Research Article | Open Access

Intelligence & Robotics

Ren *et al. Intell. Robot.* **2026**, *6*(2), 253-74 **DOI:**10.20517/ir.2026.13

Leveraging intelligent multimodal fusion for few-shot malware classification

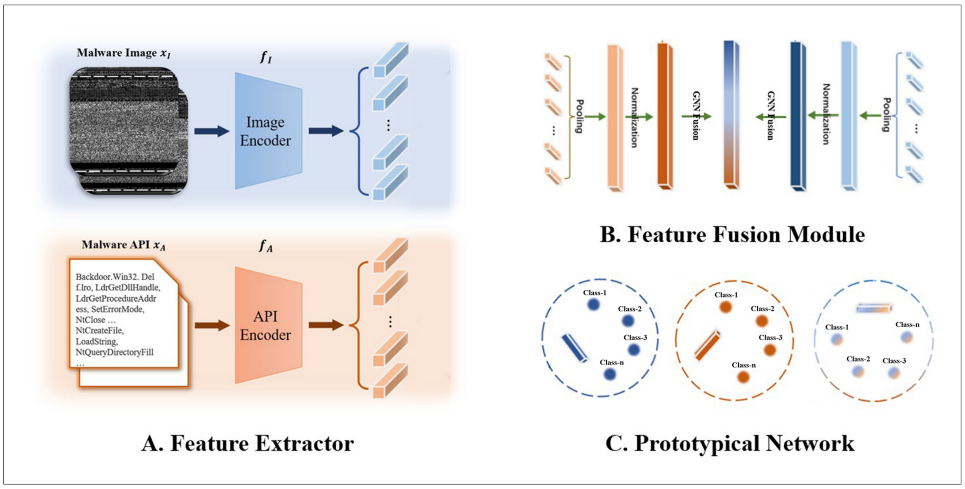
Ying Ren[1], Ziyu Liu[2], Junbo Wang[3], Peng Wang[4]

Keywords:
Intelligent multimodal learning, multimodal feature fusion, malware classification, few-shot learning

Citation: Ren, Y.; Liu, Z.; Wang, J.; Wang, P. Leveraging intelligent multimodal fusion for few-shot malware classification. *Intell. Robot.* **2026**, *6*(2), 253-74. https://dx.doi.org/10.20517/ir.2026.13

Received: 9 Dec 2025 **First Decision:** 13 Feb 2026 **Revised:** 27 Mar 2026 **Accepted:** 22 May 2026 **Published:** 12 Jun 2026

Academic Editor: Simon Yang **Copy Editor:** Pei-Yun Wang **Production Editor:** Pei-Yun Wang



Abstract

Traditional malware classification methods heavily rely on extensive labeled data and single modal features, which limits their adaptability to evolving threats. In this paper, we propose an intelligent multimodal fusion framework that leverages complementary information from static and dynamic analysis for few-shot malware classification. Specifically, we convert malware binaries into grayscale images to capture static characteristics and extract application programming interface (API) call sequences to represent dynamic behaviors. To effectively integrate these heterogeneous modalities under limited data conditions, we introduce a lightweight graph neural network-based intelligent feature fusion module. This module segments modality-specific features, constructs a bipartite graph between segments, and performs cross-modal message passing to learn fine-grained correlations. The fused representations are then used in a prototypical network for few-shot classification. We construct two malware datasets augmented with multimodal features and conduct extensive experiments under few-shot settings. Results demonstrate that our approach significantly outperforms both unimodal baselines and naive fusion methods, achieving up to 95.73% accuracy in 5-way 5-shot classification. Ablation studies and efficiency analysis confirm that our fusion module adds

[1]Department of Outpatient, West China Hospital, Sichuan University, Chengdu 610065, Sichuan, China.
[2]The Second Affiliated Hospital of Zhejiang University School of Medicine, Hangzhou 310058, Zhejiang, China.
[3]College of Software Engineering, Sichuan University, Chengdu 610065, Sichuan, China.
[4]College of Computer Science, Sichuan University, Chengdu 610065, Sichuan, China.

Correspondence to: Prof. Ying Ren, Department of Outpatient, West China Hospital, Sichuan University, Chengdu 610065, Sichuan, China. E-mail: zq156157@163.com

minimal computational overhead while enhancing both accuracy and interpretability. This work highlights the potential of intelligent multimodal integration for robust malware classification with limited labeled data.

1. INTRODUCTION

Cybersecurity has emerged as a critical concern in the face of increasing Internet usage. Among the numerous cyber threats, malware stands out as a persistent and constantly evolving menace that poses continuous challenges to the security of computer systems and networks. The term "malware" encompasses a wide range of malicious software, including viruses, worms, Trojans, ransomware, and spyware, among others[1]. These malicious programs are specifically designed to infiltrate, disrupt, or compromise the integrity of computer systems, often resulting in data breaches, financial losses, and even the compromise of critical infrastructure. Thus, the development of robust and adaptive malware classification systems is imperative in safeguarding digital ecosystems.

Traditionally, malware classification has relied on signature-based detection methods, where malware samples are matched against predefined signatures or patterns. However, this approach is limited in effectiveness, as it fails to detect new or emerging malware variants, particularly those with limited labeled samples, a situation that poses a significant threat to cybersecurity. It is important to clarify that few-shot learning differs fundamentally from zero-day detection. While zero-day detection aims to identify completely unseen threats without any prior labeled samples, few-shot learning assumes the availability of a small number of labeled examples for new classes. In this work, we focus on the few-shot setting, where the goal is to effectively leverage limited labeled data to classify new or emerging malware families. This scenario is practically relevant in cybersecurity operations, where security analysts can often obtain a few samples of a new threat through initial incident response or threat intelligence sharing. In response, the cybersecurity community has been actively exploring advanced techniques such as machine learning and deep learning to enhance the accuracy and agility of malware classification. These approaches primarily utilize various neural networks to extract features for classification[2-5]. Nonetheless, traditional machine learning models often struggle to keep pace with the ever-changing landscape of malicious software due to the need for frequent retraining on large-scale datasets.

One promising avenue of research that has gained considerable attention is the application of few-shot learning techniques to malware classification[6,7]. Few-shot learning refers to the ability of a model to learn and generalize from a limited number of examples, making it particularly suitable for detecting new malware samples when only a few labeled instances are available. This approach is crucial in the continually evolving cybersecurity landscape, where malware authors consistently create new variants to evade detection. Despite the impressive performance achieved by previous research, it has consistently relied on a single modality, such as visualized malware images or application programming interface (API) invocation sequences, which may provide only partial insights into malware behavior and structure.

Recent advances in few-shot learning for malware classification have explored various meta-learning and metric-learning paradigms. Wang *et al.* introduced a multi-prototype modeling approach to capture intra-class diversity in malware families[6]. More recently, Wang *et al.* developed AGProto, an adaptive graph prototypical network that adjusts prototypes based on sample relationships[7]. Beyond prototypical networks, transductive few-shot learning methods and data augmentation techniques specifically designed for malware have also been explored to address data scarcity. However, these approaches predominantly rely on unimodal features, limiting their ability to capture the full spectrum of malware characteristics.

Inspired by the observation that modern malware exhibits characteristics that extend beyond the confines of traditional data sources, this paper presents an intelligent multimodal fusion approach to malware classification that leverages multimodal information using few-shot learning. Our key insight is that static structural patterns (e.g., binary code converted to images) and dynamic behavioral traces (e.g., API call sequences) provide complementary views of malware, and intelligently fusing these modalities can yield more robust representations, especially under limited data conditions. Specifically, our proposed model combines both static features, such as image-based representations of malware binaries as visual clues, and dynamic features, such as API call sequences as textual information. This combination enables our model to capture diverse aspects of malware behavior and structure. To achieve this, we construct two datasets specifically tailored for few-shot malware classification. Furthermore, we devise a novel lightweight graph neural network (GNN)-based feature fusion module that operates on segmented features from both modalities, constructing a bipartite graph to enable cross-modal message passing. This design differs from prior multimodal fusion approaches that rely on simple concatenation or bilinear pooling, as it explicitly models inter-modal relationships at the segment level. The resulting fused multimodal features can be readily employed in various few-shot learning paradigms, particularly metric-based learning methods such as prototypical networks[8].

The major contributions of this work are as follows:
• We propose a novel lightweight GNN-based feature fusion module that segments modality-specific features, constructs a bipartite graph between segments, and performs cross-modal message passing to learn fine-grained correlations between static and dynamic malware characteristics.
• We introduce a modality-specific normalization strategy tailored for few-shot learning, which standardizes features from different modalities to a common scale using support set statistics, preventing modality dominance during fusion.
• We construct two few-shot malware classification datasets augmented with multimodal features, encompassing grayscale image features of static characteristics and dynamic API call sequence features, providing a benchmark for future research in this direction.
• We conduct extensive experimentation on two datasets, augmented with ablation studies and analyses, demonstrating that our approach outperforms both unimodal baselines and existing fusion methods while adding minimal computational overhead.

2. RELATED WORK

2.1. Malware classification

2.1.1. Static analysis

Static analysis is a primary technique in malware analysis, involving the extraction and selection of features from binary sequences, opcodes, function calls, printable strings, and other data found within executable malware files. These features are subsequently used for detection through machine learning or deep learning algorithms. Static analysis offers the benefits of effectiveness and efficiency but is also vulnerable to obfuscation and distortion techniques[9].

MalConv[10] initially processes the whole binary raw bytes into a deep learning network, utilizing convolutional neural networks (CNN) and recurrent neural networks (RNN) to extract features and perform classification. Gibert *et al.* examine the hierarchical nature of programs[11], and introduce a hierarchical convolutional network to analyze byte sequences as n-gram-like features[12].

Inspired by the recent huge progress in computer vision techniques, researchers focus on designing methods to better convert malware into images, facilitating recognition via deep learning methods[13,14]. Cui *et al.* propose to translate binary bytes from executable files into a grayscale image, where values are within the

range of 0 to 255[13]. Yuan *et al.* introduce a novel approach that transfers relationships within byte sequences to construct Markov images, capturing the interplay between malicious software bytes[15]. Sharma *et al.* compare grayscale, color, and Markov images for analyzing malicious software[16]. They propose the use of Gabor filtering to extract textures, identify focal areas, and recognize distinctive features. Moreover, the opcodes[17], function calls[18,19], and printable strings[20] obtained after decompilation of malicious software are also used for analysis.

2.1.2. Dynamic and hybrid analysis

Dynamic analysis gathers data on the real-time execution behavior of samples by actively running and monitoring programs, including system call APIs[5,21-25], network traffic[26,27], and file interactions[28], to comprehensively capture and scrutinize the program's operational intent, facilitating the identification and classification of its malicious characteristics[29]. In contrast to static analysis, dynamic analysis incurs a higher analytical cost and can be evaded by some anti-virtualization, anti-sandbox, or time-triggered malware, while providing more comprehensive information[30].

Besides, to counter malware evasion techniques, many works combine static and dynamic features for analysis to comprehensively represent the information of malware[31]. Specifically, Yoo *et al.* extract various static and dynamic features, such as file size, header size, number of APIs, information entropy of different segments, API sequence, file information, locks, registry, and more, to make decisions based on machine learning methods[32]. Nguyen *et al.* conduct zombie software analysis with printable string information[33]. O'Shaughnessy *et al.* combine static malware executable files and dynamic process memory dump information for malware analysis with the KNN-HOG model[34].

2.1.3. Multimodal analysis

Static and dynamic analyses have demonstrated strong potential in malware research. However, these methods are susceptible to failure due to the fast evolution of malware characterized by techniques like obfuscation, evasion, and fuzzing. Consequently, researchers have turned their attention towards multimodal approaches, seeking to integrate information across diverse modalities for enhanced efficacy. Kim *et al.* introduce multimodal deep learning for Android malware analysis, utilizing Android manifest, dex, and .so files to extract distinct features[35].

Similarly, Gibert *et al.* treat the API sequence, bytecode, and opcode of malicious software as three modal data types, designing dedicated deep neural network modules to achieve more effective enhancements[36]. Dib *et al.* utilize two modalities, grayscale image representations from binary files and readable characters as a text representation from disassembled code, for malware classification[37].

Recent advances have also explored more sophisticated multimodal frameworks to address the evolving nature of malware. He *et al.* proposed DREAM, a system that combines classifier and expert knowledge within a unified model to combat concept drift in Android malware classification[38]. Their approach embeds malware behavioral concepts within the latent space of a contrastive autoencoder while constraining sample reconstruction based on classifier predictions, enabling more effective drift detection and adaptation. DREAM integrates both static and dynamic behavioral concepts, demonstrating the value of incorporating expert knowledge into multimodal learning.

Chai *et al.* introduced MalFSCIL, a few-shot class-incremental learning framework for malware detection that addresses the challenges of catastrophic forgetting and decision boundary confusion[39]. Their method employs a decoupled training strategy combining a Variational Autoencoder (VAE) for feature enhancement

with graph attention networks for dynamic boundary delineation based on class prototypes. MalFSCIL demonstrates the effectiveness of integrating generative models with incremental learning in adapting to new malware families with limited samples.

Different from previous works, in this paper we utilize both static information (grayscale image converted from binary files) and dynamic analysis (API invocation sequences) as visual and textual modalities to obtain comprehensive representations. While DREAM focuses on concept drift detection and MalFSCIL addresses class-incremental learning, we propose a novel GNN-based fusion network specifically designed for few-shot malware classification. Unlike these approaches that primarily focus on single-modality enhancement or detection paradigms, our method explicitly models cross-modal correlations at the segment level, enabling fine-grained integration of complementary information from static and dynamic analysis.

2.2. Multimodal feature fusion

Generally, multimodal feature fusion can benefit both unimodal tasks, e.g., image classification assisted by text[40,41], and multimodal tasks such as visual question answering (VQA)[42] and image caption generation[43,44]. In early studies, Antol *et al.*[42] utilized VGGNet[45] and LSTM[46] to extract visual and textual features respectively and fused them with simple mechanisms such as addition, concatenation, and multiplication. Stacked attention network (SAN)[47] designed for VQA is to progressively search for related image regions using question semantic representations. Based on low-rank bilinear pooling, bilinear attention network (BAN)[48] generates bilinear attention maps to fuse multimodal features. These two fusion methods are also employed in this work. Recently, Transformer[49] based vision-and-language fusion becomes a popular paradigm. A typical process is to extract text features with BERT[50] and fuse them with visual features via the self-attention mechanism. In this paper, we tailor Transformer-based architecture for fusing multimodal representations of malware.

3. DATASET CONSTRUCTION

3.1. Dataset description

We construct two datasets for few-shot malware classification: VirusShare-M and LargePE-M. Both contain Windows PE malware samples with associated grayscale images and API call sequences.

VirusShare-M: Malware samples are downloaded from VirusShare and labeled using AVClass based on VirusTotal reports. We select multiple families and split them into meta-training, meta-validation, and meta-test sets with disjoint family labels. Detailed family names are provided in the Supplementary Materials.

LargePE-M: Samples are collected from a large in-house repository of Windows PE malware. Families with sufficient samples that executed successfully (i.e., those that generated API sequences in the Cuckoo sandbox) are selected and split into disjoint training, validation, and test sets by family. Detailed family names are provided in the Supplementary Materials.

Data Leakage Prevention: To ensure valid few-shot evaluation, we take three precautions: (1) training, validation, and test sets contain completely disjoint malware families; (2) all preprocessing steps [term frequency-inverse document frequency (TF-IDF) calculation, word2vec training] are performed solely on the training set; (3) N-gram vocabulary and TF-IDF weights are derived exclusively from training samples. These measures prevent information leakage and ensure that evaluation reflects generalization to unseen families.

Detailed family names and class distributions for both datasets are provided in the Supplementary Tables 1 and 2.

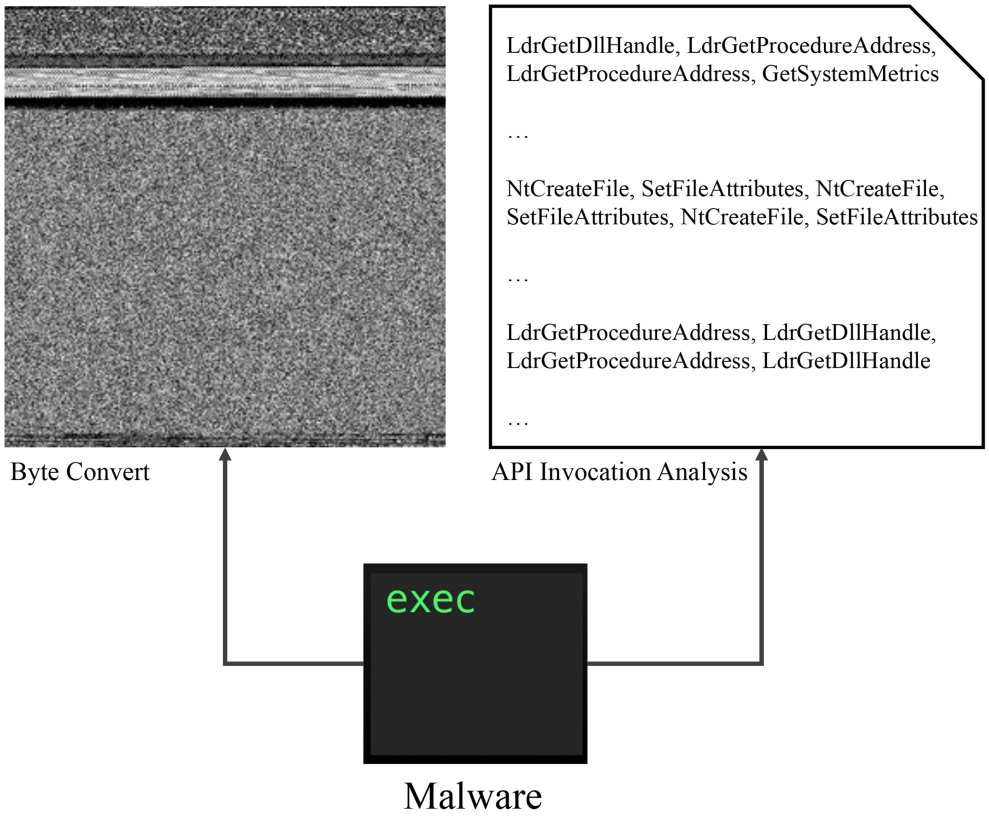


Figure 1. Convert executable malware into two different information sources in two modalities.

Table 1. Image width corresponding to converted malware of different sizes

File size range	Image width
< 10 kB	32
10-30 kB	64
30-60 kB	128
60-100 kB	256
100-200 kB	384
200-500 kB	512
500-1,000 kB	768
> 1,000 kB	1,024

3.2. Visual malware image conversion

Static grayscale image features of malware have been extensively utilized in related research on malware detection and classification[2,51,52]. We follow the approach proposed by Nataraj *et al.*, wherein each byte (8 bits) of malware binary code is transformed into an integer within the range of 0-255[51]. Subsequently, the width of the image is determined based on the size of the malware sample (as outlined in Table 1). To minimize information loss, we add zeros to the last row of pixels when the required length is not achieved. Finally, the images are resized to a standardized dimension of 256 × 256 pixels, and the pixel values are normalized to obtain the final grayscale images (as depicted in Figure 1 left).

Table 2. Hyperparameters for API sequence processing

Type	Hyperparameter	Value
Preprocessing	Number of TF-IDF top items *k*	2,000
	Maximum sequence length *l*	300
	N-gram window size	3
word2vec	Embedding-dim	300
	Learning rate	0.025 → 0.001
	Training epoch	5

API: Application programming interface; TF-IDF: term frequency-inverse document frequency.

3.3. Textual API sequence generation

To capture the behavioral characteristics of malicious software, we extract API invocation sequences from malware samples using a feature extraction process similar to the one described by Wang *et al.*[6]. Specifically, we executed malware samples within a virtual environment using the Cuckoo sandbox (https://cuckoosandbox.org/). This execution generated a detailed report, which includes the API invocation sequences, as illustrated in Figure 1 (right). Within these Cuckoo-generated reports, we selectively retain only the API names (e.g. "LdrLoadDll", "LdrGetProcedureAddress", "LdrGetDllHandle"), while disregarding both the parameters and returned values associated with each API invocation. Given that malware may initiate multiple processes concurrently, we concatenate APIs invoked by different processes to linearize the API sequence. Malware samples with sequence lengths less than 10 are excluded, which often indicates that the malware failed to run properly within the virtual environment.

Following the acquisition of malware API invocation sequences, refinement is necessary to eliminate redundant features resulting from repeated executions, such as file loading or execution loops. Consistent with the preprocessing methodology outlined in Wang *et al.*, we omit redundant API subsequences wherein the same API appears more than twice consecutively[6].

For tokenizing API sequences tailored for extracting textual features, we adopt an N-gram approach, which employs a sliding window operation of size N on the API sequence. Specifically, we divide the API sequence of length a into $(a - N + 1)$ N-gram items, treating the resultant N-gram items as new sequence features. This expansion increases the scale of the sequence's word dictionary from α to α^N if there are α unique APIs in the original sequences. Subsequently, we compute the importance of each N-gram using the TF-IDF method [Equation (1)].

$$TF-IDF(\alpha_i) = TF(\alpha_i) \cdot IDF(\alpha_i) = \frac{n_i}{\sum_j n_j} \cdot \log\left(\frac{n}{f(\alpha_i)+1}\right) \tag{1}$$

Here, α_i represents the i-th N-gram item, n_i denotes the frequency of occurrence of the i-th N-gram, and $f(\alpha_i)$ signifies the count of malware samples containing the i-th N-gram item. Following the computation of the importance of each N-gram item, we filter out the top-k N-grams based on their weights. Subsequently, for each sample's N-gram sequence, we truncate the first l N-gram items to extract the sample's features. Finally, we employ the widely used word2vec technique from the field of natural language processing (NLP) to embed these N-gram items. Specifically, we utilize a skip-gram model for pretraining, and the parameters of the word2vec model, along with the hyperparameters of the aforementioned feature extraction process, are detailed in Table 2.

While our datasets are specifically designed for few-shot evaluation - where each class contains a limited number of samples - we acknowledge that real-world malware ecosystems are more complex. In practice, malware families exhibit long-tailed distributions, temporal evolution, and concept drift. Our current

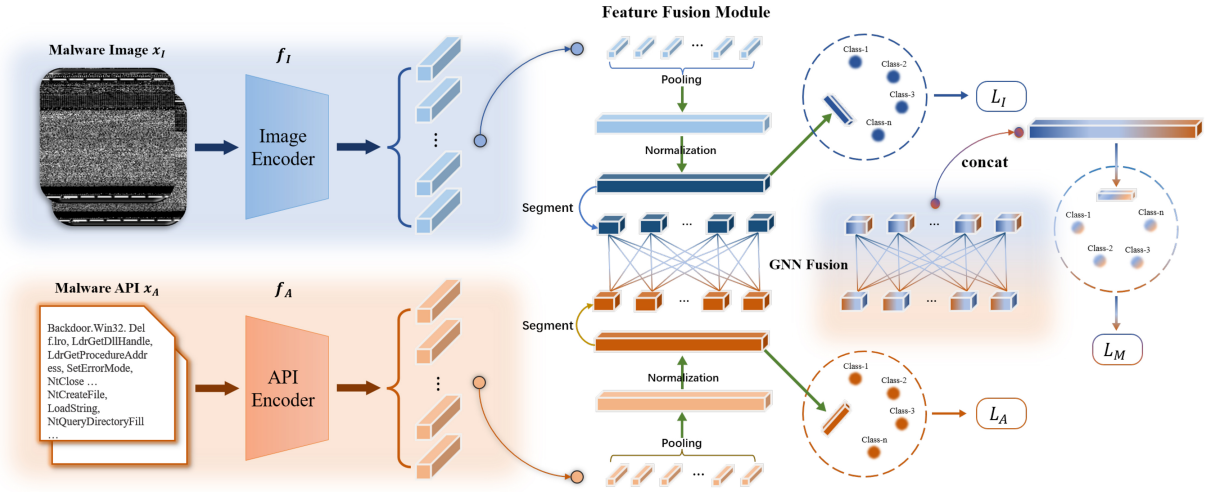


Figure 2. Framework of leveraging multimodal features for few-shot malware classification. Initially, malware images and API invocation sequences are derived from malware binary files and then fed into two distinct encoders, yielding respective unimodal features. Within the feature fusion module, features from both modalities undergo pooling and normalization separately before being segmented. The segmented features are then integrated through a GNN for feature fusion. Classification is then performed using a prototypical network. API: Application programming interface; GNN: graph neural network.

datasets, derived from curated families with balanced samples, do not fully capture these dynamics. However, they provide a controlled benchmark for evaluating few-shot learning algorithms under class-disjoint settings.

4. METHODS

In this section, we first present some preliminaries about few-shot malware classification and an overview of the proposed multimodal framework for this task. Then, we introduce the unimodal feature extraction module for each modality. Finally, we introduce our proposed multimodal feature fusion module based on GNN. The flow of the overall framework is shown in Figure 2.

4.1. Preliminaries

The goal of few-shot learning is to make accurate predictions when provided with only a limited amount of labeled data. This is particularly suitable for the classification of malware, as in most cases, it is difficult to obtain a large number of malware samples. To achieve this, we adopt the widely used meta-learning paradigm[53,54]. Given a malware dataset $\mathcal{D}$, we split it into three parts meta-training $\mathcal{D}^{train}$, meta-validation $\mathcal{D}^{val}$, and meta-test $\mathcal{D}^{test}$ with disjoint classes. The task in each part comprises a support set S and a query set Q that share the same label space.

The objective of meta-learning is to correctly classify samples in query set Q based on labeled samples in support set S into N classes. When support set S has N families in total and each family possesses K samples, this few-shot classification task is called N-way K-shot task. Generally, we employ meta-training $\mathcal{D}^{train}$ to train the model and meta-validation $\mathcal{D}^{val}$ to fine-tune hyperparameters and assess the model's generalization performance, and test the model's performance on unseen classes/families through meta-test $\mathcal{D}^{test}$.

4.2. Framework overview

The schematic representation of our proposed framework is depicted in Figure 2. When an image-API pair (x_I, x_A) is derived from a malware file using the methodologies outlined in Section 3, it is processed by dual encoders for feature extraction. Specifically, we propose employing a visual encoder $f_I(\cdot)$ to extract features

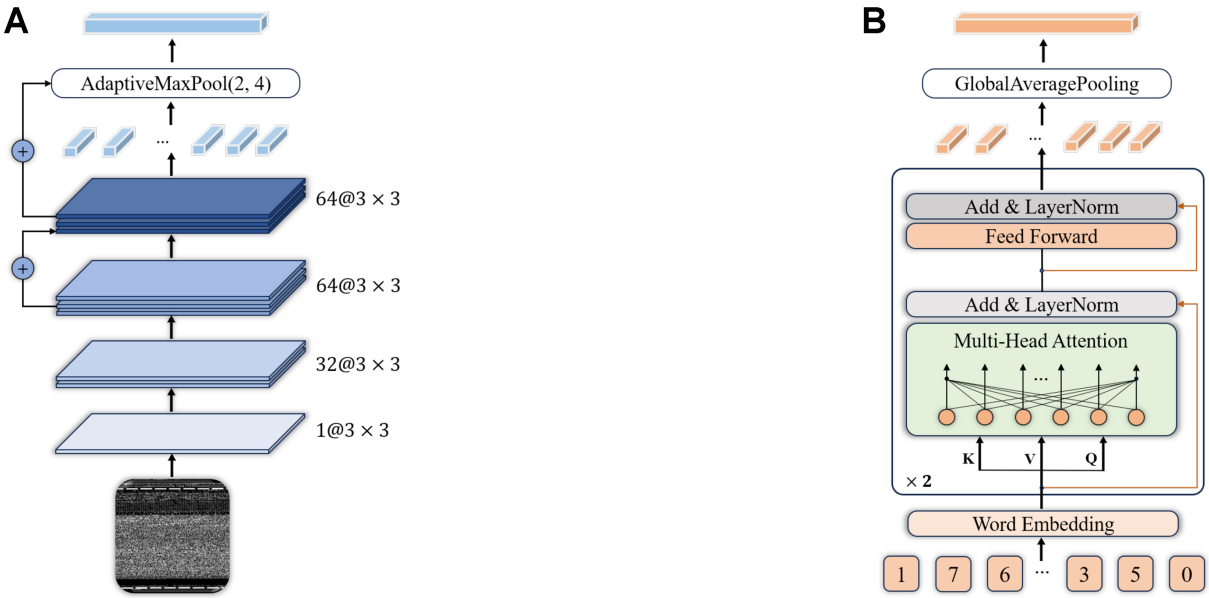


Figure 3. Proposed feature extractors for both modalities. (A) CNN4 architecture for malware image encoding, featuring 32 ⟶ 64 ⟶ 128 channel progression with residual connections and adaptive pooling; (B) Simple Transformer architecture for API sequence encoding, with two hidden layers and two attention heads. CNN: Convolutional neural network; API: application programming interface.

from the image x_I, and a textual encoder $f_A(\cdot)$ for the API sequence x_A. Subsequently, a multimodal feature fusion module is employed to integrate the feature information extracted from the grayscale image and API invocation sequence. For feature fusion, we segment the normalized features, apply a GNN to integrate them, and finally employ a prototypical network[8] for classification.

Support and Query Set Processing: For each task, support set samples S are encoded (E_I^S, E_A^S), normalized using support set statistics, segmented, and fused via GNN to produce V^S, which are aggregated into prototypes p_k. Query set samples Q undergo the same encoding, normalization (using same statistics), and fusion to produce V^Q, which are compared with prototypes via Euclidean distance: $\hat{y} = \text{argmin}_k \|V^Q - p_k\|_2$.

4.3. Feature extractor

To address the limited data in few-shot scenarios, we design two lightweight feature extractors for two modalities, with the expectation of reducing overfitting.

4.3.1. Malware image encoder

In Section 3.2, we discuss the methodology for converting malware into grayscale images. For the converted image x_I, a series of data augmentations - including resizing with a random crop and random rotation - is employed to enhance the diversity of the training images. For the embedding module, we choose the CNN4 backbone network[55], which is commonly used in few-shot image classification. Our CNN4 implementation differs slightly from the conventional architecture: it uses 32 channels in the first convolutional layer instead of the standard 64. This adjustment accounts for the input being grayscale images, initially containing only a single channel rather than the three channels found in RGB images. Gradually increasing the number of channels enhances feature extraction while reducing computational demands. Additionally, a residual connection is introduced between the last two CNN layers, a modification that has been shown to stabilize training and expedite model convergence. In the final layer, an AdaptiveMaxPool with dimensions (2, 4) is applied - considering that grayscale images contain more semantic information horizontally than vertically - resulting in an output feature dimension of 512. The structure of the Image Encoder is illustrated in Figure 3A.

4.3.2. Malware API Encoder

In Section 3.3, we describe the preprocessing methods... and obtain embeddings of each malware sample's API call sequence using Word2Vec. Following the lightweight design of the Image Encoder, we propose a streamlined Transformer architecture[49] as the API encoder $f_A(\cdot)$, featuring only two hidden layers, each with two attention heads. Specifically, the embedding dimension for each token is set to 300, and the dimension of the feedforward network is equally established at 300 to align with the output dimension of the Image Encoder. We use a dropout rate of 0.5 to mitigate overfitting. The architecture of the API Encoder is depicted in Figure 3B.

4.4. GNN-based feature fusion module

The preceding section describes the extraction methods for grayscale image features and API call sequence characteristics of malicious software. This section delineates the design of our devised lightweight feature fusion module based on GNNs. First, we normalize the pooled features independently and then segment them into discrete chunks, each representing a node in a graph. We then establish edges between nodes and perform message passing across the graph to achieve feature fusion. Finally, we employ a prototype network to calculate the distances within the fused feature space between samples and prototypes, thereby enabling classification. The pseudo code for training our proposed GNN-based feature fusion module over a single epoch is presented in Algorithm 1.

Algorithm 1 Train our network for one epoch

Require: Training dataset D^{train}
Require: Image encoders f_I, API encoders f_A
Require: P for aggregating class prototypes
Require: Distance function *dist* for calculating the distance between query samples and prototypes
Require: Parameters θ for the encoders
for batch = 1 **to** m **do**
Sample a task $\{S, Q, y\}$ from D^{train}
$x_I^S, x_A^S \leftarrow S;\ x_I^Q, x_A^Q \leftarrow Q$
$E_I^S \leftarrow f_I(x_I^S);\ E_A^S \leftarrow f_A(x_A^S);\ E_I^Q \leftarrow f_I(x_I^Q);\ E_A^Q \leftarrow f_A(x_A^Q)$
Normalize $E_I^S, E_A^S, E_I^Q, E_A^Q$
$E^S, E^Q \leftarrow \text{fuse}(E_I^S, E_A^S), \text{fuse}(E_I^Q, E_A^Q)$
protos $\leftarrow P(E^S)$
$logits \leftarrow -\text{dist}(E^Q, \text{protos})$
loss $\leftarrow$ Cross-entropy$(y, logits)$
$\theta \leftarrow \text{Adam}(\Delta\theta(\text{loss}), \theta)$
end for

Complexity Analysis: The computational complexity of Algorithm 1 is dominated by the feature extraction and GNN fusion steps. For a batch with N support samples and M query samples, the image encoder f_I has complexity $O((N + M)\cdot C_I)$ where C_I is the CNN4 complexity ($O(H\cdot W\cdot D)$). The API encoder f_A has complexity $O((N + M)\cdot L\cdot D\cdot H)$ where L is sequence length and H is number of attention heads. The GNN fusion module operates on $2N$ segments (where N is the number of segments per modality) with complexity $O(N^2\cdot D)$ for attention computation and $O(N\cdot D^2)$ for message passing. Overall, the per-epoch complexity is $O((N + M)\cdot(C_I + L\cdot D\cdot H) + N^2\cdot D + N\cdot D^2)$, which scales linearly with batch size and quadratically with the number of segments. In practice, with $N = 4$ segments and $D = 512$, the fusion overhead is negligible compared to feature extraction.

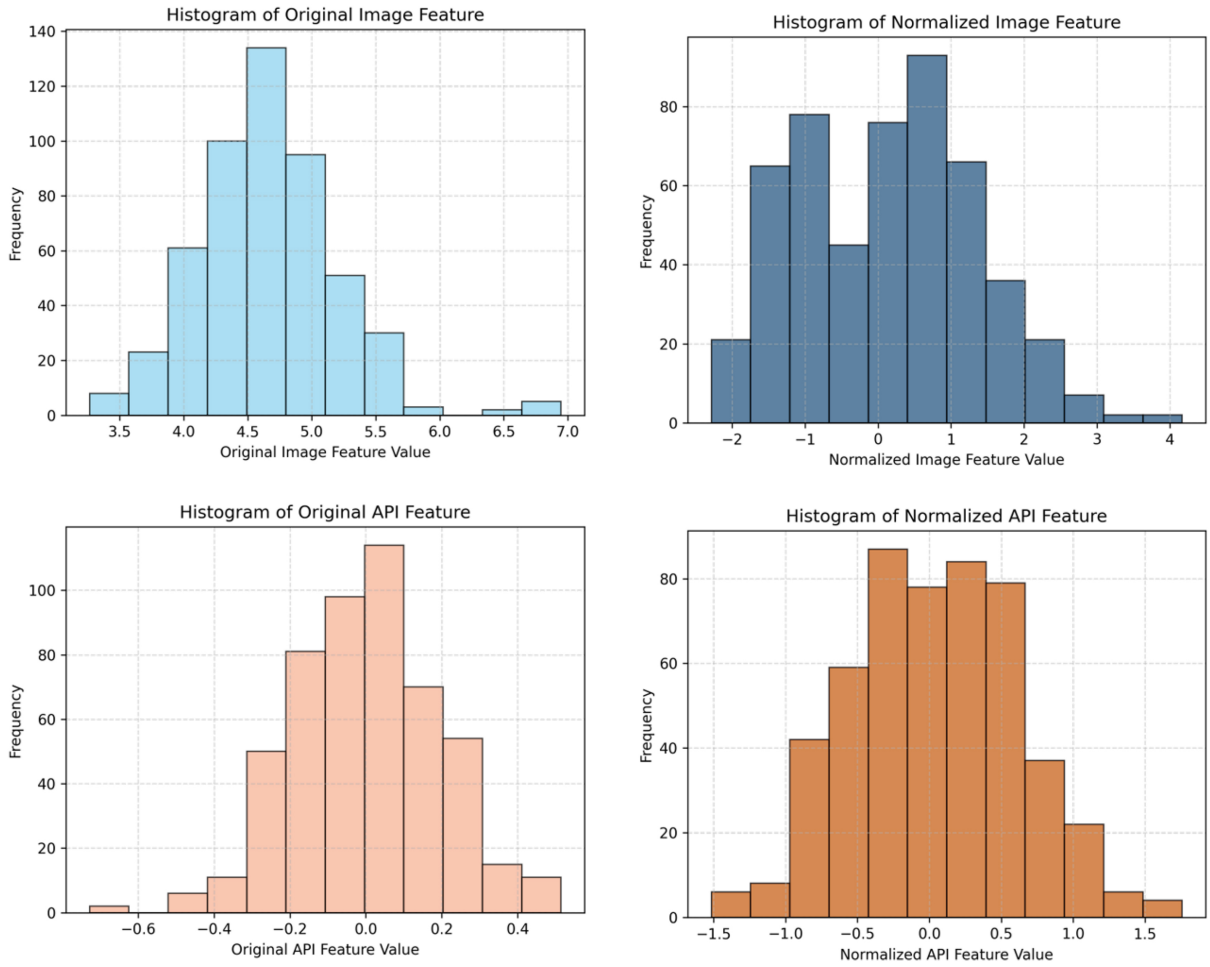


Figure 4. Comparison of value distribution before and after normalization of Image features and API sequence features. The histograms are computed from 1,000 randomly sampled feature vectors from the VirusShare-M validation set, showing the distribution shift after applying the proposed modality-specific normalization. API: Application programming interface.

4.4.1. Normalization

After extracting features from images and API sequences, both modalities are expected to contribute significantly to the final few-shot classification performance. However, as shown in Figure 4, the numerical ranges of features extracted by the Image Encoder and API Encoder differ significantly before fusion. These disparities would disproportionately bias the fusion process toward the modality with larger numerical values, even though the magnitude does not reflect the quality of the features, which could harm classification performance. To mitigate this issue and map the features of both modalities to a uniform scale, we standardize the features of both modalities to a mean of 0 and a standard deviation of 1, thereby aligning them with the standard normal distribution.

Given the unique characteristic of few-shot learning, where the samples in the support set are transparent to the model for each task, we leverage the mean and variance of the support set's feature data as prior information, applying it to the query set. This approach ensures that there is no information leakage between samples in the query set. Specifically, for the image features of the support set E_I^S, the mean and variance are calculated according to the following Equation:

$$\mu_s = \frac{1}{n}\sum_{i=1}^{n} E_{I,i}^{S} \tag{2}$$

$$\sigma_s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(E_{I,i}^{S} - \mu_s)^2} \tag{3}$$

Here, μ_s denotes the mean of the sample features within the support set S, with n representing the number of samples in the support set S, and $E_{I,i}^{S}$ signifying the feature of the i-th sample within the support set S. Upon determining the statistical measures of the support set's sample features, these mean and variance values are utilized to normalize the features of both the support set and query set samples.

$$E_I^S = (E_I^S - \mu_s)/\sigma_s \tag{4}$$

$$E_I^Q = (E_I^Q - \mu_s)/\sigma_s \tag{5}$$

As depicted in Figure 4, normalization of the two modalities' features results in a closer numerical range alignment, while preserving the relative magnitude of the original features.

4.4.2. Feature fusion GNN

We first describe how to construct the graph G on the basis of features from two modalities. Initially, we segment both the image embedding I and the API embedding A, both elements of $\mathbb{R}^{1\times D}$, into multiple parts. Each segment corresponds to a node within the graph, resulting in the node set $V = [I_1, I_2, \ldots, I_N, A_1, A_2, \ldots, A_N]$. Each node V_i embodies the feature information of a specific segment and is an element of $\mathbb{R}^{1\times K}$, where K denotes the length of each segment and is calculated as $K = D//N$. To facilitate the fusion of information across modalities, we introduce edges exclusively between segments of different modalities, thereby constructing a bipartite graph. Notably, segments within the same modality do not connect via edges. The graphical representation of this construction is depicted in Figure 2.

We then apply a graph message passing operation to the node set V. This yields the transformed feature set V', which contains the updated node features after message passing.

$$V' = \sigma(\mathcal{A}VW_{in})W_{out} + V \tag{6}$$

where W_{in} and W_{out} are the weights of fully-connected layers, σ is the activation function, $\mathcal{A}$ is the normalized adjacency matrix that is obtained by:

$$\mathcal{A} = Softmax(LeakyReLU(A)) \tag{7}$$

where the adjacency matrix A represents the connections of nodes within a graph. Next, we introduce how to generate A.

We define e_{ij} as a measure of the relationship between V_i and V_j. The underlying premise is that the closer or more relevant the nodes are to each other, the larger the value of e_{ij} should be. The formula for this relationship measure can be represented as follows:

$$e_{ij} = f_\theta(V_i, V_j) \tag{8}$$

Here, f_θ is defined as a distance function between different samples, where θ denotes the learnable parameters.

The proximity between feature segments from two distinct modalities suggests similarity in their semantic content. For instance, an API call sequence from a malware sample corresponds to a binary code segment, which in turn corresponds to a patch in the converted grayscale image. We posit that if the model can identify that the features of the API call sequence and the patch in the grayscale image are a match, then it is possible to amalgamate the information from both modalities, thereby obtaining a deeper representation of features.

To learn such semantic representations and to ensure symmetry in these relationships, such that $e_{ij} = e_{ji}$, we employ a multilayer perceptron (MLP) structured after the absolute difference between two vector nodes, as proposed by the referenced literature:

$$f_\theta(x, y) = MLP_\theta(abs(x - y)) \tag{9}$$

where MLP represents a multilayer perceptron architecture, adopting a structure similar to[7]. Here, "abs" denotes the absolute value function.

Upon establishing pairwise distances between segments, we can construct the adjacency matrix A, represented mathematically as:

$$A = \{e_{ij} | 1 \leq i, j \leq 2N\} \tag{10}$$

Following the construction of the adjacency matrix, message passing operations as described in Equation (6) facilitate the infusion of API features into image feature segments and vice versa. Subsequently, each segment is concatenated, resulting in a fused multimodal feature set $V' = concat[V_1', ...] \in \mathbb{R}^{1 \times 2D}$, which incorporates the combined features of both modalities.

This design is inspired by the observation that malware behaviors manifest across both static code structures and dynamic API invocations. By segmenting features and constructing a bipartite graph between modality segments, our model can capture cross-modal correlations that are missed by global feature fusion approaches. For instance, a specific API call pattern related to registry manipulation may align with image regions corresponding to code sections that perform similar operations. The message passing mechanism allows these complementary signals to reinforce each other, leading to more discriminative features for few-shot classification.

4.5. Prototypical network

We then use the fused multimodal features for few-shot malware classification. We have opted to utilize foundational and typical prototypical networks[8]. As shown in Figure 2, after obtaining the fused feature V', we generate the prototype for each class as:

$$p_k = \frac{1}{|S_k|} \sum_{V_i^S \in S_k} V_i^S \tag{11}$$

Here, S_k denotes the set of samples belonging to class k, and V_i^S represents the features of the i-th sample within the support set.

Subsequently, we compute the probability corresponding to each class for a given query vector V_i^Q as follows:

$$\text{prob}(y = k | V_i^Q) = \frac{\exp(-d(V_i^Q, p_k))}{\sum_{k'} \exp(-d(V_i^Q, p_{k'}))} \tag{12}$$

where d is the distance function, for which we have selected the Euclidean distance. The class receiving the highest probability will be designated as the final malware classification category.

The training model's loss function is composed of three parts: besides the cross-entropy loss of the predicted probabilities in the prototypical network using multimodal features L_M, it also includes the losses L_I and L_A for the predicted probabilities in the prototypical network using unimodal features, respectively (as shown in Figure 2). The composite loss function is expressed as:

$$L = \lambda_0 L_M + \lambda_1 L_I + \lambda_2 L_A \tag{13}$$

The calculations for L_M, L_I and L_A are analogous; here, L_M is provided as an example:

$$L_M = -\sum_{k=1}^{C} y_k \log(\text{prob}(y = k|V^Q)) \tag{14}$$

5. EXPERIMENTS

We mainly evaluate our method on two datasets: VirusShare-M and LargePE-M. A detailed description of the datasets is given in Section 3.

5.1. Baseline models

The baseline models for the comparison of our methods are shown below, including methods that only use API sequences and images as well as simple multimodal feature fusion methods.

• **ProtoNet**[8] is a classic few-shot learning method that learns a prototype representation for each class. Applicable to both API and image modalities.

• **HybridAttentionNet**[56] uses instance-level and feature attention modules to better weight query-related samples and feature dimensions. Applicable to both modalities.

• **API Frequency Histogram**[57] counts API invocation frequencies as feature vectors for malware samples.

• **Pixel k-NN** classifies malware based on image pixels using the k-nearest neighbors algorithm.

• **Early fusion** concatenates features from different modalities before feeding into prototypical network.

• **Late fusion** averages the final logits from all modalities.

• **Butterfly Vision Transformer (BViT)**[58]**:** A ViT architecture for visualization-based malware classification that captures both local and global spatial representations via butterfly construction enabling parallel patch processing. We evaluate BViT/B16 (174.2 M params) and BViT/L16 (425.6 M params).

• **DREAM**[38]**:** A concept drift detection system that embeds malware behavioral concepts in a contrastive autoencoder latent space, enabling model-sensitive drift detection without training data access during testing. We adapted DREAM to work with our grayscale images.

• **MalFSCIL**[39]**:** A few-shot class-incremental learning framework using VAE for feature enhancement and graph attention networks for dynamic prototype-based boundary delineation to address catastrophic forgetting. It converts binaries to grayscale images using a ResNet18 backbone.

Table 3. The accuracy and 95% confidence interval of our proposed method and baselines on the two datasets

Dataset		Virusshare-M				LargePE-M	
Model	Modality	5-shot		10-shot		5-shot	
		5-way	10-way	5-way	10-way	5-way	10-way
Pixel kNN	I	72.15 ± 1.12	63.30 ± 1.22	74.31 ± 1.35	66.73 ± 0.97	72.94 ± 1.06	66.37 ± 1.21
HybridAttentionNet	I	81.11 ± 0.25	74.11 ± 0.25	82.57 ± 0.31	74.51 ± 0.29	83.28 ± 0.24	74.66 ± 0.32
ProtoNet	I	83.27 ± 0.15	75.74 ± 0.18	84.16 ± 0.19	79.80 ± 0.16	84.56 ± 0.14	75.87 ± 0.17
BViT/B16	I	88.47 ± 0.12	81.23 ± 0.15	89.31 ± 0.11	84.56 ± 0.13	88.94 ± 0.10	80.43 ± 0.14
BViT/L16	I	89.82 ± 0.10	82.67 ± 0.13	90.45 ± 0.09	85.89 ± 0.11	90.17 ± 0.09	81.76 ± 0.12
DREAM	I	90.15 ± 0.09	83.42 ± 0.12	91.23 ± 0.08	86.71 ± 0.10	91.08 ± 0.08	82.94 ± 0.11
MalFSCIL	I	90.58 ± 0.08	84.27 ± 0.11	92.65 ± 0.07	88.15 ± 0.09	91.87 ± 0.07	84.56 ± 0.10
API Frequency Hist.	A	85.22 ± 0.33	81.40 ± 0.21	87.39 ± 0.27	85.17 ± 0.24	86.21 ± 0.25	78.53 ± 0.29
HybridAttentionNet	A	84.14 ± 0.22	80.11 ± 0.37	87.77 ± 0.25	83.96 ± 0.27	85.78 ± 0.23	77.98 ± 0.31
ProtoNet	A	85.32 ± 0.14	79.70 ± 0.16	86.60 ± 0.13	82.44 ± 0.15	86.12 ± 0.13	77.11 ± 0.16
Early fusion	A + I	91.47 ± 0.10	86.70 ± 0.14	93.47 ± 0.09	88.25 ± 0.12	94.02 ± 0.07	91.76 ± 0.12
Late fusion	A + I	92.01 ± 0.10	84.22 ± 0.14	94.18 ± 0.08	87.78 ± 0.13	94.46 ± 0.06	92.28 ± 0.19
BViT/B16 + API (Late)	A + I	93.84 ± 0.09	88.91 ± 0.12	94.73 ± 0.08	89.62 ± 0.10	95.21 ± 0.07	92.85 ± 0.09
BViT/L16 + API (Late)	A + I	94.68 ± 0.08	89.85 ± 0.11	95.41 ± 0.07	90.77 ± 0.09	95.98 ± 0.06	93.42 ± 0.08
DREAM + API (Late)	A + I	94.92 ± 0.08	90.23 ± 0.10	95.87 ± 0.06	91.34 ± 0.08	96.24 ± 0.05	93.71 ± 0.07
MalFSCIL + API (Late)	A + I	95.23 ± 0.07	91.05 ± 0.09	96.42 ± 0.05	92.18 ± 0.07	96.87 ± 0.04	94.25 ± 0.06
GNN fusion (Ours)	A + I	95.73 ± 0.07	92.04 ± 0.11	97.31 ± 0.05	94.89 ± 0.09	96.59 ± 0.06	93.73 ± 0.09

I and A stand for image and API respectively. API: Application programming interface; GNN: graph neural network.

5.2. Experimental setup

Following the paradigm in most few-shot experiments, we set N = [5, 10] and K = [5, 10] (notations in Section 4.1) resulting in 4 combinations for the VirusShare-M dataset, and N = [5, 10] and K = 5 for LargePE-M due to the limited number of samples. We use Adam optimizer with a learning rate of $5e^{-5}$ for multimodal and API sequences only models and $1e^{-4}$ for images only models, weight decay is set to $1e^{-4}$. All the networks are trained for 5×10^4 epochs and are saved when achieving the lowest loss on the meta-validation set for testing. All methods are implemented in PyTorch[59] and we use a machine with an Intel(R) Core(TM) i9-10900X CPU with four NVIDIA GeForce RTX 2080Ti GPUs for all the experiments.

5.3. Experimental results

We present the main results of our method and baselines in Table 3. We train the models with three different random seeds (0, 1, and 2) and report the average as the overall performance.

We find that simple few-shot learning frameworks such as the prototypical network (ProtoNet) can provide consistent performance across different modalities, proving the effectiveness of ProtoNet as the backbone.

For unimodal models, we find that API sequences can provide more effective features compared with malware grayscale images. For example, ProtoNet applied to images can only achieve 83.27% accuracy in the 5-way 5-shot setting on VirusShare-M dataset while the same model based on API sequences can reach 85.32%. Recently proposed vision transformer-based methods, such as BViT/B16 and BViT/L16, significantly improve image-based performance, achieving 88.47% and 89.82% respectively on the same task. This confirms the advantage of transformer architectures in capturing spatial representations of malware images.

More recent advances in malware concept drift detection and few-shot class-incremental learning have demonstrated even stronger performance. DREAM[38], a concept-based drift detection system for Android malware, achieves 90.15% accuracy when adapted to our image-based few-shot setting. Its model-sensitive concept learning approach enables more effective detection of new malware families. MalFSCIL[39], a few-shot class-incremental learning framework specifically designed for malware detection, further improves image-based performance to 90.58% on VirusShare-M 5-way 5-shot. This method's decoupled training strategy and VAE-based feature enhancement effectively mitigate catastrophic forgetting, making it particularly suitable for incremental learning scenarios.

It should be noted that our evaluation is conducted under a class-disjoint meta-learning protocol, where the training, validation, and test sets contain disjoint malware families. This setup simulates the scenario where new families emerge, and the model must generalize to them with limited support samples. However, this evaluation does not fully capture real-world zero-day scenarios, which may involve significant distribution shifts, concept drift, or entirely novel attack behaviors that differ substantially from training data. Therefore, while our results demonstrate strong performance in few-shot settings, further evaluation under more challenging conditions - such as open-set recognition, domain adaptation, or temporal drift - would be valuable future work.

For multimodal baselines, we observe that even the simplest methods of multimodal integration - early fusion via concatenation and late fusion at the decision-making stage - significantly outperform models that utilize unimodal information alone. This observation substantiates the effectiveness of leveraging multimodal information for few-shot malware classification tasks. When combining BViT/L16 with API features via late fusion, accuracy further improves to 94.68%, demonstrating that even strong unimodal encoders benefit from multimodal integration. DREAM and MalFSCIL, when combined with API features via late fusion, achieve 94.92% and 95.23% accuracy respectively, approaching the performance of our proposed GNN fusion method.

Our proposed GNN fusion module achieves superior performance compared to methods that rely on simple feature fusion. Our method reaches 95.73% accuracy on VirusShare-M 5-way 5-shot, outperforming both BViT-based fusion approaches (94.68%) and the more recent DREAM (94.92%) and MalFSCIL (95.23%) fusion variants. This suggests that while advanced single-modality methods like DREAM and MalFSCIL continue to push the boundaries of image-based malware classification, our GNN fusion module's ability to model fine-grained cross-modal correlations at the segment level provides advantages over simple late fusion, even when using state-of-the-art image encoders. Notably, MalFSCIL's VAE-based feature enhancement and DREAM's concept reliability detection offer complementary strengths that could potentially be integrated with our fusion approach in future work. The subsequent section on ablation studies and analysis will discuss how the GNN fusion module operates and provide a comparative evaluation of the training costs associated with various fusion methodologies.

5.4. Analysis and ablation study

5.4.1. Effect of normalization

As indicated in Table 4, we observe that the normalization module proposed in Section 4.4.1 significantly enhances the classification accuracy of multimodal models. This improvement aligns with the original intent behind introducing the module, which is to standardize the numerical ranges of features across multiple modalities to a common scale. This concept is discussed in Section 4.4.1 and will not be elaborated further here.

Table 4. Ablation experiments on effect of normalization

Model	Image	API	Norm.	Accuracy
ProtoNet	√			83.27 ± 0.15
ProtoNet	√		√	84.64 ± 0.14
ProtoNet		√		85.32 ± 0.14
ProtoNet		√	√	87.78 ± 0.12
Early fusion	√	√		91.47 ± 0.10
Early fusion	√	√	√	94.94 ± 0.08
GNN fusion	√	√		93.57 ± 0.08
GNN fusion	√	√	√	95.73 ± 0.07

API: Application programming interface; GNN: graph neural network.

Table 5. Ablation experiments on effect of loss function

Model	Image	API	MM	Accuracy
Early fusion			√	91.47 ± 0.10
Early fusion	√	√	√	92.78 ± 0.10
GNN fusion			√	95.26 ± 0.08
GNN fusion	√	√	√	95.73 ± 0.07

API: Application programming interface; MM: multimodal; GNN: graph neural network.

Unexpectedly, the normalization module also appears to improve the performance of unimodal models. Further analysis reveals that the benefits of normalization are more pronounced for API sequence information than for images. We hypothesize that normalization promotes a more uniform distribution across each dimension of the features, as demonstrated in Figure 4. Within a single modality, there are discrepancies in feature values. The role of normalization in this context is analogous to that of layer normalization in transformers; it diminishes the magnitude disparities across samples while preserving the relative differences among features. This is particularly justifiable given that API sequence features of malicious software represent temporal variations of N-gram items, where the relational dynamics within the features are inherently tight. Moreover, we propose that the normalization module can be generalized to any small-sample task, serving as a universal technique to enhance classification outcomes.

5.4.2. Effect of loss function

We examine the contributions of the loss function described in Section 4.5. As demonstrated in Table 5, incorporating the individual predictive values of each modality into the loss function substantially enhances the performance of the model. Beyond our proposed GNN fusion approach, the application of our designed loss function also improves accuracy by approximately 1% for the simple concatenation fusion model in 5-way-5-shot task on the VirusShare-M dataset.

5.4.3. Computing cost

In previous sections, we mentioned that our GNN-based feature fusion module is lightweight. Here we will discuss the specific computing cost of this module. To provide a more direct comparison, we have calculated the FLOPs and number of parameters of the GNN-based feature fusion module and compared it to the feature extraction modules described in Figure 3. Additionally, to evaluate suitability for real-world deployment scenarios, we measured inference latency and memory usage on a CPU-based system (Intel Core i7-10750H, 16GB RAM) without GPU acceleration. All measurements are averaged over 1,000 runs. The hyperparameter settings for the GNN fusion module are consistent with those discussed in Section 5.2.

Table 6. Computing cost and deployment efficiency of each module in our framework

Module	MACs	Params	Inference time (ms)	Memory (MB)
CNN4	10.09 GMac	95.49 k	12.3	89.4
Simple transformer	43.41 GMac	1.39 M	8.7	112.6
GNN fusion	189.86 MMac	79.11 k	3.2	55.8

MACs: Multiply-accumulate operations; CNN: convolutional neural network; GNN: graph neural network.

As demonstrated in Table 6, using the ptflops library (https://pypi.org/project/ptflops/), we quantified the computational metrics. The GNN fusion module only requires 189.86 MMac of multiply-accumulate operations (MACs) and has a remarkably low parameter count of 79.11 k. When converted to percentages, the GNN fusion module increases the model's MACs by just 0.35% and its parameter volume by 5%, yet achieves effective feature fusion. In terms of deployment efficiency, the GNN fusion module adds only 3.2 ms of latency and 55.8 MB of memory overhead, which is acceptable for most real-time applications. This underscores the module's utility in scenarios where computational resources are limited, and maintaining model simplicity without sacrificing performance is crucial. Such attributes make the GNN fusion module particularly suitable for real-time applications and devices with constrained computational capabilities.

5.5. Interpretability analysis

To understand how our GNN fusion module leverages cross-modal correlations, we analyze the attention weights learned during message passing for representative malware samples.

Case Study: Zbot Family. For a Zbot (Zeus) malware sample, the API sequence contains N-gram segments related to registry operations (e.g., "RegOpenKeyEx"-"RegSetValueEx") and file writes (e.g., "CreateFile"-"WriteFile"). The GNN assigns high attention weights between these API segments and specific image patches from the binary's code and data sections, where corresponding registry key strings and configuration data are stored. This alignment indicates that the GNN learns to associate behavioral patterns (API calls) with structural evidence (binary content) serving the same malicious purpose.

Quantitative Analysis. Across all correctly classified samples in the VirusShare-M test set, registry-related API segments show an average attention weight of 0.342 to code section image patches, compared to 0.156 to other sections (119% higher). For correctly classified samples of the visually similar Swizzor.gen!E and Swizzor.gen!I families, the top-3 attention weights account for 67.3% of total attention, whereas for misclassified samples they account for only 41.8%. Additionally, attention patterns remain stable across different few-shot tasks, with a standard deviation of only 0.042, indicating transferable cross-modal relationships rather than sample-specific overfitting.

Contrast with Unimodal Models. An API-only model detects registry manipulation but lacks structural context; an image-only model sees suspicious strings but cannot verify their runtime usage. The GNN fusion bridges this gap by explicitly modeling cross-modal relationships, enabling more robust classification.

Differentiating Similar Families. For visually similar families like Swizzor.gen!E and Swizzor.gen!I, correctly classified samples show GNN attention focused on discriminative segment pairs (e.g., unique network API patterns aligned with corresponding API hashes in the binary), while misclassified samples exhibit diffuse attention across many segments.

Implications for Few-Shot Learning. The GNN's ability to capture consistent cross-modal correlations provides additional supervisory signal beyond limited labeled examples, improving generalization in few-shot scenarios.

In summary, the GNN fusion module enhances interpretability by revealing how behavioral and structural evidence complement each other, supporting both model understanding and analyst trust.

6. CONCLUSIONS

In this paper, we proposed leveraging multimodal information from malware, specifically API invocation sequences and converted grayscale images, for few-shot malware classification. Additionally, we contributed two datasets to facilitate this research. Furthermore, we introduced a lightweight, GNN-based feature fusion module that constructs a graph network among features from multiple modalities, achieving feature integration through message passing. We conducted evaluations on the two datasets we constructed, and the experimental results demonstrate that utilizing multimodal features significantly enhances the accuracy of few-shot malware classification. The results also indicate that our proposed GNN fusion module effectively integrates features. Additional ablation studies discuss the impact of each component of our module and also show that our feature fusion module achieves effective integration with minimal computational overhead.

DECLARATIONS

Authors' contributions

Conceptualization: Ren, Y.; Wang, J.
Data curation: Wang, J.; Liu, Z.
Formal analysis: Ren, Y.; Liu, Z.
Funding acquisition: Wang, P.
Investigation: Ren, Y.; Wang, P.
Methodology: Ren, Y.
Project administration: Wang, P.
Resources: Wang, P.
Software: Ren, Y.; Wang, J.
Supervision: Wang, P.; Liu, Z.
Validation: Ren, Y.
Visualization: Ren, Y.; Liu, Z.
Writing - original draft: Ren, Y.; Liu, Z.; Wang, J.
Writing - review and editing: Wang, J.; Wang, P.

Availability of data and materials

The original contributions presented in the study are included in the article/Supplementary Materials; further inquiries can be directed to the corresponding author(s).

AI and AI-assisted tools statement

Not applicable.

Financial support and sponsorship

None.

Conflicts of interest

All authors declared that there are no conflicts of interest.

Ethical approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Supplementary Materials

Supplementary Materials

REFERENCES

1. Stuttard, D.; Pinto, M. The web application hacker's handbook: finding and exploiting security flaws. John Wiley & Sons; 2011. https://books.google.com/books?id=NSBHAAAAQBAJ. (accessed 2026-06-08).
2. Bhodia, N.; Prajapati, P.; Di Troia, F.; Stamp, M. Transfer learning for image-based malware classification. *arXiv* **2019**, arXiv:1903.11551. Available online: https://doi.org/10.48550/arXiv.1903.11551. (accessed 2026-06-08).
3. Vu, D. L.; Nguyen, T. K.; Nguyen, T. V.; Nguyen, T. N.; Massacci, F.; Phung, P. H. A convolutional transformation network for malware classification. In *2019 6th NAFOSTED conference on information and computer science (NICS)*. Hanoi, Vietnam, Dec 12-13, 2019. IEEE; 2019. pp. 234-39. DOI
4. Vasan, D.; Alazab, M.; Wassan, S.; Safaei, B.; Zheng, Q. Image-based malware classification using ensemble of CNN architectures (IMCEC). *Comput. Secur.* **2020**, *92*, 101748. DOI
5. Kolosnjaji, B.; Zarras, A.; Webster, G.; Eckert, C. Deep learning for classification of malware system call sequences. In *AI 2016: Advances in Artificial Intelligence: 29th Australasian Joint Conference*. Hobart, Australia, December 5-8, 2016. Springer; 2016. pp. 137-49. DOI
6. Wang, P.; Tang, Z.; Wang, J. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. *Comput. Secur.* **2021**, *106*, 102273. DOI
7. Wang, J.; Lin, T.; Wu, H.; Wang, P. AGProto: adaptive graph ProtoNet towards sample adaption for few-shot malware classification. *Electronics* **2024**, *13*, 935. DOI
8. Snell, J.; Swersky, K.; Zemel, R. Prototypical networks for few-shot learning. *arXiv* **2017**, arXiv:1703.05175. Available online: https://doi.org/10.48550/arXiv.1703.05175. (accessed 2026-06-08).
9. Vinayakumar, R.; Alazab, M.; Soman, K.; Poornachandran, P.; Venkatraman, S. Robust intelligent malware detection using deep learning. *IEEE. Access.* **2019**, *7*, 46717-38. DOI
10. Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; Nicholas, C. Malware detection by eating a whole EXE. *arXiv* **2017**, arXiv:1710.09435. Available online: https://doi.org/10.48550/arXiv.1710.09435. (accessed 2026-06-08).
11. Gibert, D.; Mateu, C.; Planes, J. A hierarchical convolutional neural network for malware classification. In *2019 International Joint Conference on Neural Networks (IJCNN)*. Budapeest, Hungary, Jul 14-19, 2019. IEEE; 2019. pp. 1-8. DOI
12. Raff, E.; Zak, R.; Cox, R.; et al. An investigation of byte n-gram features for malware classification. *J. Comput. Virol. Hack. Tech.* **2018**, *14*, 1-20. DOI
13. Cui, Z.; Xue, F.; Cai, X.; Cao, Y.; Wang, Gg.; et al. Detection of malicious code variants based on deep learning. *IEEE. Trans. Ind. Inform.* **2018**, *14*, 3187-96. DOI
14. Jiang, Y.; Li, S.; Wu, Y.; Zou, F. A novel image-based malware classification model using deep learning. In *International Conference on Neural Information Processing*. Springer; 2019. pp. 150-61. DOI
15. Yuan, B.; Wang, J.; Liu, D.; Guo, W.; Wu, P.; Bao, X. Byte-level malware classification based on markov images and deep learning. *Comput. Secur.* **2020**, *92*, 101740. DOI
16. Sharma, O.; Sharma, A.; Kalia, A. Windows and IoT malware visualization and classification with deep CNN and Xception CNN using Markov images. *J. Intell. Inf. Syst.* **2023**, *60*, 349-75. DOI
17. Azmoodeh, A.; Dehghantanha, A.; Choo, K. K. R. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE. Trans. Sustain. Comput.* **2019**, *4*, 88-95. DOI
18. Zhu, H.; Gu, W.; Wang, L.; Xu, Z.; Sheng, V. S. Android malware detection based on multi-head squeeze-and-excitation residual network. *Expert. Syst. Appl.* **2023**, *212*, 118705. DOI
19. Sasidharan, S. K.; Thomas, C. ProDroid - an Android malware detection framework based on profile hidden Markov model. *Pervasive. Mob. Comput.* **2021**, *72*, 101336. DOI
20. Lee, Y. T.; Ban, T.; Wan, T. L.; et al. Cross platform IoT-malware family classification based on printable strings. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. Guangzhou, China, Dec 29 2020 - Jan 01 2021. IEEE; 2020. pp. 775-84. DOI
21. Han, W.; Xue, J.; Wang, Y.; Huang, L.; Kong, Z.; Mao, L. MalDAE: detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *Comput. Secur.* **2019**, *83*, 208-33. DOI
22. Xu, J.; Li, Y.; Deng, R. H.; Xu, K. SDAC: a slow-aging solution for android malware detection using semantic distance based API clustering. *IEEE. Trans. Dependable. Secure. Comput.* **2022**, *19*, 1149-63. DOI

23. Yan, S.; Ren, J.; Wang, W.; Sun, L.; Zhang, W.; Yu, Q. A survey of adversarial attack and defense methods for malware classification in cyber security. *IEEE. Commun. Surv. Tutor.* **2023**, *25*, 467-96. DOI
24. Han, X.; Yu, X.; Pasquier, T.; et al. SIGL: Securing software installations through deep graph learning. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association; 2021. pp. 2345-62. https://www.usenix.org/conference/usenixsecurity21/presentation/han-xueyuan. (accessed 2026-06-08).
25. Amer, E.; Zelinka, I.; El-Sappagh, S. A multi-perspective malware detection approach through behavioral fusion of API call sequence. *Comput. Secur.* **2021**, *110*, 102449. DOI
26. Lin, K.; Xu, X.; Xiao, F. MFFusion: a multi-level features fusion model for malicious traffic detection based on deep learning. *Comput. Netw.* **2022**, *202*, 108658. DOI
27. Almashhadani, A. O.; Carlin, D.; Kaiiali, M.; Sezer, S. MFMCNS: a multi-feature and multi-classifier network-based system for ransomworm detection. *Comput. Secur.* **2022**, *121*, 102860. DOI
28. Wang, Q.; Hassan, W. U.; Li, D.; et al. You are what you do: hunting stealthy malware via data provenance analysis. In *Network and Distributed Systems Security (NDSS) Symposium 2020*. San Diego, USA, Feb 23-26, 2020. https://www.ndss-symposium.org/wp-content/uploads/2020/02/24167.pdf. (accessed 2026-06-08).
29. Damodaran, A.; Di Troia, F.; Visaggio, C. A.; Austin, T. H.; Stamp, M. A comparison of static, dynamic, and hybrid analysis for malware detection. *J. Comput. Virol. Hack. Tech.* **2017**, *13*, 1-12. DOI
30. Gopinath, M.; Sethuraman, S. C. A comprehensive survey on deep learning based malware detection techniques. *Comput. Sci. Rev.* **2023**, *47*, 100529. DOI
31. Huang, X.; Ma, L.; Yang, W.; Zhong, Y. A method for windows malware detection based on deep learning. *J. Sign. Process. Syst.* **2021**, *93*, 265-73. DOI
32. Yoo, S.; Kim, S.; Kim, S.; Kang, B. B. AI-HydRa: advanced hybrid approach using random forest and deep learning for malware classification. *Inform. Sci.* **2021**, *546*, 420-35. DOI
33. Nguyen, T. N.; Ngo, Q. D.; Nguyen, H. T.; Nguyen, G. L. An advanced computing approach for IoT-botnet detection in industrial Internet of Things. *IEEE. Trans. Ind. Inform.* **2022**, *18*, 8298-306. DOI
34. O'Shaughnessy, S.; Sheridan, S. Image-based malware classification hybrid framework based on space-filling curves. *Comput. Secur.* **2022**, *116*, 102660. DOI
35. Kim, T.; Kang, B.; Rho, M.; Sezer, S.; Im, E. G. A multimodal deep learning method for android malware detection using various features. *IEEE. Trans. Inf. Forensics. Secur.* **2019**, *14*, 773-88. DOI
36. Gibert, D.; Mateu, C.; Planes, J. HYDRA: a multimodal deep learning framework for malware classification. *Comput. Secur.* **2020**, *95*, 101873. DOI
37. Dib, M.; Torabi, S.; Bou-Harb, E.; Assi, C. A multi-dimensional deep learning framework for iot malware classification and family attribution. *IEEE. Trans. Netw. Serv. Manag.* **2021**, *18*, 1165-77. DOI
38. He, Y.; Lei, J.; Qin, Z.; Ren, K.; Chen, C. Combating concept drift with explanatory detection and adaptation for Android malware classification. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*. New York, USA. Association for Computing Machinery; 2025. pp. 978-92. DOI
39. Chai, Y.; Chen, X.; Qiu, J.; et al. MalFSCIL: a few-shot class-incremental learning approach for malware detection. *IEEE. Trans. Inf. Forensics. Secur.* **2025**, *20*, 2999-3014. DOI
40. Miller, S. J.; Howard, J.; Adams, P.; Schwan, M.; Slater, R. Multi-modal classification using images and text. *SMU Data Sci. Rev.* **2020**, *3*, 6. https://scholar.smu.edu/cgi/viewcontent.cgi?article=1165&context=datasciencereview. (accessed 2026-06-08).
41. Audebert, N.; Herold, C.; Slimani, K.; Vidal, C. Multimodal deep networks for text and image-based document classification. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019*. Würzburg, Germany, Sep 16-20, 2019. Springer; 2020. pp. 427-43. DOI
42. Antol, S.; Agrawal, A.; Lu, J.; et al. VQA: visual question answering. 2015. https://openaccess.thecvf.com/content_iccv_2015/papers/Antol_VQA_Visual_Question_ICCV_2015_paper.pdf. (accessed 2026-06-08).
43. Xu, K.; Ba, J.; Kiros, R.; et al. Show, attend and tell: neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, PMLR*. 2015. pp. 2048-57. https://proceedings.mlr.press/v37/xuc15.html. (accessed 2026-06-08).
44. Anderson, P.; He, X.; Buehler, C.; et al. Bottom-up and top-down attention for image captioning and visual question answering. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018. pp. 6077-86. DOI
45. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556. Available online: https://doi.org/10.48550/arXiv.1409.1556. (accessed 2026-06-08).
46. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural. Comput.* **1997**, *9*, 1735-80. DOI PubMed

47. Yang, Z.; He, X.; Gao, J.; Deng, L.; Smola, A. Stacked attention networks for image question answering. *arXiv* **2015**, arXiv:1511.02274. Available online: https://doi.org/10.48550/arXiv.1511.02274. (accessed 2026-06-08).
48. Kim, J. H.; Jun, J.; Zhang, B. T. Bilinear attention networks. *arXiv* **2018**, arXiv:1805.07932. Available online: https://doi.org/10.48550/arXiv.1805.07932. (accessed 2026-06-08).
49. Vaswani, A.; Shazeer, N.; Parmar, N.; et al. Attention is all you need. *arXiv* **2017**, arXiv:1706.03762. Available online: https://doi.org/1.48550/arXiv.1706.03762. (accessed 2026-06-08).
50. Devlin, J.; Chang, M. W.; Lee, K.; Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805. Available online: https://doi.org/10.48550/arXiv.1810.04805. (accessed 2026-06-08).
51. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B. S. Malware images: visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*. Association for Computing Machinery; 2011. p. 1-7. DOI
52. Seok, S.; Kim, H. Visualized malware classification based-on convolutional neural network. *J. Korea. Inst. Inf. Secur. Cryptol.* **2016**, *26*, 197-208. https://www.researchgate.net/publication/301236691_Visualized_Malware_Classification_Based-on_Convolutional_Neural_Network. (accessed 2026-06-08).
53. Hospedales, T.; Antoniou, A.; Micaelli, P.; Storkey, A. Meta-learning in neural networks: a survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *44*, 5149-69. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9428530. (accessed 2026-06-08).
54. Song, Y.; Wang, T.; Mondal, S. K.; Sahoo, J. P. A comprehensive survey of few-shot learning: evolution, applications, challenges, and opportunities. *arXiv* **2022**, arXiv:2205.06743. Available online: https://doi.org/10.48550/arXiv.2205.06743. (accessed 2026-06-08).
55. Vinyals, O.; Blundell, C.; Lillicrap, T.; Kavukcuoglu, K.; Wierstra, D. Matching networks for one shot learning. *arXiv* **2016**, arXiv:1606.04080. Available online: https://doi.org/10.48550/arXiv.1606.04080. (accessed 2026-06-08).
56. Gao, T.; Han, X.; Liu, Z.; Sun, M. Hybrid attention-based prototypical networks for noisy few-shot relation classification. *Proc. AAAI. Conf. Artif. Intell.* **2019**, *33*, 6407-14. DOI
57. Ndibanje, B.; Kim, K. H.; Kang, Y. J.; Kim, H. H.; Kim, T. Y.; Lee, H. J. Cross-method-based analysis and classification of malicious behavior by API calls extraction. *Appl. Sci.* **2019**, *9*, 239. DOI
58. Belal, M. M.; Sundaram, D. M. Global-local attention-based butterfly vision transformer for visualization-based malware classification. *IEEE. Access.* **2023**, *11*, 69337-55. DOI
59. Paszke, A.; Gross, S.; Chintala, S.; et al. Automatic differentiation in PyTorch. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*. Long Beach, USA. 2017. https://openreview.net/pdf?id=BJJsrmfCZ. (accessed 2026-06-08).