This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



# SoK: Leveraging Transformers for Malware Analysis

Pradip Kunwar[1], Kshitiz Aryal[1], Maanak Gupta[1], *Senior Member, IEEE*, Mahmoud Abdelsalam[2], Elisa Bertino[3], *Fellow, IEEE*
[1] *Tennessee Tech University,* [2] *North Carolina A&T State University,* [3] *Purdue University*

**Abstract**—The introduction of transformers has been an important breakthrough for AI research and application, as transformers are the foundation behind Generative AI. Transformers are promising in cybersecurity, especially malware analysis. The reason is the flexibility of the transformer models in handling long sequential features and understanding contextual relationships. However, as the use of transformers for malware analysis is still in the infancy stage, it is critical to evaluate, systematize, and contextualize existing literature to foster future research. This Systematization of Knowledge (SoK) paper aims to provide a comprehensive analysis of transformer-based approaches designed for malware analysis. Based on our systematic analysis of existing knowledge, we structure and propose taxonomies based on: (a) how different transformers are adapted, organized, and modified across various use cases; and (b) how diverse feature types and their representation capabilities are reflected. We also provide an inventory of datasets used to explore multiple research avenues in the use of transformers for malware analysis and discuss open challenges with future research directions. We believe that this SoK paper will assist the research community in gaining detailed insights from existing work and will serve as a foundational resource for implementing novel research using transformers for malware analysis.

**Index Terms**—Malware Analysis, Transformers, Pre-trained Transformers, Feature Representation, Cybersecurity

## 1 Introduction

The growing dependency on digital technology and connectivity has led to an unprecedented surge in the generation and distribution of malicious software, also referred to as malware [1]. Malware poses significant threats to cybersecurity, targeting individuals, organizations, and critical infrastructures in various forms, including viruses, worms, ransomware, etc. As malware evolves in complexity [2], [3], improving classification, detection, and mitigation methods is critical. To address this growing challenge in malware analysis, Artificial Intelligence (AI) and Machine learning (ML) based solutions, including Deep Learning (DL) [4], Computer Vision [5], [6], and Natural Language Processing (NLP) [7], [8], have been recently proposed. These techniques help automate feature extraction and improve static and dynamic analysis for detection/classification tasks against sophisticated and evasive malware. Among various approaches, transformer-based models are recently garnering significant attention. The transformer model introduced by Vaswani et al. [9] has revolutionized a broad range of tasks [10] in various domains including NLP [11], [12] for text classification [13], machine translation, question answering [14] and text generation [15]; computer vision for image classification [16]–[18], object detection [19], image and video generation [20], [21]; speech recognition [22], and etc. with significant performance improvements.

In malware analysis, transformers have begun to emerge as a versatile and powerful tool [23], [24]. Transformers excel in capturing intricate patterns, such as spatial, temporal, structural, etc., across high-dimensional data making them well-suited for complex static and dynamic malware analysis. These models analyze extensive datasets from raw binary files, disassembled codes, graphs, images, and many other feature representations of malware to uncover malicious behavior and identify emerging threats. Furthermore, their application spans various malware analysis sub-domains including detection [24]–[28] and classification [29], [30], binary code similarity detection [31], evasion techniques [32], as well as explanation and interpretation [23], [33], underlining their capability to understand malicious behavior accurately.

Although transformers were first introduced in 2017, their integration into malware analysis is relatively at an initial stage, presenting opportunities for future research and advancements. Therefore, it is critical to organize the existing knowledge based on its application, novel research findings, empirical studies, and analysis, along with open challenges to understand and foster future research leveraging transformers in malware analysis. In this paper, we explore the evolution of transformers, their impact on malware analysis, and highlight various approaches and challenges encountered by the research community. Through an extensive literature review, we examine and systematize the knowledge based on the use cases of transformers, their types, modeling aspects, representation techniques, and different features employed in adapting transformer architectures to address challenges in malware analysis. Further, to solve the issue of the lack of a comprehensive dataset repository, we compile an inventory of features, representation techniques, and datasets to assist the community.

**Methodology and Literature Search**: We collected available literature from several sources that cover a broad range

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



of research using transformer or its modified version for malware analysis tasks. Along with relevant conference venues and journals, we carried out searches on platforms such as Google Scholar and Research Rabbit using keywords like *transformers in malware analysis*, *malware transformer*, *transformer survey*, *transformer malware survey*, *sok transformer malware*, *attention malware*, etc. We collected 43 papers covering the period from 2017 to 2023, focusing on the application of transformers for malware analysis. From the collected articles, we extracted 30 different categories of information, covering objectives, motivations, end goals, domains, features, datasets, representation techniques, modeling, transformer type, need for transformer, weakness, challenges, gaps, future scopes, etc. We assessed the extracted information multiple times to critically analyze and structure the knowledge.

We synthesized and structured the available knowledge based on five major aspects by finding the answers to the questions listed below:

- **Purposes of use for transformers**: Why are the transformers used in malware analysis, and what are the similarities in those use cases?
- **End goals**: Besides the purposes for using transformers, what are the different objectives, such as malware detection, malware classification, detection evasion, etc.?
- **Type of transformer**: Which transformers architectures are used to analyze malware? Are there custom enhancements or modifications performed in the existing variations of transformer architectures?
- **Features**: What are the different features, representing malware behaviors, used to accomplish the end goals that align with the working mechanism of transformers?
- **Datasets**: What are the different datasets used in the malware analysis community that provide the features to model the representation using the transformers?

The remainder of this paper is organized as follows. Section 2 introduces basic concepts about malware analysis, and the transformer architecture and its evolution. Section 3 presents the systematization of transformers for malware analysis followed by Section 4 discussing the limitations, open challenges, and future directions. Section 5 summarizes and concludes the paper. Appendix Section 6 presents in-depth technical discussion on specific transformer architectures and their modifications, a comparative analysis of the performance of different transformer models across various malware tasks, and practical insights on the implementation of transformer models in real-world systems.

## 2 Background

### 2.1 Malware Analysis

In general, malware analysis techniques examine malicious software and unwanted code in computer systems or networks. This analysis helps to understand behaviors, intents, functions, and impacts of malicious code to assist in anticipating future activities and mitigating attacks. Traditionally, malware analysis is divided into three approaches: static, dynamic, and hybrid. Static analysis is performed without actually executing a program, dynamic analysis requires the program's execution in an isolated virtual environment (i.e., a sandbox), and the hybrid approach combines both approaches.

Static analysis is based on signature-based approach [34] and offline code examination such as memory corruption flaws [35]. However, maintaining the signatures database is impractical because of exponentially increasing malware volumes and the inability to recognize zero-day malware. In addition, static techniques fail to detect evolving malware that integrates evasive techniques such as obfuscation and polymorphism. To mitigate such limitations, dynamic analysis is used, which examines malware interactions with the operating system, memory, network, and applications, thus tracking its actions, network communications, and modifications to system files and the registry [36]. To leverage both benefits, the hybrid approach combines static and dynamic analysis to gather critical features for performing different malware analysis tasks. However, applying conventional methods is less effective in detecting evolving and sophisticated zero-day malware [37]. In addition, advanced malware typically can detect the presence of a sandbox environment and, in turn, can evade behavior-based detection approaches by ceasing or altering their true malicious behavior. A significant limitation of conventional approaches is their inability to generalize from the known patterns to detect new, sophisticated threats that have not been previously encountered, which results in a higher rate of false negatives [38].

To address these issues, the application of ML in malware analysis has increased significantly to enhance real-time detection and classification with high accuracy and low false positives [23], [25], [39], [40]. ML and in particular deep learning (DL) models can automate the detection of new and evolving malware variants by learning and distinguishing complex patterns from large datasets of benign and malicious samples [41]. These practices enhance the detection of patterns, anomalies, behaviors, and intents indicative of malware, improving the speed and accuracy of detection and response. With the ML advancements, adopting such models has offered significant possibilities for new approaches and techniques to strengthen malware analysis techniques. Among these advancements, the transformer architecture stands out for its ability to handle sequential data, making it particularly suitable for analyzing malware's complex and polymorphic nature.

### 2.2 Transformer

In this section we discuss the transformer architecture, tracing its development and historical knowledge of its evolution over time, and present how the development progressed to solve contemporary issues.

#### 2.2.1 *Evolution of Transformer Architecture*

The development of the transformer model showcases the decades of progress in ML. As shown in Fig. 1, it can all be traced back to the introduction of classical ML algorithms like Logistic Regression, Decision Trees, and SVMs [42], which laid the groundwork for understanding structured data. However, these methods struggled with representing sequential and temporal data, which were critical for machine translation tasks—the very solutions the research community was eagerly seeking. As neural networks,

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



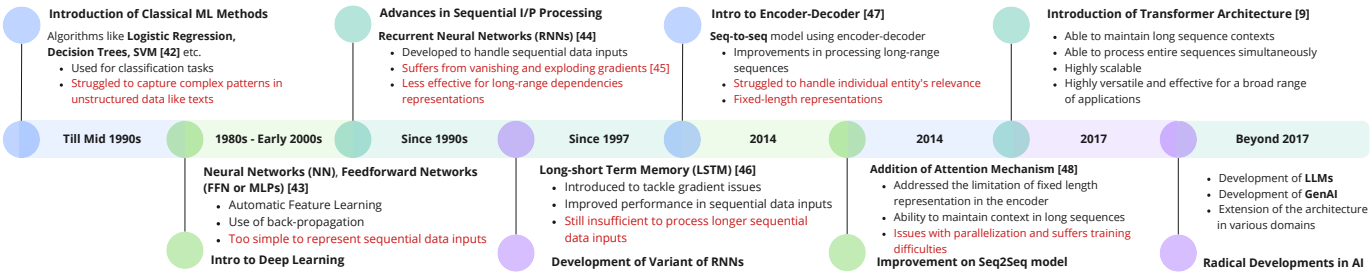


Fig. 1. Evolution of the Transformer Architecture: A Milestone in Advancing Machine Translation Task *(Note: Texts in red highlight the shortcomings)*

feed-forward networks, and Multi-Layer Perceptron (MLP) equipped with backpropagation [43] were introduced, these DL methods could learn complex and nonlinear relationships. But, the models were too simple to represent contextual semantics. Thus, the Recurrent Neural Networks (RNNs) [44] were developed to introduce memory in the network but faced significant challenges in learning long-term dependencies due to vanishing and exploding gradient issues [45]. The introduction of Long Short-Term Memory (LSTM) partially addressed this issue [46] with better memory mechanisms. Also, the encoder-decoder architecture [47] and subsequent integration of the attention mechanism in the seq2seq model [48] further enhanced the network's capabilities to improve attention to the relevant parts of the input sequence. Later, in 2017, Vaswani et al. [9] proposed the revolutionary transformer architecture, which offered abilities like scalability, versatility, unprecedented parallelization, and efficiency with performance improvements.

Fig. 1 illustrates the evolution of transformer architecture through a series of enhancements to existing machine translation solutions. However, this groundbreaking development not only revolutionized machine translation tasks but also extended its application to a wide range of sequential tasks across various fields, including NLP [11], [12], computer vision [17]–[19], and cybersecurity [23], [24]. While the timeline depicted in the figure extends only until 2017, it is important to note that there have been significant developments since then. In the fields of Generative AI and Large Language Models, the expanded applications of transformers, groundbreaking innovations, such as GPT-4 Chatbot [15], Sora video generator model [21], DALL-E image generator [20] and beyond, highlight their continued impact and versatility in shaping AI.

### 2.2.2 *Transformer Architecture*

A standard transformer architecture, also known as 'Vanilla transformer', consists of two sub-networks called encoder and decoder, as shown in Fig. 2. In general, the encoder network maps the input sequence into contextual representations which are further processed by the decoder network to generate an output sequence one at a time in an autoregressive manner i.e. the current value in a sequence is processed as a function of its all previous values.

***Input Sequence:*** For a machine-translation task, the encoder's input sequence is the language to be translated and the first input token for the decoder is the null character, which starts the translation of the language. Once the first token is generated, the output token is shifted right and amended to the existing input sequence and sent to the decoder in an autoregressive manner. These inputs are then embedded (as shown in Fig. 2) with dimension $d$ before being passed to the positional encoding layer. With these different inputs for the two sub-networks, as a whole, the architecture produces the predictions and continues learning through training.

***Positional Encoding:*** It includes the sequence order of the input in the embedding values by adding positional values. The positional encoding values also share the same dimension $d$ as the input embedding. The encodings are added using sine and cosine functions of different frequencies. The equations 1 and 2 show one of the many approaches to embed positional encoding, as mentioned by Vaswani et al. [9].

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d}) \quad (1)$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d}) \quad (2)$$

where *pos* is the position of the element in the sequence and $i$ is the *th* dimension.

### 2.2.3 *Encoder Network*

The encoder network consists of two sublayers. The first is the Self-Attention layer where the input dimensions are divided into $N$ number of identical sublayers, thus assembled to create a Multi-head Self-Attention mechanism described below. The second sublayer is a position-wise fully connected feed-forward network.

***Self-Attention Mechanism:*** Attention values are the weights of each input entity concerning other input entities. They are quantified as the percentage of attention provided while encoding all the input sequences. The attention function is the mapping of *Key(K), Query(Q), and Value(V)* where $K$, $Q$, and $V$ originate from the identical input sequences after positional encoding as shown in Fig. 2. The dimensions of both $K$ and $Q$ are $d_k$, and the dimension of $V$ is $d_v$. The setup allows each position in the input sequence to attend all positions in the same sequence to dynamically weigh and integrate information across the entire sequence, as shown in the steps below [18].

- *Step 1:* Calculate the similarity between Query and Key vectors using the dot product $Q.K^T$; $K^T$ *here is the transposition of the matrix K.*
- *Step 2:* Normalize the score to maintain the stability in the gradient by dividing the similarity score as $\frac{Q.K^T}{\sqrt{d_k}}$
- *Step 3:* Translate the scores into probabilities using the softmax as: $softmax(\frac{Q.K^T}{\sqrt{d_k}})$
- *Step 4:* Obtain the Value matrix by multiplying $V$ with the probabilities, which gives the attention weight of each input entity as given in equation 3:

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



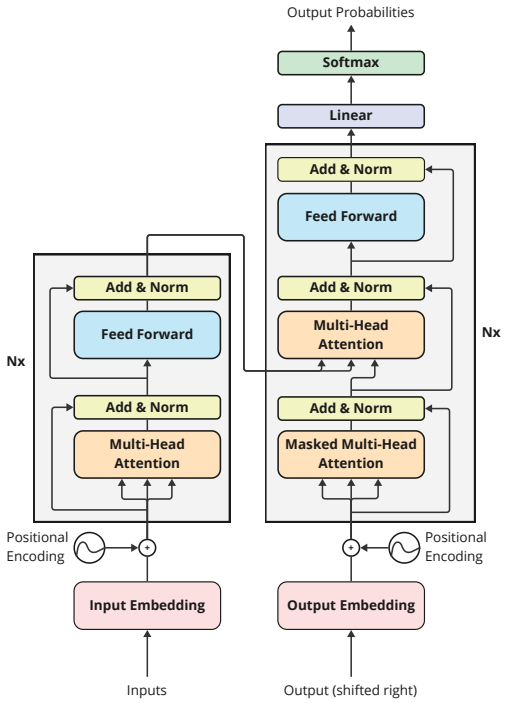


Fig. 2. Transformer architecture as proposed by Vaswani et al. [9]

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3}$$

***Multihead Attention Mechanism:*** The three identical input values as $Q$, $K$, $V$ are multiplied with their respective weights. The resultant matrices are divided into multiple chunks (basically, the $d\_model$ is broken into smaller chunks), and self-attention values are calculated. They are called heads ($h$) and are concatenated and multiplied with an attention weight to form multi-head attention values.

***Residual Connection and Normalization:*** In both encoder and decoder networks, the layer normalization is followed by residual connections around each of the two sublayers in the encoder and around each of the three sublayers in the decoder. The residual connection, which is also called skip connection, allows the input to a particular layer to be added to its output by skipping a layer in between. This helps to alleviate the vanishing gradient problem, facilitate deeper networks, and improve learning capability.

The Layer Normalization technique [49] in DL is used to improve the training speed and help the network learn more effectively by improving internal covariate shifts (shifts in the distribution of activations as data moves through the network during training). With normalization, for each training sample, the mean and variance are computed across all neurons in a layer. The output from residual connection and normalization is given as equation 4:

$$LayerNorm(X + Sublayer(X)) \tag{4}$$

where $X$ is the input sequence after positional encoding and $Sublayer(X)$ is the function of the layer preceding the residual connection (e.g., Attention layer, FFN layer, etc.)

***Position-wise Feed Forward Network (FFN):*** In the encoder and decoder network, there is one sublayer of FFN, which operates independently on each position to capture local and positional non-linear relationships. It consists of two linear transformations and one non-linear activation function, denoted as the following function below as equation 5.

$$FFN(\mathbf{X}) = \sigma(W_1X + b_1)W_2 + b_2 \tag{5}$$

where $X$ is the input to the FFN layer, $W_1$ and $W_2$ are the two linear transformations and $b_1$ and $b_2$ are the corresponding biases and $\sigma$ is the non-linear activation function like RELU [50], GELU [51], etc.

#### 2.2.4 *Decoder Network*

The decoder network maintains the same structural stack as the encoder but incorporates two distinct modifications. The first one is the added sublayer called Masked Multi-Head Self-Attention, which takes input elements (mentioned as output in Fig. 2), which are shifted right, and future tokens are masked (this is because each token can only be influenced by previous tokens but not future ones). The second one is the Multi-Head Self-Attention mechanism implemented in the sublayer, followed by the masked self-attention sublayer, where the key and value pairs are passed from the output of the encoder network. This allows every position in the decoder to extract the attention of all positions in the input sequences at the encoder network.

***Linear, Softmax and Output:*** The output of the whole network is a linear vector of predictions of the next element produced after processing the input and output sequences, along with attention values and FFN processing. Using the Softmax function in the transformer architecture helps determine the percentage for considering each entity. Thus, it is used to normalize the vector numbers into probabilities and ensure that the output is a valid probability distribution summing up to one.

### 2.3 Pre-Trained Transformers

Pre-trained transformers are advanced ML models initially trained into a large corpus of data to learn the general representations. The concept of pre-trained models came into existence because DL models with many parameters need a much larger dataset to fully train the parameters. However, creating large-scale labeled datasets is a great challenge and costly. On the other hand, large-scale unlabeled datasets are relatively easier to create. Therefore to leverage this, pre-training approaches are typically applied to unlabeled data, known as self-supervised learning, which enables models to first learn the general representation from a large volume of data and use the learned representation to perform domain-specific downstream tasks by fine-tuning it. Various pre-trained models have been developed in recent years. Here, we discuss only the models that are used in the malware analysis domain.

***Bidirectional Encoder Representation from Transformer (BERT):*** BERT is a groundbreaking pre-trained encoder-only transformer model introduced by Devlin et al. [14], which revolutionized the NLP domain. It can understand the context of a word based on all its surroundings (both in left and right directions) in all model layers. Fine-tuning BERT adapts BERT to specific downstream tasks by continuing the training process on a task-specific dataset, which is a part of the transfer learning approach. Based on various aspects like

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



size, speed, efficiency, and data requirements, there are a lot of variants of BERT, such as DistilBERT [52], RoBERTa [53], AlBERT [54], etc.

***CANINE:*** CANINE [55] is a pre-trained transformer-based tokenization-free neural encoder trained on character-level sequences for language representation. Unlike other pre-trained models trained on tokenization approaches like words, sentences, or other forms of tokenizations, CANINE is pre-trained using character sequences without explicit tokenization or vocabulary. There are two pre-trained CANINE models - CANINE-s (pre-trained using subword loss) and CANINE-c (pre-trained on characters with autoregressive character loss).

***Vision Transformer (ViT):*** ViT [16] applies the transformer architecture based encoder directly to sequences of image patches, treating each patch as a token. It introduced a new way to handle 2D images by flattening and projecting them into an embedding space like word embedding in text processing.

***Generative Pre-trained Transformer (GPT2):*** GPT-2 [56] significantly expanded the original transformer architecture by adding many parameters and training data. It uses a stacked decoder-only architecture to generate text by predicting the next word. As GPT-2 was trained with extensive and diverse data from Internet, it is able to generate more coherent and contextual text across multiple domains.

## 3 Systematization: Transformers in Malware Analysis

In this section, we introduce taxonomies based on the comprehensive application of transformers, the use of transformers to solve challenges, variations of transformers applied to malware analysis, and the diverse feature representations and correlations processed with the transformers.

In Fig. 3 we show the different types of malware analysis tasks performed using transformers and in Fig. 4 a detailed taxonomic structure diagram. The taxonomy diagram consolidates and illustrates various modules and custom architectural enhancements used to represent different feature correlations. The visual structure maps out how diverse transformer architectures, from standard to custom-enhanced models, are tailored to capture and process a wide array of intricate patterns and correlations across different applications. We also present the extended in-depth scope of this topic in Appendix section 6 where we discuss in detail about how different transformers are adapted, enhanced and fine tuned for malware anaylsis tasks, along with evaluation and practical implementation insights.

The rest of this section is organized as follows. In Subsection 3.1 we discuss the application of transformer models to perform various malware analysis-based end tasks like detection, classification, etc. In Subsection 3.2 we discuss how the application of transformers is incorporated to address certain challenges in malware analysis. Then, in Subsection 3.3 we categorize different types of transformers applied to address specific tasks in malware analysis, and in Subsection 3.4 we discuss and categorize feature input types and representation techniques. Finally, in Subsection 3.5 we present the dataset inventory.

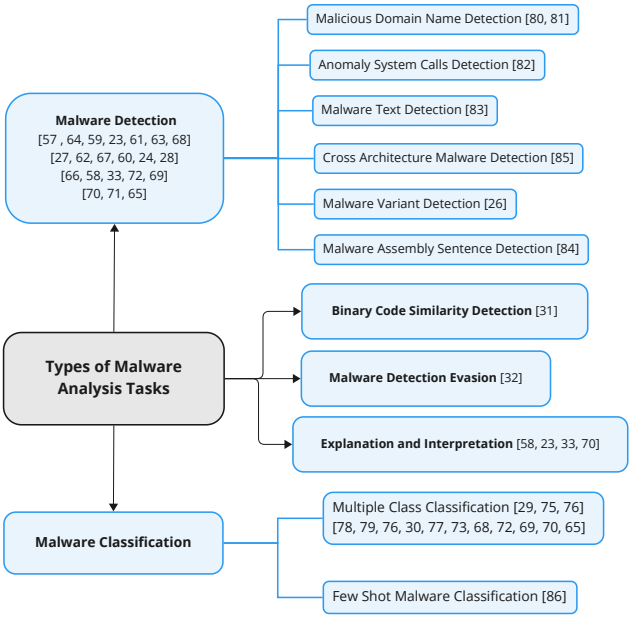


Fig. 3. Types of Malware Analysis Tasks performed using Transformers

### 3.1 Applications of Transformers in Malware Analysis

Our analysis shows that the adoption of transformer architectures, within the malware analysis domain, supports and enhances the conventional security tasks (see Fig. 3). From the figure, we can notice that a significant portion of the research has focused on malware detection [23]–[25], [27], [28], [33], [57]–[72] as an end goal while harnessing the transformers. These approaches vary significantly employing different methodologies, feature representation techniques, and model combinations which are discussed in further sections. Additionally, some of these detection tasks have been extended not only to perform classification [29], [30], [65], [66], [69], [70], [72]–[79] but also to provide explanation and interpretation [23], [33], [58], [70], thereby enhancing the transformer's application to security analysis. Beyond malware-specific detection objectives, we uncovered a range of other detection-focused approaches in the field of malware analysis. These include malicious domain name detection [80] [81], anomaly system call detection [82], malware-related text detection from threat reports [83], malware variant detection [26], malware assembly sentence detection [84], and cross-architecture based malware detection in IoT systems [85]. Moreover, a few unique approaches have addressed more complex aspects such as evasion techniques [32] and binary code similarity detection [31], further demonstrating the broad spectrum of applications for transformer technology in addressing a variety of malware threats.

Building upon the diverse end goals achieved through transformer architecture, most researchers have primarily utilized transformers for **generating contextual representations or embeddings** for input sequences (see Section 3.4.4 for additional discussion). This method utilizes the transformer's proficiency in capturing both local and long-range dependencies, crucial for processing complex data formats effectively. In addition to generating embeddings, numerous approaches have been proposed focusing on

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



other innovative applications of transformers. For instance, Hu et al. [32] developed MalGPT, fine-tuned on benign files, to generate benign-looking perturbations that could evade detection systems. This approach represents a strategic shift towards using transformers not just for detection, but for enhancing evasion techniques. Similarly, Shahid et al. [83] leveraged the transfer learning approach to extract critical cybersecurity terms from malware reports, thereby improving automated forensic analysis. Furthermore, Demirkıran et al. [77] introduced an ensemble of pre-trained transformers, BERT and CANINE (applied CANINE for the first time in the malware analysis domain), configured as a bagging-based Random Transformer Forest (RTF) to showcase the applicability of transformer forest which could outperform state-of-the-art models (performing multi-class classification) even with highly imbalanced datasets.

The transformative applications of transformers within the realm of malware analysis are profound and diverse which demonstrates their evolving role in advancing Cybersecurity defenses.

***Takeaway:*** *Transformers' versatility across multiple analysis tasks—ranging from detection and classification to evasion and interpretation—demonstrates their evolving role in enhancing cybersecurity defenses.*

### 3.2 Transformers Addressing Challenges in Malware Analysis: Efficiency, Effectiveness, and Adaptability

To tackle the challenge of emerging malware variants and various obfuscation techniques, Lu et al. [26] developed a comprehensive approach for creating malware variant datasets. With that new dataset, they perform adversarial training of the BERT-based detection model to enhance the robustness and generalization abilities to work under obfuscation conditions. They disassembled samples into the assembly language and inserted API call sequences while maintaining the original code execution structure. This obfuscation technique involves inserting code and then extracting API so that the obfuscated malware can be used as a variant of the original malware. This technique helped to create an adversarial data set and retrain the model to make it more aware and robust against obfuscation techniques.

To address the processing efficiency challenges posed by the transformer processing requirements, Li et al. [27] improved the training time efficiency and detection accuracy of the transformer-based malware detector by improving the standard transformer architecture. They introduced an additive attention mechanism - an improved version of the self-attention mechanism. Unlike the standard encoder, instead of calculating the dot product among Query($Q$), Key($K$), and Value($V$), which has a computational complexity of $O(N^2)$, they introduced a linear method to capture the context. Initially, the context information in $Q$ is **compressed** and **summarized** into a global query vector $q$ and it interacts with the matrix $K$ to produce the global matrix $k$. The resultant global context holding $k$ is multiplied by the value matrix $V$ to produce the attention values. In addition, they also replaced the standard residual connection and layer normalization with residual weight parameters $\alpha_i$. This adjustment also alleviates the gradient problem by allowing the model to dynamically adjust the residual contribution during training. These modifications collectively reduce the computational complexity by 5 times and accelerate the convergence of the network.

Furthermore, Ravi et al. [28] addressed the limitation of adaptability of transformers in resource-constraint environments such as IoT. They developed a lightweight, resource-efficient Vision Transformer (ViT) based model, ViT4Mal, for IoT malware detection, optimizing it for edge devices by simplifying its architecture. The limitation was addressed by reducing the computational complexity and memory requirements of the model. They reduced the dimensionality of image patch vectors through a linear projection in the patch embedding layer, used a learnable position embedding to the patch embedding, streamlined fine-tuned transformer encoder blocks, and simplified the decoder network without using an extra class token, unlike the original ViT architecture. After deploying the proposed lightweight model on the FPGA board (applying specialized hardware optimizations as well), they were able to speed up the processing by 41 times while maintaining the detection accuracy as compared to the original ViT.

To evaluate the cost-effectiveness of a transformer-based model for varying sequence lengths, Or-Meir et al. [29] compared it with various architectures, including LSTM, LSTM with Attention, Bi-LSTM and Bi-LSTM with Attention. They evaluated each model's time-cost effectiveness to accurately classify unseen malware with 22 varying input sequence lengths (ranging from 10 to 4000). Based on their observation, they found that the transformer's performance was costly and less effective for lower sequence lengths, whereas it performed exceptionally better with the increasing sequence length. This pattern highlights the transformer's suitability for handling larger data sequences.

Bu et al. [86] tackled the challenge of enabling deep neural networks to cope with an increasing number of unknown malware samples and the reliance on large labeled datasets for new malware. To address this, they developed a few-shot malware classification solution using a graph transformer trained with triplet loss [87]. This approach strategically applies transformers to handle novel malware samples in a few-shot scenarios, which are likely to be in real-world scenarios. Their method involves generating a control flow graph (CFG) from the malware's assembly code. The graph transformer selectively weights the correlations between nodes to capture the functional characteristics of the malware. The triplet loss function forces the network to learn a disentangled representation by minimizing the distance between similar malware samples and maximizing the distance between dissimilar ones. This approach enhances the transformer's ability to distinguish between malware types with minimal training data, improving the robustness of malware detection systems against obfuscation and variability in malware signatures.

***Takeaway:*** *Transformers are potentially capable of overcoming key challenges concerning computational efficiency, adaptation to obfuscation techniques, and resource usage reduction.*

### 3.3 Transformer Variants Used in Malware Analysis

Transformer architectures have been widely customized to tailor them to specific malware analysis tasks. We have

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



developed the taxonomy, shown in Fig. 4 to systematically organize all the resulting variations and their applications in capturing various intricate patterns and correlations. We have also consolidated information about the varieties of pre-trained and custom-enhanced transformers that were applied to malware analysis in Table 1.

#### 3.3.1 Standard (Vanilla) Transformer Architecture

Our analysis shows that the use of modular components is more ubiquitous than the use of the transformer architecture as a whole. As we did not find the use of a decoder-only module, we discuss only the attention mechanism and encoder-only modules in the section below.

***Attention Mechanism:*** This mechanism mathematically quantifies the importance of different segments of the input data and allows the model to learn intricate patterns and relationships within the sequences. In malware analysis, this allows the network to prioritize critical elements of input sequences, and dynamically adjust weights based on their relevance. A key aspect of the application of attention to malware analysis is the incorporation of positional encoding, which is vital for understanding the order and context of operations within the code. Positional encoding adds the necessary context to the model, helping it recognize the importance of sequence positioning in detecting malicious patterns. Thus, numerous approaches leverage the capabilities of attention mechanism, to learn intricate relationships within the input sequences and analyze them to make the decisions [29], [57], [59], [61], [63], [68]. In the context of transformer architectures, positional encoding typically uses 1D values to maintain the order of data sequences. However, to analyze network traffic bytes, Barut et al. [66] innovated within this framework by employing both 1D positional encoding and 2D convolutional feature embedding in their Residual 1-D Image Transformer (R1DIT) model. The term 1-D Image in R1DIT refers to the conceptual treatment of traffic byte sequences as one-dimensional data arrays that are processed using methods associated with image data using convolutional neural networks. Thus, the 1D positional encoding extracts the temporal sequence, ensuring that the model recognizes the order in which network events occur whereas the 2D convolutional feature embedding extracts and learns spatial features. This dual approach significantly improves the model's capability to detect malware based on network traffic behaviors.

***Encoder Only:*** The encoder-only module consists of an attention mechanism and the position-wise feed-forward network sublayer. This setup is crucial in emphasizing significant input parts and capturing complex non-linear relationships, making it suitable for malware analysis. It leverages the attention mechanism to emphasize significant parts of the input and feed-forward network to capture local and positional non-linear relationships. While it is consistently used across various studies, its application is uniquely tailored to address specific challenges in malware analysis. For example, Oliveira et al. [25] utilize the encoder's ability to enhance malware detection through advanced feature representation, while Li et al. [31] focus on decoding assembly instructions' syntactical and semantic nuances. Hu et al. [58] leverage the encoder to efficiently analyze IoT malware by harnessing self-attention from the encoder, and Trizna et al. [72] utilize the encoder for processing dynamic behavior patterns in varying sequence lengths. Similarly, Li et al. [75] and Guan et al. [82] highlight the encoder's role in understanding complex patterns in API call data and system call sequences, respectively. Moon et al. [73] emphasize the encoder's proficiency in managing long sequences, using its self-attention mechanism to capture the complex dependencies and structural nuances within Control Flow Graphs (CFGs). These varied applications demonstrate the encoder's versatility in advancing malware analysis.

> ***Takeaway:*** *The attention mechanism and encoder-only modular components are widely utilized in malware analysis. These components play a crucial role in capturing long-term dependencies and relationships within malware data, making them effective for various malware tasks.*

#### 3.3.2 Pre-Trained Transformers

In malware analysis, there is a broad use of pre-trained models as shown in Table 1, focusing on *encoder-only* and *decoder-only* architectures, alongside a mix of various attention mechanisms and embeddings. Here are the different types of pre-trained transformers used in the malware field.

***BERT and BERT Variants:*** BERT is used to generate robust embeddings by providing more nuanced indications of malware text [83], focusing on deep contextual characteristics from an NLP perspective to detect malware [30], [69], [74], addressing the challenge of detecting IoT malware with limited training data [62], and processing contextual understanding to analyze network traffic [33], [64]. Pandya et al. [67] compared the embeddings generated from BERT, DistillBERT, RoBERTa, and AlBERT using opcode sequences to classify malware classes, demonstrating superior accuracy over context-free embeddings.

***Fine-Tuned BERT:*** BERT has also been fine-tuned with specific data, like opcode sequences extracted from executable files [85] and by constructing obfuscated and unobfuscated malware variants [26], which reflect the real-world malware behavior post-obfuscation, to perform the downstream tasks of malware detection and classification.

***CANINE:*** The character-level processing ability of CANINE-c is leveraged by Gogoi et al. [81] to effectively analyze malicious domain names without tokenization. Demirkıran et al. [77] experimented on an ensemble of BERT and CANINE leveraging BERT's deep contextualized representations and CANINE's flexibility in handling raw text data without tokenization and also addressed the issue of imbalanced malware datasets for malware classification.

***GPT-2:*** GPT's advanced natural language processing capabilities with its attention mechanism have been leveraged to understand the context and relationship within opcode sequences [60] for malware detection. Assembly instructions extracted from `.text` sections of PE files are treated as sentences and documents to label as benign or malicious [84] using GPT-2. In contrast, Hu et al. [32] used GPT-2 to generate adversarial malware examples in a single-shot black-box setting by adding *benign-looking* perturbations in malwares.

***Vision Transformer (ViT) and Variants:*** There are innovative approaches that utilize the abilities of Vision transformer (ViT) [16] to capture long-range dependencies across

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



TABLE 1
Application of Pre-trained Transformers and Custom Enhancements/Improvements on Transformer Architecture for Malware Analysis

| Date | Transformer Type | Transformer Module | Objective of the Study | Objective of Integrating Transformer |
|---|---|---|---|---|
| | | | **Pre-Trained Transformers** | |
| 2020 | BERT [83] | Encoder Only | Forensic Analysis of Threat Reports | Semantic Extraction of Malicious terms |
| 2021 | BERT [62] | Encoder Only | Detect IoT Malware with Limited Training (Explore BERT Model) | Enhance Detection with Robust Embeddings |
| 2021 | BERT [74] | Encoder Only | Classify Android Malware | Enhance Classification with Robust Embeddings |
| 2021 | Fine Tuned BERT [85] | Encoder Only | Cross-architecture IoT malware Detection (Explore Fine-Tuning of BERT Model) | Enhance Detection with Robust Embeddings |
| 2021 | GPT-2 [60] | Decoder Only | Explore GPT2 for Malware Detection | Enhance Detection with Robust Embeddings |
| 2021 | GPT-2 [32] | Decoder Only | Malware Detection Evasion using Benign Looking Perturbations | Construct Adversarial Malware Examples in a Single shot Black-box Setting |
| 2021 | BERT [64] | Encoder Only | Detect IoT Network Intrusion (Explore BERT Model) | Enhance Detection with Robust Embeddings |
| 2021 | BERT [33] | Encoder Only | Interpret IoT Malware Detection | Enhance Detection by Integrating with Multi-modal System |
| 2022 | Fine Tuned BERT [26] | Encoder Only | Construct Malware variants and Robustify Malware Detection | Enhance Robustness against Obfuscated Techniques |
| 2022 | Ensemble of BERT and CANINE [77] | Encoder Only | Explore Random Transformer Forest for Highly Imbalanced Class Dataset for Malware Classification | Enhance Classification with Robust Embeddings |
| 2022 | GPT-2 [84] | Decoder Only | Explore GPT-2 as Natural Langugage Model to Detect Malicious Assembly Sentences | Enhance Detection by Integrating with Multi-modal System |
| 2022 | ViT [76] | Encoder + Lambda Attention | Malware Classification with Reduced Computational Complexity and Memory usage using Lambda Attention | Enhance Malware Classification with Robust Embedding |
| 2022 | ViT [78] | Encoder Only + Patch Encoding | Malware Classification with Enhanced Existing Model using Patch Encoding | Enhance Malware Classification with Robust Embedding |
| 2023 | ViT [65] | Encoder Only | Explore Self-Supervised Learning and Transfer Learning for Downstream Tasks of Malware Classification | Enhance Malware Classification with Robust Embedding |
| 2022 | BERT [30] | Encoder Only | Classify Android Malware | Enhance Classification with Robust Embeddings |
| 2022 | BERT [69] | Encoder Only | Detect and Classify Android Malware | Enhance Detection and Classification of Android Malware with Robust Embeddings |
| 2023 | BERT, DistilBERT, RoBERTa, ALBERT [67] | Encoder Only | Comparison of BERT based Models to Detect Malware | Enhance Detection with Robust Embeddings |
| 2023 | CANINE-c [81] | Encoder Only | Explore CANINE Model in Malicious Domain Name Detection | Enhance Malicious Domain Detection with Robust Embedding |
| 2023 | ViT [70] | Encoder Only + Attention | Interpret Malware Detection and Classification using Attention Maps | Enhance Malware Classification with Robust Embedding |
| | | | **Custom Enhancement and Improvements** | |
| 2021 | Novel Transformer [23] | Galaxy Transformer | Explore Unique Arrangement of Star Transformers to Enhance Malware Detection and Interpretation against Obfuscations Techniques | Complex Sequence Processing and Generate Robust Embedding |
| 2023 | Novel Transformer [24] | Heterogeneous Temporal Graph Transformer (HTGT) | Explore Combination of Transformers to Model Malware Evolution and Propagation Pattern | Enhance Malware Detection with Robust Embedding |
| 2023 | Novel Transformer [86] | Graph Transformer | Enhance Few-shot Classification with Less Volume of Data | Enhance Effectiveness of Few-shot Malware Classification with Robust Embeddings |
| 2023 | Improved Vanilla Transformer [27] | Additive Attention + Residual weight parameters | Explore Custom Improvements to Enhance Detection | Increase Efficiency of Model by Reducing Computational Complexity |
| 2023 | Improved Vanilla Transformer [71] | AdaTrans | Integrate Inter-component Communication (ICC) to Enhance Detection | Enhance Malware Detection with Robust Embedding |
| 2023 | Improved ViT Transformer [79] | B_ViT | Explore Global-local Attention to Enhance Malware Classification and Resilience to Polymorphic Obfuscation | Enhance Classification by Integrating with Multi-modal system |
| 2023 | Improved ViT Transformer [28] | Lightweight ViT : VT4Mal | Explore Adaptability in Resource-constrained Environment by Developing Lightweight Vision Transformer | Enhance Detection with Robust Embedding and Explore Adaptability of Transformer in Low-Resource Environment |

the entire image without being limited by the local receptive fields as that in Convolutional Neural Networks (CNNs).

Seneviratne et al. [65] employed a self-supervised approach using a ViT-based Masked Auto Encoder, SHERLOCK, where the input image patches are masked and the model is trained to reconstruct the masked patches. Once the self-supervised training of SHERLOCK is completed, the learned representations are then transferred to fine-tune 3 different downstream models performing binary classification, multi-class (47 categories), and family classification (696 categories). The model achieved high accuracy in all 3 classification tasks outperforming models like ResNet, DenseNet, and MobileNetV2. In this study, the transfer learning approach using learned representations for downstream tasks appears faster, more convenient, and less data-heavy as compared to training those models from scratch.

Chen et al. [76] introduced Lambda attention in the Encoder, which reduced the computational complexity from quadratic to linear by abstracting the interaction between query, key, and value into lambda functions and effectively aggregating the context from the input sequence. Thus, they simplified the complexity from $O(N^2)$ to $O(N)$. Park et al. [78] also introduced a technique distinct from the standard ViT. By adding patch by embedding along with the encoder which splits the input into patches and applies a transformer encoder directly, the approach adds an extra encoding step for each patch, which enriches the input data with more focused local features and positional information.

Furthermore, Jo et al. [70] used ViT to enhance malware detection and interpretability. The input images are passed through ViT to interpret the attention-based focus on different parts of the images and find the most relevant features for detection using the attention map from the ViT model.

> ***Takeaway:*** *The integration of pre-trained models provides a powerful foundation for malware detection workflows, reducing the need for large-scale labeled datasets while at the same time improving accuracy with fine-tuning, making them highly practical for real-world applications.*

#### 3.3.3 *Custom Enhancements to Standard Architecture*

Many researchers have focused on enhancing the existing transformer architecture by introducing novel architectural changes and unique arrangements as shown in Table 1. Li et al. [23] proposed a novel Galaxy transformer based on the arrangements of star transformers [88] at the fundamental level, inspired from the heavenly bodies, creating three

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



components based on the organization of star transformers: satellite-planet (to understand the basic blocks), planet-star (to understand assembly functions), and star-galaxy transformer (to understand the comprehensive semantics).

Bu et al. [86] proposed a novel graph transformer designed to handle control flow graphs. It incorporates a directional embedding mechanism to capture sequential nature of control flow within the graph. It also adapts multi-head attention to prioritize critical attack paths and functional characters of malware samples. It integrates triplet loss function to learn a disentangled representation which ensures similar malware samples are embedded closely. With these integrations, they propose few-shot based malware classifier that mitigates the dependency on sample volume and performs distance-based malware classification.

In Android malware analysis, Fan et al. [24] proposed a novel Heterogeneous Temporal Graph transformer (HTGT), a combination of novel heterogeneous spatial transformer (to capture heterogeneity attention over each node and edge) and novel heterogeneous temporal transformer (to aggregate its historical sequences of a given node attentively), specifically designed to model malware's evolution and propagation pattern to learn their latent representation. HTGT (or called Dr. Droid) outperformed multiple state-of-the-art comparable models and showcases more than 98% true positive rate in detecting novel malware samples. The model was deployed in the anti-malware industry to serve over 700 million mobile users worldwide against evolving android malware attacks.

Besides custom architectures and unique arrangements, there are several improvements made to the existing Vanilla transformer as well. In IoT malware analysis, to address the issue of computational complexity, Li et al. [27] introduced additive attention, which converts the quadratic complexity into a linear, as discussed in Section 3.2. The authors claim that their approach decreases the complexity five times as compared to the standard architecture. Similarly, Pi et al. [71] proposed AdaTrans, an adaptive transformer based on adaptive multi-head attention network to detect malware that uses inter-component communication. Adaptive attention modifies how attention scores are computed by adjusting the weights based on relevance offering a more flexible way to focus only on the most informative parts.

Furthermore, several approaches optimize the Vision transformer (ViT) [16]. Belal et al. [79] proposed Butterfly_ViT, which incorporates a global-local attention mechanism, arranging the multiple-attention mechanism in such a way that it partitions images into segments to capture detailed local features and uses an entire image to extract global details in contrast to the standard ViT, which focuses uniformly across the whole image. However, Butterfly is computationally expensive, but harnessing parallel processing and leveraging the local-global representation seems to be a good approach. Ravi et al. [28], to cope with the limited resource constraints in IoT systems, proposed a ViT4Mal (a lightweight vision transformer) for edge devices by improving several aspects of standard ViT. They reduced the dimension in the patch embedding layer by projecting image segments linearly, streamlined the encoder by adjusting the number and complexity of the block used, and optimized hardware through quantization of model weights and activation, loop pipe-lining, and array partitioning.

Conclusively, from a higher level perspective, we observed that overall performance measures highlight the exceptional accuracy of transformer models in malware detection, often achieving accuracy greater than 95%. Custom architectures that blend transformers with task-specific components have shown significant improvements. While high accuracy is common, there is often a trade-off regarding computational cost. Complex models, although requiring more resources, yield superior accuracy, demonstrating that the selection of a model should balance accuracy with available computational resources. For a more comprehensive understanding of transformer adaptations, their effectiveness, and performance measures, refer to the detailed insights in Appendix Sections 6.1 and 6.2.

> ***Takeaway:*** *Custom enhancements to transformer architectures provide a significant leap in their application to malware analysis, making them more effective, scalable, and adaptable.*

### 3.4 Feature Representations using Transformers

When applying AI techniques to malware analysis, proper feature representation is a critical design choice, as it directly impacts the model's ability to accurately analyze malicious software. Using appropriate feature representations, transformers can better understand and interpret complex malware behaviors.

This section focuses on different types of feature and correlation representations using transformer models, as outlined in Table 2, demonstrating their flexibility in analyzing malware through diverse combination of features. We first categorize the features into 11 types as listed below, examining how these are extracted, generated, and transformed. Next, we discuss the integration of transformers in multi-modal systems, highlighting how combining textual, visual, and structural data representations can enhance malware analysis. Then, we categorize correlations into 7 different representation types and discuss how different studies harness transformers to generate robust embeddings for the different input features.

#### *3.4.1 Feature and Input Types*

As shown in Table 2, we have categorized features into 11 feature types focusing on how they are used as the *medium* of representation. We also analyze approaches for extracting, generating, or transforming features.

***Binary Sequences:*** These are the sequence of 0s and 1s, representing the executable codes. In malware analysis, using streams of binaries from the executables, the streams are transformed into images and patches of images [28], [65], [70], [78], [79] to visualize unseen aspects of malware.

***Opcode Sequences:*** Opcodes are the low-level instruction sets executed by the CPU (hexadecimal form). In malware analysis, these sequences allow one to identify patterns of malware by analyzing the contextual relationship between the sequence of opcodes. The sequences are employed in diverse ways: as n-gram words for textual analysis [27], to represent code blocks [57], converted into images for visual analysis [76], mapped to functions for structural insights [58], and utilized as text sequences [32], [67], [85].

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



TABLE 2
A Comprehension of Feature Representation Techniques Utilizing Various Transformer Models for Diverse Analytical Objectives

| Surveyed Works | Year | Feature Types | | | | | | | | | | Input Types | | | Feature Co-relations Types | | | | | | | Transformer Modules | | | | | | | | | Analysis Objectives | | | | Analysis Methods | | | Transformer |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1. API & System Call | 2. Assembly code | 3. Binary Sequence | 4. Domain name | 5. File System Info | 6. Function call | 7. Manifest File | 8. Network Behavior | 9. Opcode sequence | 10. System Behavior | 11. Others | 1. Graph | 2. Image | 3. Text | 1. Functional | 2. Semantic | 3. Sequential | 4. Spatial | 5. Structural | 6. Syntactic | 7. Temporal | 1. Attention | 2. Encoder Only | 3. Decoder Only | 4. Graph Transformer | 5. HST | 6. Star Transformer | 7. Temporal Transformer | 8. Custom Attention | 9. Custom Changes | 1. Detection | 2. Classification | 3. Explanation | 4. Others | 1. Static | 2. Dynamic | 3. Network | |
| Chen et. al. [57] | 2020 | | | | | | | | | ✓ | | | | | ✓ | | ✓ | | | | | | ✓ | | | | | | | | | | ✓ | | | ✓ | | | Vanilla Transformer |
| Shahid et. al. [83] | 2020 | | | | | | | | | | | ✓ | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | | | | ✓ | | | | BERT |
| Hu et. al. [58] | 2020 | | | | | | | | | ✓ | | | | | ✓ | | ✓ | | | ✓ | | | | ✓ | | | | | | | | ✓ | | ✓ | | | ✓ | | Vanilla Transformer |
| Moon et. al. [73] | 2021 | | ✓ | | | | | | | | | | ✓ | | | ✓ | | | | ✓ | | | | ✓ | | | | | | | | | ✓ | | | ✓ | | | Vanilla Transformer |
| Hamad et. al. [85] | 2021 | | | | | | | | | ✓ | | | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | | | | ✓ | | | Fine Tuned BERT |
| Long et. al. [59] | 2021 | ✓ | | | | | | | ✓ | | | | | | ✓ | | | ✓ | | | | ✓ | | | | | | | | | | ✓ | | | | | ✓ | | Vanilla Transformer |
| Guan et. al. [82] | 2021 | ✓ | | | | | | | | | | | | | ✓ | | | ✓ | | | | | | ✓ | | | | | | | | ✓ | | | | | ✓ | | Vanilla Transformer |
| Or-Meir et. al. [29] | 2021 | ✓ | | | | | | | | | | | | | ✓ | | | ✓ | | | | ✓ | | | | | | | | | | | ✓ | | | | ✓ | | Vanilla Transformer |
| Li et. al. [23] | 2021 | | ✓ | | | | | | | | | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | | | | | | | ✓ | | | | ✓ | | ✓ | | | ✓ | ✓ | | Galaxy Transformer |
| Rahali et. al. [74] | 2021 | | | | | | | ✓ | | | | | | | ✓ | ✓ | ✓ | | | ✓ | | | | ✓ | | | | | | | | | ✓ | | | | ✓ | | BERT |
| Sahin et. al. [60] | 2021 | | ✓ | | | | | | | | | | | | ✓ | | ✓ | | | | ✓ | | | | ✓ | | | | | | | ✓ | | | | ✓ | | | GPT-2 |
| Hu et. al. [32] | 2021 | | | | | | | | | ✓ | | | | | ✓ | | ✓ | | | | | | | | ✓ | | | | | | | | | | ✓ | ✓ | | | GPT-2 |
| Oliveira et. al. [25] | 2021 | ✓ | | | | | | | | | | | | | ✓ | | ✓ | ✓ | | | ✓ | | | ✓ | | | | | | | | ✓ | | | | ✓ | ✓ | | Vanilla Transformer |
| Wangwang et al. [61] | 2021 | | | | | | | | ✓ | | | | | | ✓ | | ✓ | | | | | ✓ | | | | | | | | | | ✓ | | | | | ✓ | | Vanilla Transformer |
| Bellante et. al. [62] | 2021 | ✓ | ✓ | | | ✓ | | | ✓ | | ✓ | | | | ✓ | | ✓ | ✓ | | | | | | ✓ | | | | | | | | ✓ | | | | ✓ | ✓ | ✓ | BERT |
| Fan et al. [24] | 2021 | ✓ | | | | | | ✓ | | | | ✓ | ✓ | | | | | | ✓ | | | ✓ | | | | | ✓ | | ✓ | | | ✓ | | | | ✓ | ✓ | | HTGT |
| Ullah et. al. [33] | 2022 | | | | | | | | ✓ | | | | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | | ✓ | | | | ✓ | BERT |
| Li et. al. [75] | 2022 | ✓ | | | | | | | | | | | | | ✓ | | | ✓ | | | | | | ✓ | | | | | | | | | ✓ | | | ✓ | ✓ | | Vanilla Transformer |
| Chen et. al. [76] | 2022 | | | | | | | | | ✓ | | | | ✓ | | | | | ✓ | | | | | ✓ | | | | | | ✓ | | | ✓ | | | ✓ | | | ViT (Vision Transformer) |
| Lu et. al. [26] | 2022 | ✓ | | | | | | | | | | | | | ✓ | ✓ | ✓ | | | | | | | ✓ | | | | | | | | ✓ | | | | | ✓ | | Fine Tuned BERT |
| Qi et. al. [63] | 2022 | ✓ | | | | | | | | | | | | | ✓ | | ✓ | | | | ✓ | ✓ | | | | | | | | | | ✓ | | | | | ✓ | | Vanilla Transformer |
| Demirci et. al. [84] | 2022 | | ✓ | | | | | | | | | | | | ✓ | | ✓ | | | | | | | | ✓ | | | | | | | ✓ | | | | ✓ | | | GPT-2 |
| Ghourabi et. al. [64] | 2022 | | | | | ✓ | | | ✓ | | ✓ | | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | | | | ✓ | | ✓ | BERT |
| Demirkiran et. al. [77] | 2022 | ✓ | | | | | | | | | | | | | ✓ | | | | | | | | | ✓ | | | | | | | | | ✓ | | | ✓ | ✓ | | Random Transformer Forest |
| Seneviratne et. al. [65] | 2022 | | | ✓ | | | | | | | | | | ✓ | | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | ✓ | | | ✓ | | | ViT |
| Park et. al. [78] | 2022 | | | ✓ | | | | | | | | | | ✓ | | | ✓ | | | ✓ | | | | ✓ | | | | | | | ✓ | | ✓ | | | ✓ | | | ViT |
| Yang et. al. [80] | 2022 | | | | ✓ | | | | | | | | | | ✓ | | ✓ | ✓ | | | | | ✓ | | | | | | | | | ✓ | | | | ✓ | | | Vanilla Transformer |
| Barut et. al. [66] | 2022 | | | | | | | | ✓ | | | | | | ✓ | | | | ✓ | | | ✓ | ✓ | | | | | | | | | ✓ | ✓ | | | | ✓ | | Vanilla Transformer |
| Rahali et. al. [30] | 2023 | | | | | | | ✓ | | | | | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | | ✓ | | | ✓ | | | BERT |
| Gogoi et. al. [81] | 2023 | | | | ✓ | | | | | | | | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | | | | ✓ | | | CANINE Transformer |
| Li et. al. [31] | 2023 | | ✓ | | | | | | | | | | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | | | | ✓ | | | Vanilla Transformer |
| Pandya et. al. [67] | 2023 | | | | | | | | | ✓ | | | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | | | | ✓ | | | BERT, DistilBERT, RoBERTa, AlBERT |
| Deng et. al. [68] | 2023 | ✓ | | | | | ✓ | | | | | | ✓ | | | | ✓ | | | | | | ✓ | | | | | | | | | ✓ | | | | ✓ | | | Vanilla Transformer |
| Li et. al. [27] | 2023 | ✓ | | | | | | | | ✓ | | | | | ✓ | | ✓ | | | | | | | | | | | | | ✓ | ✓ | ✓ | | | | ✓ | | | Improved Vanilla Transformer |
| Belal et. al. [79] | 2023 | | | ✓ | | | | | | | | | | ✓ | | | | | ✓ | | | | | | | | | | | ✓ | ✓ | | ✓ | | | | ✓ | | Butterfly ViT |
| Saracino et. al. [69] | 2023 | ✓ | | | | | | | | | | | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | ✓ | | | ✓ | | | BERT |
| Ravi et. al. [28] | 2023 | | | ✓ | | | | | | | | | | ✓ | | | | | ✓ | | | | | ✓ | | | | | | | | ✓ | | | | ✓ | | | Lightweight ViT |
| Jo et al. [70] | 2023 | | | ✓ | | | | | | | | | | ✓ | | | ✓ | | | ✓ | | | ✓ | ✓ | | | | | | | | ✓ | ✓ | ✓ | | ✓ | | | ViT |
| Pi et al. [71] | 2023 | | | | | | ✓ | | | | | | ✓ | | | | ✓ | | | | | | | | | | | | | ✓ | | | ✓ | | | ✓ | | | Improved Vanilla Transformer |
| Trizna et al. [72] | 2023 | ✓ | | | | ✓ | | | ✓ | | ✓ | | | | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | ✓ | | | | ✓ | | Vanilla Transformer |
| Bu et al. [86] | 2023 | | ✓ | | | | | | | | | | ✓ | | | ✓ | | ✓ | | | | | | | | ✓ | | | | | | | ✓ | | | ✓ | | | Proposed Transformer |

***Assembly Codes:*** Assembly codes are expressed in low-level programming languages, closely related to the machine code and are readable by humans. In malware analysis, such readable assembly codes are usually obtained from executable files using disassembling tools and used as a sequence of text [31], [60], [84]. The goal of such transformation is to extract contextual relationships among the opcodes and operands in the codes. Assembly codes can also be combined with other features extracted from different dynamic analyses [23], [62] to enrich the set of features. Moreover, Moon et al. [73] used Control Flow Graphs (CFGs) generated from assembly codes to represent jumps and function calls. Bu et al. [86] extracted CFGs from assembly to simulate malware attack paths & propagation.

***API and System Call Sequences:*** API calls are requests made by programs to external libraries or services, and the system calls are requests made directly to the operating system (OS), which also represent interactions with the hardware. Several approaches have used API call sequences [26], [63], [75], [77] and system call sequences [25], [29], [82] that are usually extracted from sandbox (dynamic analysis) environments. API and system call sequences can also be combined with other features [27], [59], [62], [72] to construct long call sequences and extract insights about the malware's actions and intentions. On the other hand, Fan et al [24] and Deng et al. [68] proposed approaches to convert API call sequences into graphs as feature representations, and conversely, Saracino et al. [69] proposed an approach to generate API call sequences from the API call graphs. These contrasting approaches highlight different analytical perspectives. In contrast, generating sequences from graphs can simplify the analysis by linearizing the relationships, making it easier to apply sequence-based machine learning models.

***Function Calls:*** To uncover hidden or latent malware behaviors, sub-graphs generated from critical function calls (API calls that indicate potentially malicious behaviors) are used in the approach by Deng et al. [68], whereas Pi et al. [71] integrated Inter-component communication in the sensitive function call sub-graphs.

***File System Information:*** A few approaches use file system information and other features to enrich the features' strength. Relevant features of file system include a large variety of static information, such as file metadata (size, creation date, and format) [62], file operation path [72], header information [64], and file properties such as name, size, and list of strings [64], as well as dynamic information, such as file operation type [72], imported and exported functions, and properties sections [64].

***Manifest File Information:*** In Android systems, manifest files play an important role as they provide essential information about the OS, permissions the app requires, activities used, services used, broadcast receivers used, content providers used, version supported, etc. In malware analysis, Rahali et al. [30], [74] proposed an approach using manifest files to reveal suspicious permissions or uncommon features, and Fan et al. [24] proposed an approach using manifest files as one of the features to map the malware

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



propagation and reveal attack path.

***Network Behaviors:*** Malware analysis may also analyze network-based patterns or actions related to how malware communicates or behaves over the network. Network properties used in malware analysis include protocol information, data in transfer, network addresses [62], connection ports and server names [72], dynamic feature items (extracted with strace and tcdump from system calls and IP activities) [59], traffic data (IP, HTTP, DNS, unencrypted TLS record headers) [66] and packet files (.pcap) [33], [61], [64].

***System Behaviors:*** In malware analysis, the behavior of malware that showcases the interactions and activities on the OS level are tracked as system behaviors to detect malicious activities. System behaviors like processes (execution, termination, manipulation) [62], file modification (creation, deletion, or alteration) [62], registry key accesses, access type and key-value [62], [72], monitoring logs and behaviors (sensor data, OS logs) [64] are used to map the system behaviors. These behaviors are used as one of the features rather than using them as a standalone feature set.

***Domain Names:*** Domain names play a crucial role in network malware analysis as they can be used to identify command-and-control servers, phishing sites, and other malicious destinations. Features, such as domain names and n-gram location, are often used to classify the semantic patterns into benign and malicious categories. Gogoi et al. [81] leverage character-level analysis using the CANINE transformer to analyze domain names without tokenization, enhancing the detection accuracy for malicious domains. Similarly, Yang et al. [80] utilize n-gram features to capture the contextual information within domain names, further improving the classification of malicious domains.

***Others:*** In addition to the commonly used features, some approaches incorporate unique features for malware analysis. Static features, like printable string, PE imports, and PE header numerical, were used by Li et al. [23] to observe the behavior, purpose, and origin of a file. Fan et al. [24] used Android-based dynamic features such as loaded dex files, connected URLs, generated texts, and the application's social information (application name, affiliation, market, signature) to map the malware propagation behaviors. Shahid et al. [83] used natural language features obtained from malware threat report sentences to automate the detection of malicious terms from the NLP reports.

> ***Takeaway:*** *The ability of transformers to process a wide range of feature types enhances their capability to analyze malware from multiple angles, providing a holistic understanding of malicious behaviors. This flexibility allows for a combination of static, dynamic, and network-based analysis within a single framework.*

#### 3.4.2 Transformers Integration in Multi-Modal System

Multi-modal ML refers to the integration and processing of data from multiple modalities, such as text, images, and graphs, to enhance the analysis and understanding of complex information. In the context of malware analysis, this approach is intriguing as it allows for a more comprehensive understanding of malware behavior by leveraging different types of data, which can reveal various aspects of malicious activities that might be missed when considering a single modality. In malware analysis, integrating transformers or modules of transformers with multi-modal data opens up new possibilities for exploring the synergistic potential of multi-modal combinations from multiple perspectives.

Recent approaches for *windows malware* detection show that integrating transformers in multi-modal systems is effective. The MDFA framework proposed by Qi et al. [63] leverages the Bi-LSTM and attention mechanisms to analyze API call sequences, capturing both temporal dependencies and critical sequence patterns. Demirci et al. [84] enhanced static malware detection by combining stacked BiLSTM and GPT-2 models, treating assembly instructions(as word, sentence, and document) as textual data to extract syntactic and semantic features. The Belal et al.'s [79] Global-Local Attention-Based Butterfly Vision Transformer model employs global and local attention mechanisms to process image and textual data, enabling the detection of complex malware patterns through a comprehensive analysis and offering resilience to polymorphic obfuscation.

In the *IoT malware* analysis domain, Ullah et al. [33] developed an explainable malware detection system that employs transformers-based transfer learning and multi-modal visual representation, combining BERT (with a self-developed FAST extractor) extracted textual features with malware-to-image conversions (using self-developed BRIEF descriptor) for visual analysis. Their approach, which also incorporates CNN for deep feature extraction, emphasizes the importance of explainable AI in making detection processes transparent and reliable. To detect *android malware*, Oliveira et al. [25] proposed a multi-modal deep learning network, called Chimera. The multi-network processes three different feature types - *permission and intents*, *API call sequences*, and *apk images*, individually fed into Deep Neural Network (DNN), Transformer Encoder Network (TN), and Convolutional Neural Network (CNN), respectively. Further, at the fusion layer, the collected representations from each sub-networks are concatenated as an ensemble of those subnetworks thus outperforming the classical ML methods.

Additionally, to analyse *linux malware*, Guan et al. [82] proposed a hybrid LSTM-Transformer model for system call anomalies detection. This hybrid approach highlights the value of combining LSTM's capability to handle sequential data with transformers' strength in learning global dependencies. Furthermore, Barut et al. [66] introduced the Residual 1-D Image Transformer (R1DIT) for privacy-preserving malware traffic classification. This model leverages raw data transformation and attention-based modules to classify different malware types and benign traffic without interfering with IP addresses, port numbers, and the payload. Their results demonstrate superior accuracy and generalization, especially for handling new traffic types like TLS 1.3.

> ***Takeaway:*** *Transformers integrated into multi-modal systems offer a powerful approach to malware analysis, as they can draw on diverse data types to provide an holistic view of malicious behaviors, particularly in handling complex, polymorphic, or evasive malware.*

#### 3.4.3 Transforming Feature Correlations Through Transformer Integration

Representing feature correlations involves capturing the relationships and dependencies between different data inputs. This is crucial because understanding these relationships

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



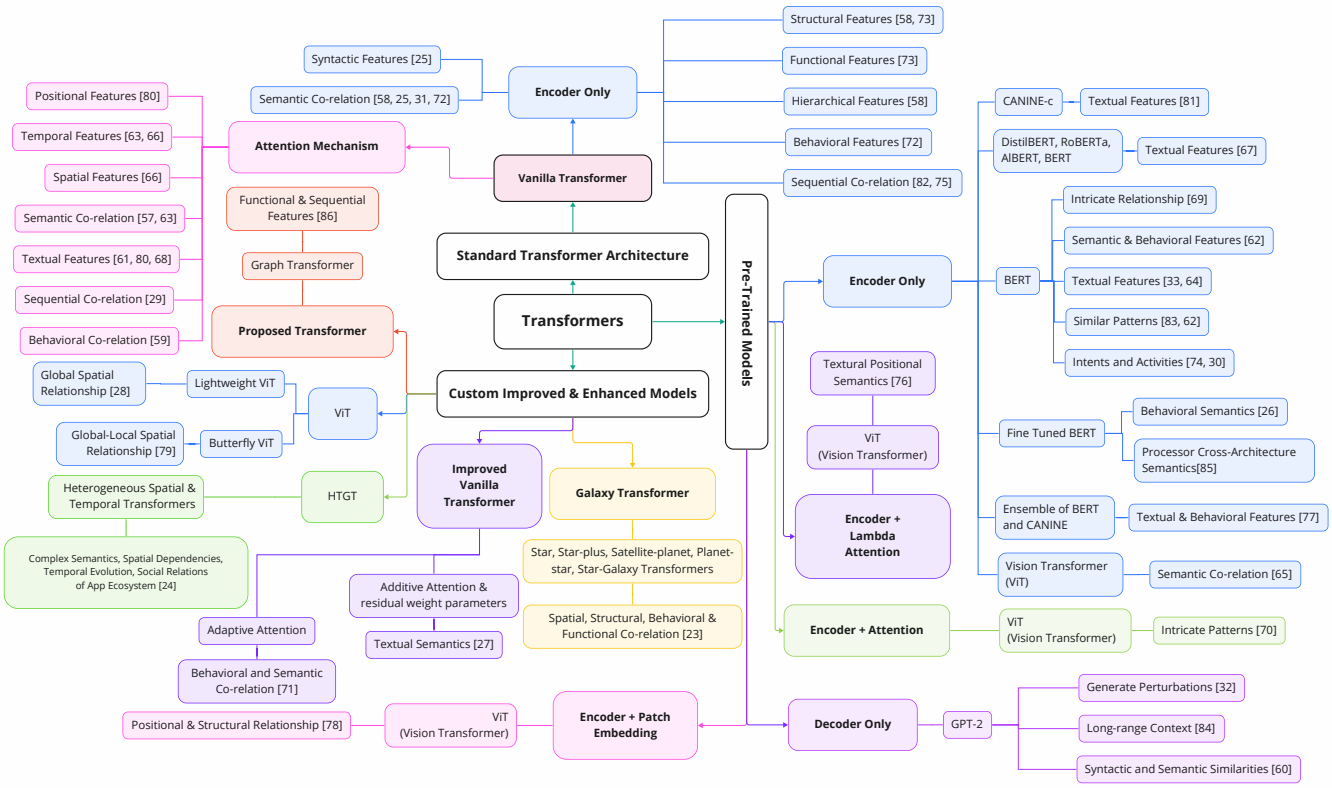


Fig. 4. Taxonomy of transformers that are applied in malware analysis to capture various correlations and relationships from the input features

helps in accurately analyzing malware behaviors. Transformers are particularly well-suited for this task due to their ability to model complex interactions within the data. As detailed in Table 2, we have categorized the correlations, studied in the malware domain, into seven types. This categorization provides a structured overview of how various studies address the representation of features correlations in their analyses. Additionally, Fig. 4 shows how different feature correlation representation techniques are captured with the integration of various transformer types.

***Functional Correlation:*** It refers to the relationship between different parts of the code that perform specific tasks or functions. In malware analysis, research has focused on functional relationship extraction by using assembly codes converted as control flow graphs [73], [86], API and system call sequences [26], manifest files [74], and basic blocks as hierarchical structure (organization of data or entities in layers of increasing complexity or specificity) [23] to map malware intents and activities.

***Semantic Correlation:*** It refers to the meaningful relationship between data based on their meaning or context. Most of the reviewed literature [23], [25]–[27], [30]–[33], [57], [58], [60]–[65], [67]–[72], [74], [77], [78], [80], [81], [83]–[85] extracts the semantic relationship using a variety of features. Semantic representations are widely used to observe malware behaviors, complex patterns, hierarchical structure, long-range dependencies, malware intents & activities, etc.

***Sequential Correlation:*** It refers to the relationships between the data points that depend on their order or sequence. In the malware domain, mainly the system and API call sequences [25], [29], [59], [62], [75], [82] as text along with the domain as n-gram words [80] and the assembly codes as graphs [86] are used to investigate the position-dependent malicious nature of malware.

***Spatial Correlation:*** It refers to the physical or logical relationship between data entities based on their spatial arrangement or proximity. In malware analysis, features are converted to images [28], [76], [79] to observe spatial patterns, graphs [24] are used to map dependencies, and texts [66] are used to map the hierarchical structure [23].

***Structural correlation:*** Unlike spatial correlation, structural correlation deals with the arrangement and connections between components, defining how they are put together to form a larger system. In the malware domain, there is the use of features as images [70], [78], as graphs [73], features as texts to map the hierarchical structure [23], [58] and malware intents & activities [74].

***Syntactic Correlation:*** It is based on the structural rules i.e. syntax of data. In malware analysis, these correlations are observed in long text sequence analysis [25], [60].

***Temporal Correlation:*** It refers to relationships between data entities that change over time. In the malware domain, capturing malware behaviors that evolve with time [24], tracking of the timing of API calls [63], and network behaviors [66] using timestamps are experimented.

> ***Takeaway:*** *The ability of transformers to capture a wide array of feature correlations enables them to detect more complex and sophisticated malware, as well as pave the way to model malware behaviors using multi-correlations. This makes them invaluable for defending against evolving security threats.*

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



#### 3.4.4 Generating Robust Embedding Representations using Transformers

In this subsection, we cover approaches that use modules of standard or improved transformer architectures for robust embedding representations as high-dimensional data from different input types: images, graphs, and texts. These embeddings aim to capture critical information and relationships into a more manageable, higher-dimensional space.

***Images to Represent Features:***

When processing images as input, the vision transformer (ViT) [16] is most commonly used. As mentioned in Section 2.3, the ViT is based on an encoder-only module. It processes patches from input images as tokens and generates the embedding representation in higher-dimensional space. The embeddings are further used as per the goal of the project. Utilizing ViT for feature representation from images is increasingly common. Such an approach, detailed in the *Survey on Vision transformer* by Han et al. [18] and in the *Survey of transformer in Vision* by Khan et al. [17], i.e., representing features with ViT for robust embeddings are found in malware analysis as well. Though there are several approaches for generating images to represent malware samples, the role of ViT usually appears once the input images are converted into patches and they are provided to ViT as input tokens to generate and represent the images into higher dimensional embeddings. Park et al. [78] used ViT to encode positional information of image patches and sequential information between the local features of the malware image. Unlike Park's classification objective, Seneviratne et al.'s [65] goal is to learn malware image reconstruction and classify malware samples, but the core use of ViT is similar in both cases - to generate feature representations. In addition to the detection and classification task using the embeddings from ViT, Jo et al. [70] propose to use a ViT attention map to provide explanations about the evaluation of malware samples. While such approaches use the standard ViT architecture, Chen et al. [76] proposed an improved ViT by introducing a Lambda layer. The Lambda layer in the ViT enhances its ability to learn positional relationships dynamically during training, unlike the original ViT which relies on static positional encodings. It also replaces the traditional self-attention mechanism with a more efficient linear function, reducing computational complexity and memory usage.

***Graphs to Represent Features:***

In malware analysis, graphs are usually generated from different ways to map structural aspects of malware samples. Among the various approaches, the one by Saracino et al. [69] generates API call graphs from the Android APKs and converts them into API call sequences. Similarly, the approach by Deng et al. [68] generates graphs from sensitive API and function calls to reflect latent behavioral patterns. The approach by Feng et al. [71] utilizes the features used in the approach by Deng et al. [68] and also incorporates Inter-component communication (ICC) features to generate graphs based embeddings. Since API calls are often used to represent malicious patterns, in real-world instances, malware often exploits existing APIs in the system to mimic the normal behavior of benign apps to evade detection. So, by incorporating ICC patterns in addition to API call graphs, malware detection becomes more robust to model evasion due to the integral nature of component interaction within an application which is less susceptible to modification or removal by malicious intentions. Besides, the limitations of using only traditional CFG to generate embeddings are also highlighted by Moon et al. [73]. They proposed a new method to incorporate edge information to preserve opcode semantics and add functional characteristics to the existing structural representations from CFG. This is achieved through a method that transforms opcodes into nodes. In addition, they utilize the DeepWalk algorithm [89] to traverse the CFG and create sequences that reflect the comprehensive relationships within the graph, including the newly added edge information. The sequences generated by DeepWalk are then embedded using a transformer encoder model, handling long sequences and capturing complex dependencies between nodes more effectively than traditional LSTM-based methods.

***Texts to Represent Features:***

Like in NLP, in malware analysis, the transformer models are applied to sequences of textual features for detecting and classifying malicious behaviors. The approaches mentioned in this section primarily use transformers to focus on generating robust representation embeddings using different types of textual features. Using opcode sequences generated from disassembly, several approaches have been proposed [57], [58], [60], [67], [85] that extract syntactic and semantic co-relation represented embeddings. Similarly, approaches have been proposed [30], [72], [75] that use API call sequences along with Manifest Files, Permissions, Services, and Intents [30], network connection data and registry access information [72] to generate robust embeddings. To detect malicious domain names, the approach by Yang et al. [80] uses n-gram location and text information, and the one by Gogoi et al. [81] that uses domain names. The approach by Ghourabi et al [64] combines file information, header information, imported and exported functions, .pcap network files, sensor data, and OS logs to generate embeddings in the IoT domain. The approach by Bellante et al. [62] uses file metadata, binary code, network behavior, system behavior, and API call sequences to generate embeddings and also addresses the problem of fewer samples by using BERT in IoT malware analysis. Li et al. [31] introduced GenTAL, a generative denoising skip-gram (aimed to predict context words given a target word) transformer [90] for binary code similarity detection by learning compact and meaningful representations from assembly codes.

> ***Takeaway:*** *Transformer models are effectively used to generate robust embeddings from diverse input types—images, text, and graphs—in malware analysis. This enables transformers to capture complex relationships, enhancing malware detection and classification across multiple data modalities.*

### 3.5 Datasets Inventory

We tried our best to access and verify all the datasets that are included in the Table 3. Android malware Genome (marked[1] in Table 3) dataset is available but the efforts to update it is stopped as mentioned in the source of the dataset. The reference provided for the ICE Dataset (marked[2]), Oliviera Dataset (marked[3]), and MalImg Dataset (marked[4]) are the references of work where the datasets

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



TABLE 3
Inventory of datasets that are used in malware analysis leveraging transformer architecture. *Labels mentioned as [1] to [7] are further explained in Section 3.5*

| Dataset Name | Dataset Contents Information | Malware Analysis End-Goal Task |
|---|---|---|
| | **Android Environment Compatible** | |
| Omnidroid [91] | 22,000 real malware and benign Android applications | Malware Detection [25] |
| Tencent Security Lab | Real-world data that contains 82,831 Android apps collected from 55 app stores/marketplaces/websites | Malware Detection [24] |
| MalNet [92] | 1,262,024 malware images extracted from real-world Android applications in AndroZoo | Malware Detection & Classification [65] |
| | **IoT Environment Compatible** | |
| IoTPOT [93], [94] | 124,799 executables captured by IoTPOT honeypot | Malware Detection [28], [58] & Explanation [58] |
| IoT-23 [95] | 20 malicious and 3 benign traffic files created by AIC Lab by Avast Software | Malware Detection [61] |
| Android Genome[1] | [96] Android malware samples and associated metadata | Malware Detection [62] |
| CICInvesAndMal2019 | [97] Data from 5000 Android samples, including 5065 benign apps and 426 malware apps | Malware Detection & Explanation [33] |
| ECU-IoHT [98] | Traces of several types of attacks launched to target medical devices | Intrusion Detection [64] |
| ToN-IoT [99] | IoT network intrusion that combines information from pcap files, Bro logs, sensor data, and OS logs | Intrusion Detection [64] |
| Edge_IIoTset [100] | Data from IoT devices such as temperature and humidity sensors, heart rate sensors, flame sensors, etc | Intrusion Detection [64] |
| EMBER [101] | Features extracted from 1.1M binary files distributed as malicious and benign | Intrusion Detection [64] |
| ICE [102][2] | Network analysis of set of ransomware attacks performed in an Integrated Clinical Environment (ICE) | Intrusion Detection [64] |
| IMG_DS [103] | 14733 benign and 2486 malicious Android IoT image representation | Malware Detection [28] |
| | **Android and IoT Environment Compatible** | |
| Drebin [104] | Malware and Benign Android apps collected from Android Malware Genome Project, Google Playstore, Chinese Markets, Russian Markets, Android websites, Malware forums and security blogs | Malware Detection [68], [69], [71] & Classification [30], [57], [69] |
| Androzoo [105] | 1046190 malware samples with 37 malware classes | Malware Detection [25], [59], [68], [70], Classification [30], [70], [74] & Explanation [70] |
| CICMalDroid2020 or Maldroid [106] | 17,341 Android samples from VirusTotal, Contagio, AMD, and MalDozer | Malware Detection [33], [68]–[71], Classification [69], [70] & Explanation [33], [70] |
| | **Windows Environment Compatible** | |
| Malshare | Self Collected data that contains 38,427,440 basic blocks of 5 to 250 instructions on each | Malware Detection & Explanation [23] |
| Catak [107] | Windows API call sequences obtained within Cuckoo Sandbox and the malware classes of 7107 records | Malware Classification [75], [77] |
| Sorel-20m [108] | 672 benign and 822 malware samples | Malware Assembly Sentence Detection [84] |
| Oliveira [109][3] | 42,797 malware and 1,079 benign API call sequences analysed dynamically | Malware Classification [77] |
| MalImg [110][4] | MalImg by Vision Research Lab contains 9339 malware byteplot images | Malware Classification [79] |
| BIG2015 | BIG2015 by Microsoft contains 21,741 malware samples images | Malware Classification [79] |
| Top-1000 [111] | Cotains over 47,000 portable executable both malware and benign imports | Malware Classification [79] |
| Speakeasy Dataset | [112] Contains 93500 behavioral JSON format malware and benign data | Malware Detection & Classification [72] |
| Avast-CTU Dataset | [113] Contains 400,000 samples in JSON format | Malware Detection & Classification [72] |
| Malicious Code [114] | Malicious Code Dataset (MCD) contains labeled 30,000 samples containing API sequences in XML format | Malware Detection & Classification [72] |
| | **Network Dataset** | |
| Alexa [115], [116][5] | Alexa contains top 1 million legitimate domain names | Malicious Domain Name Detection [80], [81] |
| 360 Network Security | [117][6] Contains malicious domain names publicly collected from different DGA families of malicious domains | Malicious Domain Name Detection [80] |
| Stratosphere IPS | [118]Contains large public network capture data in .pacp format | Malware Detection & Classification [66] |
| CICIDS2017 [119] | Contains raw capture data with the whole trace record throughout the day | Malware Detection & Classification [66] |
| Netlab Open Data[7] | [120] Contains 1 million DGA generated malicious domains along with the malware family which generated them | Malicious Domain Name Detection [81] |
| SecureNLP Challenge [121] | SubTask1, Semeval Task 8 from SecureNLP challenge dataset contains 11250 sentences from Advanced Persistent Threat (APT) reports | Malware Texts Detection [83] |
| | **Others** | |
| VirusTotal [122] | Dataset for Android and Windows Environments | Malware Detection [59], Classification [29], [77], Detection Evasion [32] |
| VirusShare [123] | PE format malicious code sample binaries - Dataset for Android, IoT and Windows Environments | Cross Architecture Malware Detection [85], Malware Detection [23], [28], [60], [63], [68], [71], Malware Classification [30], [76], [77], Explanation [23], Malware Variant Detection [26], Malware Assembly Sentence Detection [84] |
| MalwareBazaar [124] | 86225 malware binary files - Dataset for Linux and Windows Environments | Binary Code Similarity Detection [31] |
| Malpedia [125] | 3158 malware binary files - Dataset for Linux and Windows Environments | Binary Code Similarity Detection [31] |
| Kaggle Microsoft Challenge [126] | Kaggle Microsoft Malware Classification Challenge contains 10868 assembly codes and binary codes (9 malware families) - Dataset for Windows and Other Environments | Malware Classification [73], [76], [78], [79], Few Shot Malware Classification [86] |
| Self-Collected Data | Work [58] - Malware samples from Honeypot and Benign Linux ARM samples, Work [85] - Previously used by HaddadPajouh et al. [127], Work [82] - Lab-generated dataset by Dymshits et al. [128] (Stream of Vectors of integers of 300 length system call sequences), Work [23] - Benign executable collected from installation paths of software programs, Works [60], [84] - Benign files from Windows OS and Commando VM, Work [32], [63] - Benign files from Windows System files, Works [30], [68], [69] - Benign files collected from Google Play Store [129], Work [31] - Benign files collected from Software programs from Linux and Windows, Work [67] - 2793 malware families with one or more samples per family [130], Work [27] - 30,000 API Calls, including 14,302 malicious samples and 15,698 benign samples. 9000 Opcodes samples of which 3500 samples are malicious and 5500 samples are benign - Dataset for Android, Windows, IoT and Other Environments | Malware Detection [23], [27], [58], [60], [63], [67]–[69], [82], Malware Classification [30], [69], Explanation [23], [58], Cross Architecture Malware Detection [85], Malware Assembly Sentence Detection [84], Malware Detection Evasion [32], Binary Code Similarity Detection [31] |

are first used or introduced. For Alexa Dataset (marked[5]), the provided link did not work, so we found another link and added it to the reference. Furthermore, for 360 Network Security Lab Dataset (marked[6]) and Netlab Open Data Project Dataset (marked[7]), the provided links were unreachable when we tried to access them. Besides the access concerns, we observed a few datasets that are widely used, such as the Drebin, Androzoo, Maldroid, VirusTotal, VirusShare, and Kaggle Microsoft Challenge datasets. In addition, researchers have created variations of the data sets according to their requirements for the experiments.

## 4 Challenges and Future Prospects

Based on our analysis of the state of the art, we identify and categorize the limitations and future research directions into five broad categories, discussed below:

**Model and Architecture Limitations:** Studies such as [66], [78], [83], [85] present initial analyses of transformer

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



architecture and attention mechanisms. As observed, with the increase in input sequence size (e.g., high-resolution images), the multi-head attention mechanism performance decreases, hindering the overall performance, so fewer multi-heads are used. Thus, a relevant research direction is the design of techniques for optimizing the multi-head attention to improve processing and inference time for specific tasks.

**Inadequate Data and Feature Representation:** The problem of imbalanced datasets and the need for diverse malware samples [57], [73], [81], [83], [84], [86] is also a critical issue in the field of malware analysis especially given the rise of sophisticated malware types. Therefore, the generation of synthetic malware exhibiting unseen characteristics would be critical. Work by Lu et al. focuses on the construction of malware variants by inserting API calls into the assembly code [26]. However, more work is needed focusing on generating synthetic malware samples using Generative AI, GANs, Auto-encoders, etc., opting for various obfuscation techniques, unseen characteristics, and intents of malware.

**Multi-modality, Cross-Attention and RL:** In malware analysis, most of the approaches use single feature types with inadequate pre-processing techniques [28], [29], [32], [59], [65], [68], [69], [71], [73], [75], [82], [85]. Addressing such a challenge requires advanced feature engineering efforts. However, as an initial step toward such a direction, we observed a good focus on feature engineering, such as efficiently reducing the dimension of opcodes by converting them into decimal representation [57], preserving edge/opcode information (which is usually lost) [73], slicing system call sequences into various lengths of subsequences to experiment with effective sequential processing [29], and using multiple feature sets [23], [64].

Besides the existing efforts, there is still scope for future research on combining efficient and novel feature engineering techniques for multi-perspective representation of input samples. One of the directions is the design of multi-feature representation based multi-modal transformer network for malware analysis. Oliveira et al. [25] experimented with multiple features like text and images, with multi-modal Deep Neural Network, CNN, and Transformer. However, they do not provide a combined perspective, in that they use an ensemble method at the fusion layer, which does not combine the multiple perspectives from the input. It rather uses a voting mechanism for the detection decision. There are fusion techniques, like cross-attention, fused attention, etc., investigated in other domains like vision [131], [132], which can be extended to the field of malware analysis. This can help explore multiple perspectives using multi-feature representations (such as generating graph, image and text data from the same input sample or different samples) using multi-modal transformer networks.

Also, experiments combining transformer attention mechanism and reinforcement learning as suggested by Ullah et al. [33] for more robust malware detection can be an interesting avenue to explore. In malware analysis, the state representations can be a sequence of actions taken by the malware which could be features like visual, textual or graphs sequences extracted from feature types like API calls, file system changes, network activities, registry modifications, etc. The definition of actions could involve identifying malicious behavior, flagging suspicious flags, etc. On top of it, the reward function could be designed as the incentives when RL agent correctly identifies malicious intents. Studies in other domains [133], [134] have shown that this integration through memory-based reasoning and sequential behavior modeling, is effective for scalability and efficiency, which could be leveraged for malware analysis.

**Robustness and Security:** The vulnerabilities of DL models have been widely investigated because of their popularity. Similarly, there are investigations that demonstrate threats and vulnerabilities in transformer-based models. Some of the attacks are as follows: BAE [135], a black box attack to generate adversarial examples using contextual perturbations from BERT; GBDA [136], the first general purpose framework for gradient-based white-box attacks against text transformers; CWBA [137], a character-level white-box adversarial attack; Vision Transformer based attack [138], a dual attack framework, which contains a Pay No Attention (PNA) attack and a PatchOut attack, to improve the transferability of adversarial samples across different ViTs; Positional Encoding based attack [139], an adversarial attack that manipulates the model by providing it with incorrect positional information enabling an evasion attack. These attacks demonstrate the inherent vulnerabilities of transformer-based models regardless of their application, which also impacts the malware analysis domain.

In malware analysis, the vulnerabilities of transformers based on adversarial attacks have not been well analyzed; however, the need for robustness has been highlighted [23], [61], [72]. So, future research should focus on adversarial training, defensive mechanisms, evaluation of models' resilience against sophisticated attacks. In addition, the potential for malware evasion using obfuscation techniques, as well as its evolving nature [24], [58], [70] highlight the need for models that can effectively detect obfuscation techniques. Research is needed to design models capable of understanding complex obfuscation patterns.

**Deployment and Real-world Application:** Since transformer-based models are computationally intensive, there are challenges related to the deployment of these models in practical settings, particularly on-edge devices [28], [61], [64], [79]. Thus, an important direction is the optimization of transformer models for various deployment environments like cloud, edge computing, and mobile platforms. Also, creating APIs that leverage transformer models for easy integration into security systems and tools, allowing for real-time malware detection and response in applications, is an interesting avenue for future research.

## 5 Conclusions

In this paper, we provide a comprehensive systematization of knowledge (SoK) on the use of transformers in malware analysis. Our analysis shows that transformers excel in capturing intricate patterns such as spatial, temporal, structural correlations, etc. across high-dimensional data, making them well-suited for detecting and classifying malware, analyzing binary code similarity, understanding evasion techniques etc. We also discuss the challenges associated with the application of transformers in malware analysis and present an inventory of datasets used in the domain to facilitate future research and development.

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



## ACKNOWLEDGEMENT

This work is partially supported by the NSF grants 2230609, 2230610, 2416990 and 2229876.

## REFERENCES

[1] AV-TEST, "Malware Statistics," https://www.av-test.org/en/statistics/malware/, [Accessed on Feb 15, 2024].
[2] J. Jang-Jaccard and S. Nepal, "A survey of emerging threats in cybersecurity," *Journal of computer and system sciences*, 2014.
[3] K. Aryal, M. Gupta, and M. Abdelsalam, "A survey on adversarial attacks for malware analysis," *arXiv preprint*, 2021.
[4] M. Gopinath and S. C. Sethuraman, "A comprehensive survey on deep learning based malware detection techniques," *Computer Science Review*, 2023.
[5] E. Raff and et al., "Malware detection by eating a whole exe," in *Workshops at the AAAI conference on artificial intelligence*, 2018.
[6] S. Abijah Roseline and et al., "Vision-based malware detection and classification using lightweight deep learning paradigm," in *Computer Vision and Image Processing*. Springer, 2020.
[7] M. Mimura and R. Ito, "Applying NLP techniques to malware detection in a practical environment," *International Journal of Information Security*, 2022.
[8] T. K. Tran and H. Sato, "NLP-based approaches for malware classification from API sequences," in *IEEE Asia Pacific Symposium on Intelligent and Evolutionary Systems*, 2017.
[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[10] T. Lin and et al., "A survey of transformers," *AI Open*, 2022.
[11] A. M. Braşoveanu and R. Andonie, "Visualizing transformers for nlp: a brief survey," in *24th IEEE International Conference Information Visualisation*, 2020.
[12] A. Gillioz and et al., "Overview of the Transformer-based Models for NLP Tasks," in *15th IEEE Conference on Computer Science and Information Systems*, 2020.
[13] H. H. Park, Y. Vyas, and K. Shah, "Efficient classification of long documents using transformers," *arXiv preprint*, 2022.
[14] J. D. M.-W. C. Kenton and et al., "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of naacL-HLT*, 2019.
[15] J. Achiam and et al., "Gpt-4 technical report," https://cdn.openai.com/papers/gpt-4.pdf, 2023.
[16] A. Dosovitskiy and et al., "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint*, 2020.
[17] S. Khan and et al., "Transformers in vision: A survey," *ACM computing surveys*, 2022.
[18] K. Han and et al., "A survey on vision transformer," *IEEE transactions on pattern analysis and machine intelligence*, 2022.
[19] N. Carion and et al., "End-to-end object detection with transformers," in *European conference on computer vision, Springer*, 2020.
[20] A. Ramesh and et al., "Hierarchical text-conditional image generation with clip latents," *arXiv preprint*, 2022.
[21] T. Brooks and et al., "Video generation models as world simulators," https://openai.com/research/video-generation-models-as-world-simulators, 2024.
[22] L. Dong and et al., "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition," in *IEEE international conference on acoustics, speech, signal processing*, 2018.
[23] M. Q. Li and et al., "I-MAD: Interpretable malware detector using galaxy transformer," *Computers & Security*, 2021.
[24] Y. Fan and et al., "Heterogeneous temporal graph transformer: An intelligent system for evolving android malware detection," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021.
[25] A. S. de Oliveira and et al., "Hunting Android Malware Using Multimodal Deep Learning and Hybrid Analysis Data," *Brazilian Society of Computational Intelligence*, 2021.
[26] F. Lu and et al., "Research on the Construction of Malware Variant Datasets and Their Detection Method," *Applied Sciences*, 2022.
[27] Y. Li and Y. Li, "IoT Malware Threat Hunting Method Based on Improved Transformer," *International Journal of Network Security*, 2023.
[28] A. Ravi and et al., "ViT4Mal: Lightweight Vision Transformer for Malware Detection on Edge Devices," *ACM Transactions on Embedded Computing Systems*, 2023.
[29] O. Or-Meir, A. Cohen, Y. Elovici, L. Rokach, and N. Nissim, "Pay attention: Improving classification of PE malware using attention mechanisms based on system call analysis," in *IEEE International Joint Conference on Neural Networks*, 2021.
[30] A. Rahali and et al., "MalBERTv2: Code Aware BERT-Based Model for Malware Identification," *Big Data and Cognitive Computing*, 2023.
[31] L. T. Li and et al., "GenTAL: Generative Denoising Skip-gram Transformer for Unsupervised Binary Code Similarity Detection," in *International Joint Conference on Neural Networks*, 2023.
[32] J. L. Hu and et al., "Single-shot black-box adversarial attacks against malware detectors: A causal language model approach," in *IEEE International Conference on Intelligence and Security Informatics*, 2021.
[33] F. Ullah and et al., "Explainable malware detection system using transformers-based transfer learning and multi-model visual representation," *Sensors*, 2022.
[34] M. Zheng and et al., "Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware," in *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2013.
[35] H. Chen and et al., "Model Checking One Million Lines of C Code." in *NDSS*. Citeseer, 2004.
[36] A. Damodaran and et al., "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, 2017.
[37] A. Djenna and et al., "Artificial Intelligence-Based Malware Detection, Analysis, and Mitigation," *Symmetry*, 2023.
[38] L. Caviglione and et al., "Tight arms race: Overview of current malware threats and trends in their detection," *IEEE Access*, 2020.
[39] J. C. Kimmel and et al., "Recurrent neural networks based online behavioural malware detection techniques for cloud infrastructure," *IEEE Access*, 2021.
[40] J. C. Kimmell and et al., "Analyzing machine learning approaches for online malware detection in cloud," in *IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2021.
[41] D. Gibert and et al., "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network & Computer Applications*, 2020.
[42] C. Cortes and V. Vapnik, "Support Vector Networks," *Machine learning*, vol. 20, pp. 273–297, 1995.
[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, 1986.
[44] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.
[45] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 1998.
[46] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[47] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.
[48] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint*, 2014.
[49] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.
[50] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint*, 2018.
[51] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint*, 2016.
[52] V. a. a. Sanh, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *arXiv preprint*, 2019.
[53] Y. Liu and et al., "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint*, 2019.
[54] Z. Lan and et al., "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint*, 2019.
[55] J. H. Clark and et al., "Canine: Pre-training an efficient tokenization-free encoder for language representation," *Transactions of the Association for Computational Linguistics*, 2022.
[56] A. a. a. Radford, "Language models are unsupervised multitask learners," *OpenAI blog*, 2019.

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708




[57] Y.-M. Chen and et al., "Android malware detection system integrating block feature extraction and multi-head attention mechanism," in *IEEE International Computer Symposium*, 2020.

[58] X. Hu and et al., "Exploit internal structural information for IoT malware detection based on hierarchical transformer model," in *IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications*, 2020.

[59] H. Long and et al., "Detecting Android Malware Based on Dynamic Feature Sequence and Attention Mechanism," in *IEEE 5th International Conference on Cryptography, Security and Privacy*, 2021.

[60] N. Şahin, "Malware detection using transformers-based model GPT-2," Master's thesis, Middle East Technical University, 2021.

[61] W. Wangwang and et al., "Network traffic oriented malware detection in IoT (internet-of-things)," in *IEEE International Conference on Networking and Network Applications*, 2021.

[62] H. Bellante and et al., "Victory: A Framework For Fast Detection Of Iot Malware," *Webology (ISSN: 1735-188X)*, 2021.

[63] X. Qi and et al., "MDFA: A Malware Detection Framework Based on Attention Mechanism with Bi-LSTM," in *4th IEEE International Conference on Applied Machine Learning*, 2022.

[64] A. Ghourabi, "A security model based on lightgbm and transformer to protect healthcare systems from cyberattacks," *IEEE Access*, 2022.

[65] S. Seneviratne and et al., "Self-supervised vision transformers for malware detection," *IEEE Access*, 2022.

[66] O. Barut and et al., "R1dit: Privacy-preserving malware traffic classification with attention-based neural networks," *IEEE Transactions on Network and Service Management*, 2022.

[67] V. Pandya and et al., "Malware Detection through Contextualized Vector Embeddings," in *IEEE Silicon Valley Cybersecurity Conference*, 2023.

[68] X. Deng and et al., "TransMalDE: An Effective Transformer Based Hierarchical Framework for IoT Malware Detection," *IEEE Transactions on Network Science and Engineering*, 2023.

[69] A. Saracino and M. Simoni, "Graph-Based Android Malware Detection and Categorization through BERT Transformer," in *Proceedings of the 18th International Conference on Availability, Reliability and Security*, 2023.

[70] J. Jo and et al., "A Malware Detection and Extraction Method for the Related Information Using the ViT Attention Mechanism on Android Operating System," *Applied Sciences*, 2023.

[71] F. Pi and et al., "AdaTrans: An adaptive transformer for IoT Malware detection based on sensitive API call graph and inter-component communication analysis," *Journal of Intelligent & Fuzzy Systems*, 2023.

[72] D. Trizna and et al., "Nebula: Self-Attention for Dynamic Malware Analysis," *arXiv preprint*, 2023.

[73] H.-J. Moon and et al., "Directional Graph Transformer-Based Control Flow Embedding for Malware Classification," in *Intelligent Data Engineering and Automated Learning: 22nd International Conference, IDEAL, Manchester, UK*. Springer, 2021.

[74] A. Rahali and et al., "MalBERT: Malware Detection using Bidirectional Encoder Representations from Transformers," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2021.

[75] C. Li and et al., "An Efficient Transformer Encoder-Based Classification of Malware Using API Calls," in *IEEE 24th Int Conf on HPC & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application*, 2022.

[76] S. Chen and et al., "Malicious Code Family Classification Method Based on Vision Transformer," in *IEEE 10th International Conference on Information, Communication and Networks*, 2022.

[77] F. Demirkıran and et al., "An ensemble of pre-trained transformer models for imbalanced multiclass malware classification," *Computers & Security*, 2022.

[78] K.-W. Park and S.-B. Cho, "A Vision Transformer Enhanced with Patch Encoding for Malware Classification," in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2022.

[79] M. M. Belal and D. M. Sundaram, "Global-Local Attention-based Butterfly Vision Transformer for Visualization-based Malware Classification," *IEEE Access*, 2023.

[80] C. Yang and et al., "N-Trans: Parallel Detection Algorithm for DGA Domain Names," *Future Internet*, 2022.

[81] B. Gogoi and T. Ahmed, "DGA domain detection using pre-trained character based transformer models," in *IEEE Guwahati Subsection Conference*, 2023.

[82] Y. Guan and et al., "Malware system calls detection using hybrid system," in *IEEE International Systems Conference*, 2021.

[83] S. Shahid and et al., "Devising Malware Characterstics using Transformers," *arXiv preprint*, 2020.

[84] D. Demırcı, C. Acarturk *et al.*, "Static malware detection using stacked BiLSTM and GPT-2," *IEEE Access*, 2022.

[85] S. A. Hamad and et al., "BERTDeep-Ware: A Cross-architecture Malware Detection Solution for IoT Systems," in *IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications*, 2021.

[86] S.-J. Bu and S.-B. Cho, "Triplet-trained graph transformer with control flow graph for few-shot malware classification," *Information Sciences*, 2023.

[87] S.-J. Bu and et al., "A Monte Carlo search-based triplet sampling method for learning disentangled representation of impulsive noise on steering gear," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020.

[88] Q. Guo and et al., "Star-transformer," *arXiv preprint*, 2019.

[89] B. Perozzi and et al., "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.

[90] T. Mikolov and et al., "Efficient estimation of word representations in vector space," *arXiv preprint*, 2013.

[91] A. Martín and et al., "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," *Information Fusion*, 2019.

[92] S. Freitas and et al., "MalNet: A large-scale image database of malicious software," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022.

[93] Y. M. P. Pa and et al., "IoTPOT: Analysing the Rise of IoT Compromises," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[94] Y. M. Pa and et al., "IoTPOT: A novel honeypot for revealing current IoT threats," *Journal of Information Processing*, 2016.

[95] S. Garcia and et al., "IoT-23: A labeled dataset with malicious and benign IoT network traffic," http://doi.org/10.5281/zenodo.4743746, 2020.

[96] "Android Malware Genome Project — malgenomeproject.org," http://www.malgenomeproject.org/, [Accessed on Feb15, 2024].

[97] L. Taheri and et al., "Extensible android malware detection and family classification using network-flows and API-calls," in *IEEE International Carnahan Conference on Security Technology*, 2019.

[98] M. Ahmed and et al., "ECU-IoHT: A dataset for analyzing cyber-attacks in Internet of Health Things," *Ad Hoc Networks*, 2021.

[99] T. M. Booij and et al., "ToN_IoT: The role of heterogeneity and the need for standardization of features and attack types in IoT network intrusion data sets," *IEEE Internet of Things Journal*, 2021.

[100] M. A. Ferrag and et al., "Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning," *IEEE Access*, 2022.

[101] H. S. Anderson and P. Roth, "Ember: an open dataset for training static pe malware machine learning models," *arXiv preprint*, 2018.

[102] L. Fernandez Maimo and et al., "Intelligent and dynamic ransomware spread detection and mitigation in integrated clinical environments," *Sensors*, 2019.

[103] Kaggle, "IOT_Malware_dataset_for Classification," https://www.kaggle.com/datasets/anaselmasry/iot-malware, 2021, [Accessible on Feb 15, 2024].

[104] D. Arp and et al., "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, 2014.

[105] K. Allix and et al., "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th international conference on mining software repositories*, 2016.

[106] S. Mahdavifar and et al., "Dynamic android malware category classification using semi-supervised deep learning," in *IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress*, 2020.

[107] F. O. Catak and A. F. Yazı, "A benchmark API call dataset for windows PE malware classification," *arXiv preprint*, 2019.

[108] R. Harang and E. M. Rudd, "SOREL-20M: A large scale benchmark dataset for malicious PE detection," *arXiv preprint*, 2020.


Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3576708



[109] A. S. de Oliveira and et al., "Behavioral malware detection using deep graph convolutional neural networks," *Authorea Preprints*, 2023.
[110] L. Nataraj and et al., "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011.
[111] A. Oliveira, "Malware Analysis Datasets: Top-1000 PE Imports," https://dx.doi.org/10.21227/004e-v304, 2019.
[112] "Speakeasy: portable, modular, binary emulator designed to emulate Windows kernel and user mode malware," https://github.com/mandiant/speakeasy, Nov. 2021, [Accessed on Feb15, 2024].
[113] B. Bosansky, D. Kouba, O. Manhal, T. Sick, V. Lisy, J. Kroustek, and P. Somol, "Avast-CTU public CAPE dataset," *arXiv preprint*, 2022.
[114] "Malicious Code Dataset," https://github.com/kericwy1337/Datacon2019-Malicious-Code-DataSet-Stage1, Jul. 2019, [Accessed on Feb15, 2024].
[115] S. Ghodke, "Alexa Top 1 million sites," https://www.kaggle.com/datasets/cheedcheed/top1m, [Accessed on Feb15, 2024].
[116] "Alexa Top Websites," https://www.expireddomains.net/alexa-top-websites/, [Accessed on Feb15, 2024].
[117] "DGA Dataset," https://data.netlab.360.com/dga/, [Accessed by respective author on 10 December 2021].
[118] "Stratosphere Laboratory Datasets," https://www.stratosphereips.org/datasets-overview, 2015, [Accessed on Feb15, 2024].
[119] I. Sharafaldin and et al., "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, 2018.
[120] "Home - Netlab OpenData Project," https://data.netlab.360.com/feeds/dga/dga.txt, accessed by author on Jun29, 2022.
[121] P. Phandi and et al., "SemEval-2018 task 8: Semantic extraction from CybersecURity REports using natural language processing (SecureNLP)," in *Proceedings of The 12th International Workshop on Semantic Evaluation*, 2018.
[122] Chronicle LLC, "VirusTotal," https://www.virustotal.com/gui/[Accessed on Feb15, 2024].
[123] "VirusShare," https://virusshare.com/, [Accessed Feb15, 2024].
[124] "MalwareBazaar — Malware sample exchange," https://bazaar.abuse.ch/, [Accessed on Feb15, 2024].
[125] F. FKIE, "Malpedia," https://malpedia.caad.fkie.fraunhofer.de/, [Accessed on Feb15, 2024].
[126] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv preprint*, 2018.
[127] H. HaddadPajouh and et al., "A deep recurrent neural network based approach for internet of things malware threat hunting," *Future Generation Computer Systems*, 2018.
[128] M. Dymshits and et al., "Process monitoring on sequences of system call count vectors," in *IEEE International Carnahan Conference on Security Technology*, 2017.
[129] "Android Apps on Google Play," https://play.google.com/store, [Accessed on Feb15, 2024].
[130] S. Kim, "PE Header Analysis for Malware Detection," https://scholarworks.sjsu.edu/etd_projects/624, 2018.
[131] A. Nagrani and Y. et al., "Attention bottlenecks for multimodal fusion," *Advances in neural information processing systems*, 2021.
[132] J. Zhang, Y. Xie, W. Ding, and Z. Wang, "Cross on cross attention: Deep fusion transformer for image captioning," *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.
[133] L. Chen and et al., "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.
[134] C. Chen, Y.-F. Wu, J. Yoon, and S. Ahn, "Transdreamer: Reinforcement learning with transformer world models," *arXiv preprint arXiv:2202.09481*, 2022.
[135] S. Garg and G. Ramakrishnan, "Bae: Bert-based adversarial examples for text classification," *arXiv preprint*, 2020.
[136] C. Guo and et al., "Gradient-based adversarial attacks against text transformers," *arXiv preprint arXiv:2104.13733*, 2021.
[137] A. Liu and et al., "Character-level white-box adversarial attacks against transformers via attachable subwords substitution," *arXiv preprint*, 2022.
[138] Z. Wei and et al., "Towards transferable adversarial attacks on vision transformers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.
[139] S. Gao and et al., "Pe-attack: On the universal positional embedding vulnerability in transformer-based models," *IEEE Transactions on Information Forensics and Security*, 2024.

**Pradip Kunwar** is currently pursuing Ph.D. in Computer Science at Tennessee Tech University, Cookeville, TN, USA. He received his B.Tech degree in Electronics and Communication from NIT Rourkela, India. His current research interests are Transformers in Malware Analysis, Security of/for LLMs, PEFT Methods, Mixture of Experts Architectures etc. He is interested in researching the underlying vulnerabilities of AI systems and making them more robust against adversarial attacks.

**Kshitiz Aryal** is currently pursuing a Ph.D. degree with the Department of Computer Science, Tennessee Technological University, Cookeville, TN, USA. He also received his M.S. in Computer Science from Tennessee Tech and B.S. in ECE from Tribhuvan University, Nepal. His current research interests include adversarial attacks/defense, malware analysis, AI security, security of AI, explainable AI, and data science.

**Maanak Gupta** (Senior Member, IEEE) is an Assistant Professor in Computer Science at Tennessee Technological University, Cookeville, USA. He received M.S. and Ph.D. in Computer Science from the University of Texas at San Antonio (UTSA) and has also worked as a postdoctoral fellow at the Institute for Cyber Security (ICS) at UTSA. His primary area of research includes security and privacy in cyber space focused in studying foundational aspects of access control, malware analysis, AI and machine learning assisted cyber security, and their applications in technologies including cyber physical systems, cloud computing, IoT and Big Data. His research has been funded by the US National Science Foundation (NSF), NASA, and US Department of Defense (DoD) among others.

**Mahmoud Abdelsalam** received the M.Sc. and Ph.D. degrees from the University of Texas at San Antonio (UTSA), in 2017 and 2018. He worked as Postdoctoral Research Fellow with the Institute for Cyber Security (ICS), UTSA, and as an Assistant Professor with the Department of Computer Science, Manhattan College. He is currently working as an Assistant Professor with the Department of Computer Science, North Carolina A&T State University. His research interests include computer systems security, anomaly and malware detection, cloud computing security and monitoring, cyber-physical systems security, and applied ML.

**Elisa Bertino** is Samuel Conte professor of Computer Science at Purdue University. Before Purdue, she was a professor and department head at the Department of Computer Science and Communication of the University of Milan. She has been a visiting researcher at the IBM Research Laboratory in San Jose (now Almaden), at Rutgers University, at Telcordia Technologies. She also held visiting professor positions at the Singapore National University and the Singapore Management University. Her recent research focuses on security and privacy of cellular networks and IoT systems, and on edge analytics for cybersecurity. Elisa Bertino is a Fellow member of IEEE, ACM, and AAAS. She received the 2002 IEEE Computer Society Technical Achievement Award for "For outstanding contributions to database systems and database security and advanced data management systems", the 2005 IEEE Computer Society Tsutomu Kanai Award for "Pioneering and innovative research contributions to secure distributed systems", the 2019-2020 ACM Athena Lecturer Award, and the 2021 IEEE 2021 Innovation in Societal Infrastructure Award. She received an Honorary Doctorate from Aalborg University in 2021 and an Honorary Research Doctorate in CS from the University of Salerno in 2023. She is currently serving as ACM Vice-president.

Authorized licensed use limited to: Tennessee Technological University. Downloaded on July 29,2025 at 03:21:34 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.