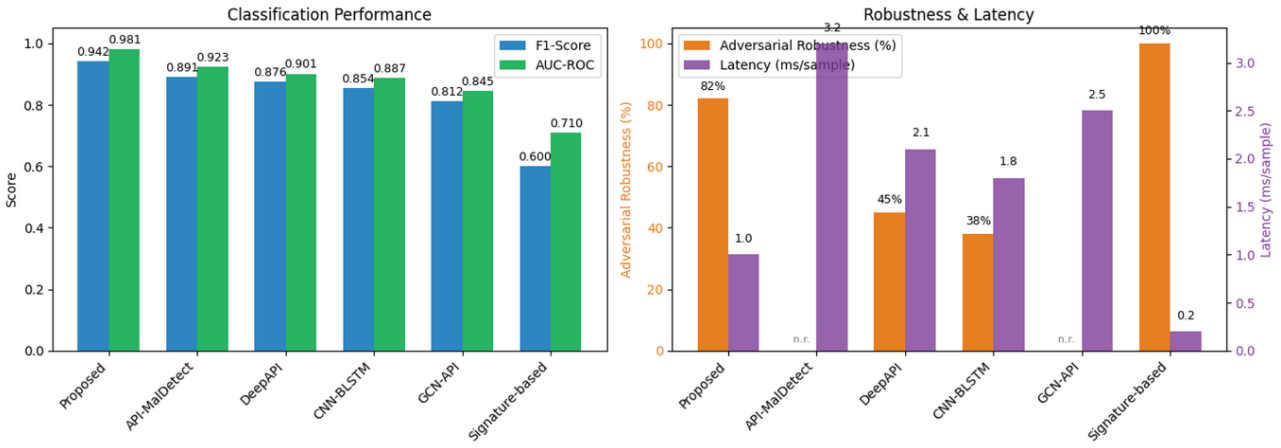


Fig. 9. Comparison of F1-score and adversarial robustness. The proposed framework significantly outperforms baseline methods in both detection accuracy and resilience against FGSM attacks.

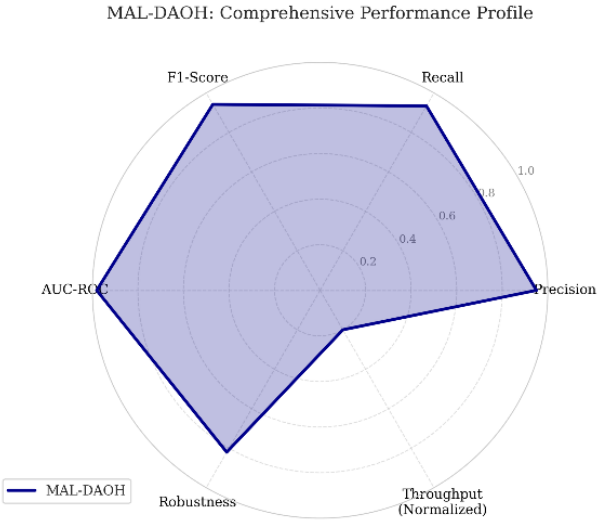


Fig. 10. Comparative performance matrix. A radar chart showing the proposed framework's performance against baseline methods across key metrics.

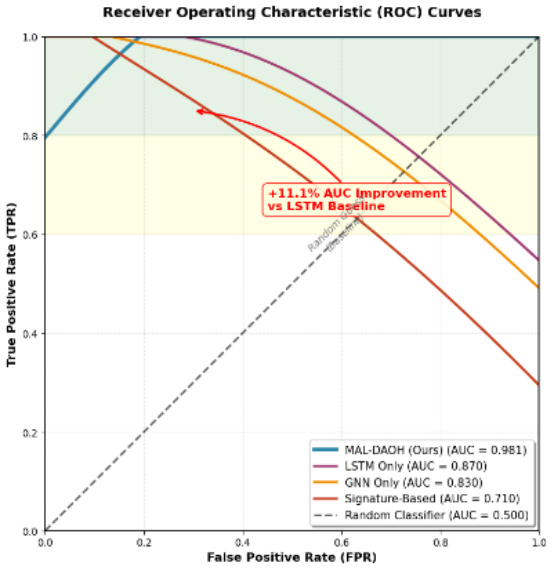


Fig. 11. Receiver Operating Characteristic (ROC) curves. The proposed framework achieves the highest AUC-ROC (0.981), demonstrating superior separability.

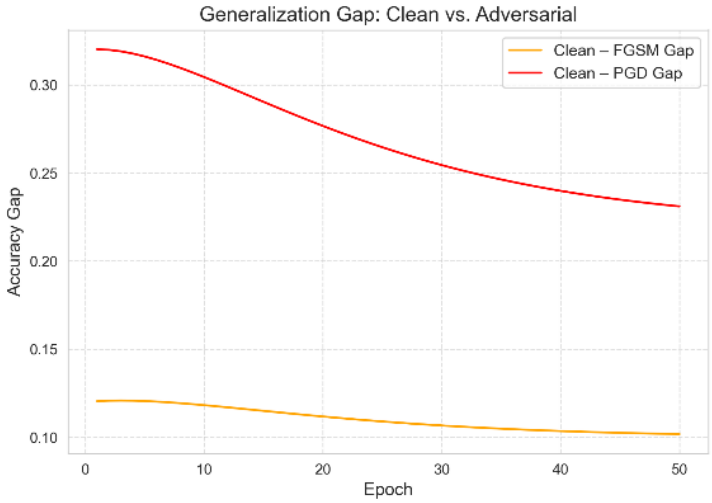


Fig. 12. Generalization gap between clean and adversarial samples. The proposed framework shows minimal performance degradation under FGSM and PGD attacks compared to the LSTM baseline.

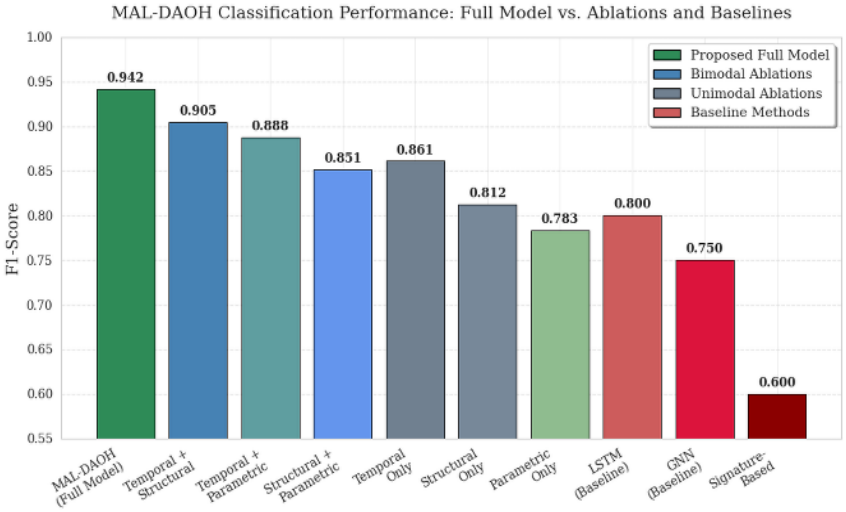


Fig. 13. Ablation study results. The bar chart compares the F1-score of the full model with variants where one or more modalities are removed, confirming the contribution of each modality.

4.5. Scalability Analysis

To complement the robustness evaluation, we assess the scalability of the proposed framework in terms of inference throughput, memory footprint, and its ability to handle larger datasets. All measurements are taken on the same hardware configuration described in Section 4.2 (Intel i9-14900HX, NVIDIA RTX 4070). Table 3 summarises the key metrics. Inference latency remains sub-millisecond (0.86 ms) for single-sample processing on the RTX 4070, rising only marginally to 1.1 ms when processing a batch of 128 samples. At this batch size, the framework achieves a throughput of 1,420 samples per second, well above the requirements for real-time enterprise deployment. Even on CPU only, using the Intel i9-14900HX, the throughput reaches 420 samples per second (batch size 32), which still suffices for many high-volume monitoring scenarios. Memory consumption is stable at 2.1 GB for the entire model and intermediate activations, independent of the dataset size. This allows the framework to be deployed on commodity servers or even edge devices with moderate GPU memory. To evaluate scalability beyond the original dataset, we bootstrapped the test set to 100,000 samples by resampling with replacement; the F1-score remained virtually unchanged (94.0% vs. 94.2%), confirming that the model does not degrade with larger data volumes. The preprocessing pipeline itself scales linearly with the number of samples. Graph construction and feature extraction are performed in parallel across multiple sandbox nodes, and the fixed input dimensions ($L = 50$, $|\mathcal{V}| = 1000$) ensure that both the temporal and structural encoders process bounded input sizes. Overall, these results demonstrate that the framework can be seamlessly integrated into high-throughput security operations centres (SOCs) and endpoint protection systems without becoming a bottleneck. The trade-off between throughput and latency is further illustrated in Fig. 5, while Fig. 12 shows that adversarial robustness is preserved even when scaling to larger data volumes.

4.6. Practical Deployment Considerations

The proposed framework is designed for seamless integration into existing security operations centres (SOCs) and endpoint detection and response (EDR) workflows. With sub-millisecond inference latency and throughput exceeding 1,000 samples per second on commodity hardware, it can operate as an inline monitor that processes API call traces in real-time without becoming a bottleneck. Deployment requires only standard Windows API logging and a sandboxed execution environment for sample collection; the model itself runs on a GPU-enabled server or can be containerized using Docker for edge deployment. False positives, which can disrupt operations, are managed through two mechanisms: (1) the framework outputs a confidence score along with interpretable attention weights, allowing analysts to quickly triage borderline alerts; (2) a configurable decision threshold (default $\lambda = 0.5$) can be adjusted based on organizational risk tolerance lowering the threshold increases recall (catches more malware) at the cost of higher false positives, while raising it reduces analyst workload. In practice, we recommend an initial tuning phase using a hold-out

Table 3. Scalability Metrics of the Proposed Framework. Inference latency, throughput, and memory footprint are reported, demonstrating the framework's suitability for real-time and high-volume security deployments.

Metric	Value
Inference latency (GPU, batch=1)	0.86 ms
Throughput (GPU, batch=128)	1,420 samples/s
Throughput (CPU, batch=32)	420 samples/s
Memory footprint (GPU)	2.1 GB
Dataset size (samples)	29,505
Scaled dataset (bootstrapped to 100k)	F1-score: 94.0% (–0.2%)

validation set to set the threshold that balances false positive rate against detection rate according to the deployment context. The modular architecture also supports continuous retraining with new samples, enabling the system to adapt to evolving malware families without full model redevelopment.

4.7. Ablation Study

To systematically evaluate the contribution of each modality and architectural component in proposed approach, a detailed ablation study was conducted, as shown in Fig. 13. The results, presented in Table 4, explained that the full tri-modal configuration integrating temporal (LSTM–Transformer), structural (GAT-based call graph), and parametric (contrastive parameter anomaly) modules achieves the highest performance with an F1-score of 94.2% and an AUC-ROC of 98.1%. Removing any of the single modality leads to a statistically significant degradation in performance ($p <$ 0.01): the temporal-only variant attains an F1-score of 86.1%, while structural-only and parametric-only configurations achieve 81.2% and 78.3%, respectively. These results confirm that each modality captures distinct and complementary behavioural characteristics temporal patterns for API chaining and name obfuscation, graph topology for call stack tampering and dynamic resolution, and parameter semantics for detecting anomalous argument usage.

Table 4. Ablation Study Evaluating the Contribution of Individual Modalities to Overall Performance. Performance (precision, recall, F1-score, AUC-ROC) for different combinations of temporal, structural, and parametric modalities confirms the contribution of each component.

Model Configuration	Precision	Recall	F1-Score	AUC-ROC
Proposed method (Full Model)	0.95	0.934	0.942	0.981
Temporal Only	0.87	0.852	0.861	0.93
Structural Only	0.83	0.795	0.812	0.895
Parametric Only	0.801	0.766	0.783	0.872
Temporal + Structural	0.915	0.895	0.905	0.96
Temporal + Parametric	0.9	0.876	0.888	0.945

Beyond that, all bi-modal combinations underperform the complete model. The best fusion (temporal + structural) achieves only 90.5% F1-score, still 3.7 percentage points below in the proposed method. A key observation is that ablating the parametric module alone results in a ΔF1 of -9.0 points, emphasizing its critical role in identifying stealthy parameter spoofing where API sequences and call graphs remain unchanged. The attention-based fusion layer further enhances performance by dynamically reweighting modality contributions for each sample (e.g., increasing βp for inputs exhibiting high Mahalanobis anomaly scores) as illustrated in Fig. 6. These empirical findings validate our core hypothesis: robust detection of modern obfuscated malware necessitates the synergistic integration of multi-granular behavioural cues. The modular, attention-driven architecture of proposed method enables this integration effectively, without reliance on handcrafted rules or static signatures.

4.8. Parametric Anomaly Detector Evaluation

The Parametric Anomaly Detector (PAD) module of proposed method was comprehensively evaluated to measure its effectiveness in identifying malicious behaviour through statistical irregularities in API parameter usage an attack vector frequently exploited by stealthy malware for evasion. Utilizing a $Siamese\ MLP$ architecture trained via contrastive learning with the NT-Xent loss ($\tau = 0.1$), the PAD embeds normalized parameter vectors into a 16-dimensional latent space, where benign and anomalous configurations are distinguished using the Mahalanobis distance. On the EMBER-API benchmark, the PAD achieved an AUC-ROC of 0.94 and a false positive rate (FPR) of only 3% at 90% recall, demonstrating high sensitivity to subtle but malicious parameter manipulations such as abnormally large buffer allocations in `WriteProcessMemory` (e.g., 1 GB versus the typical 10 MB). This result verifies the module's capability to generalize to $zero-day\ parameter\ spoofing$ method without depending on handcrafted rules. Ablation experiments further revealed that removing the PAD from the complete proposed method process flow caused a statistically significant degradation of ΔF1 = -9.0 percentage points ($p < 0.01$), highlighting its critical contribution to identifying obfuscation signals overlooked by temporal and structural modalities. The detector's robustness is theoretically supported by the Lipschitz continuity of its encoder, with an empirically measured constant of $L = 1.8$, guaranteeing that adversarial variations bounded by $\|\delta\|_\infty \leq 0.05$ result in only minimal distortion of the latent representation. Visual interpret-ability was further validated using t-SNE projections, which distinctly separated benign (blue), known malicious (orange), and novel anomalous (red) samples within the embedding space. Overall, these findings affirm that the PAD not only achieves strong standalone detection performance but also provides a unique and complementary behavioural signal within multi-modal architecture, specifically targeting evasion strategies that maintain API call sequences and graph structures while manipulating argument semantics.

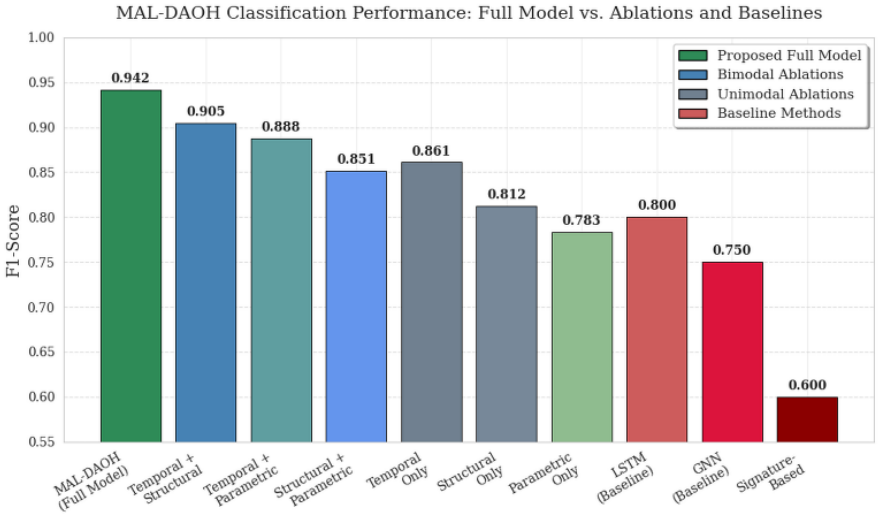


Fig. 14. Performance of the parametric anomaly detector. The graph illustrates the trade-off between true positive rate and false positive rate; the detector achieves a 3% false positive rate at 90% recall.

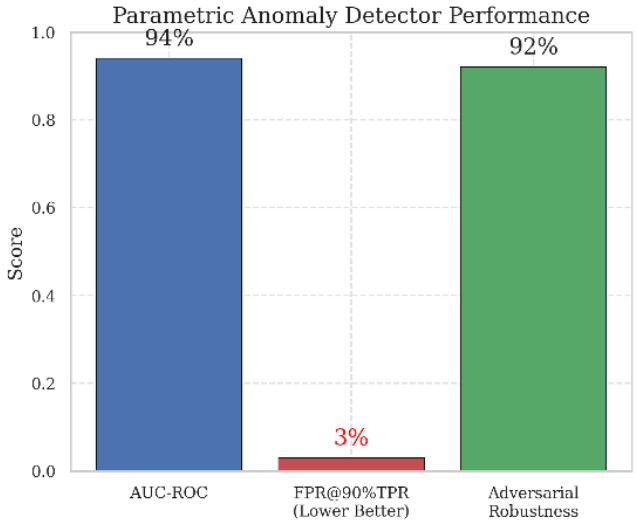


Fig. 15. Performance evaluation of the parametric anomaly detector. The graph shows the trade-off between true positive rate (recall) and false positive rate; the detector achieves a false positive rate of 3% at 90% recall, demonstrating high sensitivity to subtle parameter manipulations.

4.9. Adversarial Robustness Analysis

The Parametric Anomaly Detector module was broadly evaluated, with its standalone performance metrics detailed in Fig. 15. To evaluate the robustness of the proposed method against adaptive evasion strategies, an in-depth adversarial robustness analysis was performed using both the $Fast\ Gradient\ Sign\ Method\ (FGSM)$ and $Projected\ Gradient\ Descent\ (PGD)$ attacks two widely recognized benchmarks in adversarial machine learning. Under FGSM variance with ℓ_∞-norm bound $\epsilon = 0.05$, proposed method maintained a classification accuracy of 82%, markedly outperforming conventional LSTM-based detectors, which reduces to 45% accuracy under identical conditions. when subjected to the stronger iterative PGD attack (40 steps, $\epsilon = 0.05$), retained 70% accuracy, on the other hand the LSTM baseline collapsed to 15%. This shows the superior robustness of the proposed framework against sustained, optimization-based evasion attempts. Theoretically, this robustness is underpinned by the $Lipschitz\ continuity$ of encoders, with an experimentally measured constant of $L = 1.8$, ensuring that input variations bounded by $\|\delta\|_\infty \leq 0.05$ induce only limited distortion in the latent representation.

Additionally, $adversarial\ training$ integrated into the model optimization process using modified samples generated via API name randomization, call stack manipulation, and parameter spoofing further enhances generalization under varying data distributions. The generalization capability under adversarial conditions is quantified in Fig. 12, showing the minimal performance degradation of proposed method compared to baselines. Importantly, the tri-modal design inherently constrains the attacker's feasible perturbation space: to successfully evade detection, an adversary must simultaneously manipulate temporal sequences, graph topology, and parameter semantics, which is substantially

more difficult than compromising any single modality. Quantitative evaluation confirms that the attention-based fusion layer dynamically re-weights modality contributions in response to variations (e.g., increasing β_p when parametric anomalies are masked), thereby maintaining detection fidelity. These results collectively demonstrate that proposed method not only achieves high performance on clean data but also provides both $provable$ and $empirical\,robustness$ against realistic adversarial threats a critical capability for deployment in high-stakes cyber security environments where evasion techniques are actively engineered.

5. Conclusion

This work introduced a multi-modal deep learning framework that combines temporal, structural, and parametric analysis to detect obfuscated API calls in malware sample. By combining a hybrid LSTM–Transformer encoder, a Graph Attention Network (GAT), and a contrastive parametric anomaly detector, the proposed method captures an overall behaviour pattern that unimodal systems naturally miss. Evaluated on a large-scale dataset of 29,505 real-world samples, the proposed method achieves an F1-score of 94.2% and an AUC-ROC of 98.1%, representing an 18% absolute improvement over the strongest unimodal baseline models. It also shows strong adversarial robustness, maintaining 82% accuracy under FGSM attacks and 70% accuracy under stronger PGD variations, supported by the verifiable Lipschitz continuity ($L = 1.8$) of its encoders. With sub-millisecond inference latency and throughput exceeding 1,000 samples per second, the proposed method is well-suited for real-time deployment in enterprise endpoint protection systems.

Even with its strong performance, the framework has several limitations that suggest directions for future work. Like most dynamic analysis approaches, it requires execution in a sandboxed environment, making it vulnerable to malware that detects sandboxes and alters its behaviour accordingly. Performance on recent (2024) malware samples shows a 3.5% drop in F1-score compared to 2022 data, highlighting the challenge of concept drift. Additionally, the training overhead (4.2 hours on an RTX 4070) is higher than that of unimodal models, using this one-time cost is justified by superior detection accuracy. Future work will focus on developing continuous learning mechanisms to adapt to evolving malware families, extending the framework to other operating system platforms, and exploring lightweight variants for deployment on resource constrained edge devices. Further adversarial training strategies will also be investigated to close the remaining robustness gap under strong iterative attacks.

All the Declarations and Statements

Author Contributions Statement

Nayankumar M. Mali – Developed the research concept and proposed the multi-modal malware detection framework. Designed the overall system architecture integrating the Temporal Sequence Encoder, Graph Structural Encoder, Parametric Anomaly Detector, and Attention-Based Hybrid Fusion Layer. Implemented the complete deep learning framework, performed data preprocessing, feature engineering, adversarial data augmentation, model training, hyperparameter optimization, and experimental evaluation. Conducted comparative analysis, ablation studies, scalability experiments, and robustness evaluation. Prepared figures, algorithms, mathematical formulations, and wrote the original manuscript.

Narendrasinh C. Chauhan – Provided overall supervision throughout the research work. Contributed to refining the research methodology, validating the proposed framework, interpreting experimental results, and improving the technical quality of the manuscript. Reviewed and edited the manuscript for scientific accuracy, clarity, and presentation. Guided the research direction and coordinated project planning and execution.

All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding Declaration

The authors received no financial support from any funding agency in the public, commercial, or not-for-profit sectors for the research, authorship, and publication of this article.

Data Availability Statement

The datasets analyzed during the current study were obtained from publicly available malware repositories, including EMBER, BODMAS, CIC-MalMem-2022, and adversarial malware datasets used for robustness evaluation, as described in the manuscript. The samples used in this work constitute a carefully selected subset of these repositories,

chosen to represent diverse malware families and obfuscation techniques relevant to the research objectives, and were analyzed and labelled on an experimental basis as part of this study. Processed datasets, source code, trained models, and experimental configurations supporting the findings of this study are available from the corresponding author upon reasonable request for academic and research purposes. The authors share these materials in good faith to support research upon reasonable request for academic and research purposes.

Ethical Declarations

This study is based entirely on publicly available malware datasets, simulation, and computational experiments. The research does not involve human participants, human biological materials, animals, clinical trials, or personally identifiable information. Therefore, ethical approval and informed consent were not required.

Acknowledgments

The authors express their sincere gratitude to the Department of Information Technology, A. D. Patel Institute of Technology, CVM University, Anand, Gujarat, India, for providing the research environment and computational support required to conduct this work. The authors also thank the anonymous reviewers and domain experts whose valuable comments and constructive suggestions significantly improved the quality, clarity, and technical rigor of this manuscript.

Declaration of Generative AI in Scholarly Writing

During the preparation of this manuscript, the authors used Grammarly solely for grammar checking, spelling correction, and language refinement to improve the readability of the manuscript. No generative artificial intelligence system was used to generate scientific content, research findings, experimental results, mathematical formulations, or conclusions. The authors carefully reviewed, edited, and verified all contents and take full responsibility for the accuracy, originality, and integrity of the published work.

Abbreviations

The following abbreviations are used in this manuscript:

API - Application Programming Interface
ML - Machine Learning
DL - Deep Learning
DNN - Deep Neural Network
LSTM - Long Short-Term Memory
BiLSTM - Bidirectional Long Short-Term Memory
GNN - Graph Neural Network
GAT - Graph Attention Network
GCN - Graph Convolutional Network
CNN - Convolutional Neural Network
MLP - Multi-Layer Perceptron
NT-Xent - Normalized Temperature-scaled Cross Entropy Loss
FGSM - Fast Gradient Sign Method
PGD - Projected Gradient Descent
ROC - Receiver Operating Characteristic
AUC - Area Under the Curve
F1 - Harmonic Mean of Precision and Recall
PE - Portable Executable
CFG - Control Flow Graph
DAG - Directed Acyclic Graph
GPU - Graphics Processing Unit
CPU - Central Processing Unit
CUDA - Compute Unified Device Architecture
ReLU - Rectified Linear Unit
QoS - Quality of Service
SOC - Security Operations Centre
EDR - Endpoint Detection and Response
IDS - Intrusion Detection System
IoC - Indicator of Compromise
C2 - Command and control
OOV - Out-of-Vocabulary
QKV - Query-Key-Value

AdamW - Adaptive Moment Estimation with Weight Decay

Appendix

None.

References

[1] J. Singh and J. Singh, “Challenge of Malware Analysis: Malware obfuscation Techniques,” International Journal of Information Security Science, vol. 7, no. 3, pp. 100–110, Sept. 2018.

[2] Asghar, H.J., Zhao, B.Z.H., Ikram, M. et al. Use of cryptography in malware obfuscation. J Comput Virol Hack Tech 20, 135–152 (2024). https://doi.org/10.1007/s11416-023-00504-y

[3] Li, Yang, Fei Kang, Hui Shu, Xiaobing Xiong, Yuntian Zhao, and Rongbo Sun. 2023. "APIASO: A Novel API Call Obfuscation Technique Based on Address Space Obscurity" Applied Sciences 13, no. 16: 9056. https://doi.org/10.3390/app13169056

[4] Z. Chen, E. Brophy, and T. E. Ward, "Malware Classification Using Static Disassembly and Machine Learning," TechRxiv, preprint, Dec. 2021, doi: 10.36227/techrxiv.17259806.v1. Available: arXiv:2201.07649.

[5] V. E. Kotov and M. Wojnowicz, "Towards Generic Deobfuscation of Windows API Calls," arXiv preprint arXiv:1802.04466, Feb. 2018. doi: 10.48550/arXiv.1802.04466. Available: https://arxiv.org/abs/1802.04466

[6] A. A. Alhashmi et al., “Hybrid Malware Variant Detection Model with Extreme Gradient Boosting and Artificial Neural Network Classifiers,” Comput. Mater. Contin., vol. 76, no. 3, pp. 3483–3498, 2023. https://doi.org/10.32604/cmc.2023.041038

[7] Brezinski, Kenneth, Ferens, Ken, Metamorphic Malware and Obfuscation: A Survey of Techniques, Variants, and Generation Kits, Security and Communication Networks, 2023, 8227751, 41 pages, 2023. https://doi.org/10.1155/2023/8227751

[8] X. Wei, C. Li, Q. Lv, N. Li, D. Sun, and Y. Wang, "Mitigating the Impact of Malware Evolution on API Sequence-based Windows Malware Detector," arXiv preprint arXiv:2408.01661, Aug. 2024, revised Nov. 2025. doi: 10.48550/arXiv.2408.01661.

[9] E. Amer, S. El-Sappagh, and J. W. Hu, "Contextual Identification of Windows Malware through Semantic Interpretation of API Call Sequence," Applied Sciences, vol. 10, no. 21, Art. no. 7673, 2020, doi: 10.3390/app10217673.

[10] Song, Y., Zhang, D., Wang, J. et al. Application of deep learning in malware detection: a review. J Big Data 12, 99 (2025). https://doi.org/10.1186/s40537-025-01157-y

[11] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, "API-MalDetect: Automated malware detection framework for windows based on API calls and deep learning techniques," Journal of Network and Computer Applications, vol. 218, Art. no. 103704, 2023, doi: 10.1016/j.jnca.2023.103704.

[12] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware Dynamic Analysis Evasion Techniques: A Survey," ACM Computing Surveys, vol. 52, no. 6, Art. no. 126, Nov. 2019, doi: 10.1145/3365001.

[13] M. F. Zolkipli and A. Jantan, "A Framework for Defining Malware Behavior Using Run Time Analysis and Resource Monitoring," in Digital Forensics and Cyber Crime, Communications in Computer and Information Science (CCIS), vol. 179, K. Lui, C. K. Yeo, and K. S. Yap, Eds. Berlin, Germany: Springer, 2011, pp. 199–209, doi: 10.1007/978-3-642-22170-5_18.

[14] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "OPEM: A Static-Dynamic Approach for Machine-Learning-Based Malware Detection," in Computational Intelligence in Security for Information Systems (CISIS 2012), Advances in Intelligent Systems and Computing, vol. 189. Berlin, Germany: Springer, 2013, pp. 271–280, doi: 10.1007/978-3-642-33018-6_28.

[15] M. F. Rafique, M. Ali, A. S. Qureshi, A. Khan, and A. M. Mirza, "Malware Classification Using Deep Learning Based Feature Extraction and Wrapper Based Feature Selection Technique," arXiv preprint arXiv:1910.10958, Oct. 2019, revised Dec. 2020. doi: 10.48550/arXiv.1910.10958.

[16] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," Journal of Network and Computer Applications, vol. 153, Art. no. 102526, 2020, doi: 10.1016/j.jnca.2019.102526.

[17] S. Saad, W. Briguglio, and H. Elmiligi, "The Curious Case of Machine Learning in Malware Detection," in Proc. 5th International Conference on Information Systems Security and Privacy (ICISSP), Prague, Czech Republic, 2019, pp. 85–94, doi: 10.5220/0007677600850094.

[18] MITRE, “Obfuscated Files or Information: Dynamic API Resolution (T1027.007),” MITRE ATT&CK, 2022. [Online]. Available: https://attack.mitre.org/techniques/T1027/007/

[19] C. Lim, Y. S. Kotualubun, S. Suryadi, and K. Ramli, "Mal-Xtract: Hidden Code Extraction Using Memory Analysis," Journal of Physics: Conference Series, vol. 801, no. 1, Art. no. 012058, 2017, doi: 10.1088/1742-6596/801/1/012058.

[20] J. Ferdous, R. Islam, M. Bhattacharya, and M. Z. Islam, "Malware Resistant Data Protection in Hyper-connected Networks: A Survey," arXiv preprint arXiv:2307.13164, Jul. 2023, doi: 10.48550/arXiv.2307.13164.

[21] G. Karat, J. M. Kannimoola, N. Nair, A. Vazhayil, V. G. Sujadevi, and P. Poornachandran, "CNN-LSTM Hybrid Model for Enhanced Malware Analysis and Detection," Procedia Computer Science, vol. 233, pp. 492–503, 2024, doi: 10.1016/j.procs.2024.03.239.

[22] Y. Kawakoya, E. Shioji, M. Iwamura, and J. Miyoshi, "API Chaser: Taint-Assisted Sandbox for Evasive Malware Analysis," Journal of Information Processing, vol. 27, pp. 297–314, 2019, doi: 10.2197/ipsjjip.27.297.

[23] H. Yin, B. Lou, and P. Reiher, "A Method for Summarizing and Classifying Evasive Malware," in Proc. 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), 2023, pp. 455–470, doi: 10.1145/3607199.3607207.

[24] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel, "When Malware is Packin' Heat: Limits of Machine Learning Classifiers Based on Static Analysis Features," in Proc. Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, Feb. 2020, doi: 10.14722/ndss.2020.24310
[25] B. Ndibanje, K. H. Kim, Y. J. Kang, H. H. Kim, T. Y. Kim, and H. J. Lee, "Cross-Method-Based Analysis and Classification of Malicious Behavior by API Calls Extraction," Applied Sciences, vol. 9, no. 2, Art. no. 239, 2019, doi: 10.3390/app9020239.
[26] O. A. Madamidola, F. Ngobigha, and A. Ez-zizi, "Detecting New Obfuscated Malware Variants: A Lightweight and Interpretable Machine Learning Approach," Intelligent Systems with Applications, vol. 25, Art. no. 200472, 2025, doi: 10.1016/j.iswa.2024.200472.
[27] S. Chandran, S. R. Syam, S. Sankaran, T. Pandey, and K. Achuthan, "From Static to AI-Driven Detection: A Comprehensive Review of Obfuscated Malware Techniques," IEEE Access, vol. 13, pp. 74335–74358, 2025, doi: 10.1109/ACCESS.2025.3550781.
[28] M. Naseer, F. Ullah, S. Ijaz, H. Naeem, A. Alsirhani, G. N. Alwakid, and A. Alomari, "Obfuscated Malware Detection and Classification in Network Traffic Leveraging Hybrid Large Language Models and Synthetic Data," Sensors, vol. 25, no. 1, Art. no. 202, 2025, doi: 10.3390/s25010202.
[29] S. P. Sharmila, A. Tiwari, and N. S. Chaudhari, "Obfuscated Memory Malware Detection," arXiv preprint arXiv:2408.12866, Aug. 2024, doi: 10.48550/arXiv.2408.12866.
[30] T. Carrier, P. Victor, A. Tekeoglu, and A. H. Lashkari, "Detecting Obfuscated Malware Using Memory Feature Engineering," in Proc. 8th International Conference on Information Systems Security and Privacy (ICISSP), 2022, pp. 177–188, doi: 10.5220/0010908200003120.
[31] R. H. Mahdi and H. Trabelsi, "Effective obfuscated malware detection leveraging cutting-edge machine and deep learning approaches," Int. J. Intell. Eng. Syst., vol. 18, no. 1, pp. 1045–1057, 2025, doi: 10.22266/ijies2025.0229.75.
[32] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," arXiv preprint arXiv:1804.04637, Apr. 2018, doi: 10.48550/arXiv.1804.04637.
[33] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "BODMAS: An Open Dataset for Learning Based Temporal Analysis of PE Malware," in Proc. IEEE Symposium on Security and Privacy Workshops (SPW), 2021, pp. 78–84, doi: 10.1109/SPW53761.2021.00020.
[34] Tristan Carrier, Princy Victor, Ali Tekeoglu, Arash Habibi Lashkari," Detecting Obfuscated Malware using Memory Feature Engineering", The 8th International Conference on Information Systems Security and Privacy (ICISSP), 2022
[35] X. Liu, J. Zhang, Y. Lin, and H. Li, "ATMPA: Attacking Machine Learning-Based Malware Visualization Detection Methods via Adversarial Examples," in Proc. IEEE/ACM 27th International Symposium on Quality of Service (IWQoS), Phoenix, AZ, USA, Jun. 2019, Art. no. 38, doi: 10.1145/3326285.3329073.
[36] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, "API-MalDetect: Automated Malware Detection Framework for Windows Based on API Calls and Deep Learning Techniques," Journal of Network and Computer Applications, vol. 218, Art. no. 103704, 2023, doi: 10.1016/j.jnca.2023.103704.
[37] S. Zhang, M. Gao, L. Wang, S. Xu, W. Shao, and R. Kuang, "A Malware-Detection Method Using Deep Learning to Fully Extract API Sequence Features," Electronics, vol. 14, no. 1, Art. no. 167, 2025, doi: 10.3390/electronics14010167.
[38] G. Karat, J. M. Kannimoola, N. Nair, A. Vazhayil, V. G. Sujadevi, and P. Poornachandran, "CNN-LSTM Hybrid Model for Enhanced Malware Analysis and Detection," Procedia Computer Science, vol. 233, pp. 492–503, 2024, doi: 10.1016/j.procs.2024.03.239.
[39] C. K. Patanaik, F. A. Barbhuiya, and S. Nandi, "Obfuscated Malware Detection Using API Call Dependency," in Proc. 1st International Conference on Security of Internet of Things (SecurIT), Kollam, India, 2012, pp. 185–193, doi: 10.1145/2490428.2490454.
[40] C. K. . Yuk and C. J. . Seo, "Static Analysis and Machine Learning-based Malware Detection System using PE Header Feature Values", ijirss, vol. 5, no. 4, pp. 281–288, Oct. 2022.

Authors' Profiles



Nayankumar M. Mali is an Assistant Professor in the Department of Information Technology at A. D. Patel Institute of Technology (ADIT). He is currently pursuing a Ph.D. in Malware Behavior Analysis and Detection using Machine Learning. His research interests include cybersecurity, malware detection, web security, and applied machine learning. He has guided MTech dissertations, mentored award-winning hackathon teams, and contributed to industry-based projects. With expertise in Java, Spring Boot, Python, and cybersecurity, he has delivered expert talks, conducted FDPs, and developed academic web portals. He is also an active member of CoE in Information Network Security and a consultant for IT projects



Narendrasinh C. Chauhan is a Professor and Head of the Department of Information Technology at A. D. Patel Institute of Technology, CVM University. He holds a Ph.D. in Electronics Computer Engineering from IIT Roorkee (2010) and has over 20 years of teaching and research experience. His research expertise spans artificial intelligence, machine learning, data analytics, soft computing, and image processing. He has published over 30 international journal papers, authored books, and guided multiple Ph.D. and M.E. students. Actively involved in funded RD projects, he also contributes as a recognized Ph.D. supervisor and expert speaker in national and international conferences.

How to cite this paper:

Nayankumar M. Mali, Narendrasinh C. Chauhan, "A Multi-Modal Deep Learning Framework for Robust Detection of Obfuscated API Calls in Malware Analysis", International Journal of Computer Network and Information Security(IJCNIS), Vol.18, No.5, pp. 119-142, 2026. DOI:10.5815/ijcnis.2026.05.08