G-FLEX: A Graph-based and Fine-Tuned Transformer Framework with Explainable AI for Fileless Malware Detection

Bao Pham-Thai$^{1,2[0009-0009-7841-3300]}$, Nghi Hoang Khoa$^{1,2[0000-0001-6418-4169]}$, Ngo Duc Hoang Son$^{1,2[0009-0000-8101-7311]}$, Khanh Ho-Vi$^{1,2[0009-0009-6307-8458]}$, and Phan The Duy$^{1,2[0000-0002-5945-3712]}$ ✉

^{1} Infomation Security Lab, University of Information Technology, Ho Chi Minh City, Vietnam
^{2} Vietnam National University, Ho Chi Minh City, Vietnam
`21520156@gm.uit.edu.vn`, `{sonndh, khoanh}@uit.edu.vn`, `22520633@gm.uit.edu.vn`, ✉`duypt@uit.edu.vn`

Abstract. Fileless malware presents a growing cybersecurity challenge due to its stealthy, memory-resident behavior that evades traditional detection methods. In this paper, we propose G-FLEX (**G**raph-based **F**ileless ma**L**war**E** e**X**planation), a novel framework that combines graph representations, fine-tuned transformers, and explainable AI (XAI) to detect and interpret fileless malware threats. Our method begins with the transformation of source code into Abstract Syntax Trees (ASTs) and Control Flow Graphs (CFGs) to structure code behavior and eliminate irrelevant noise. To capture deep semantic and syntactic features, we fine-tune state-of-the-art language models such as BERT, Electra, and CodeBERT, and incorporate Hierarchical Transformers to model multi-level code dependencies. GraphCodeBERT is further employed to generate enriched graph-aware embeddings. For interpretability, we apply XAI techniques, notably SHAP, to identify influential features such as function calls, AST node patterns, and CFG structures that drive the model's decisions. Experimental results indicate that G-FLEX achieves over 99% accuracy on both standard and obfuscated fileless malware samples, significantly outperforming existing approaches. These results demonstrate the real-world potential of G-FLEX in enhancing fileless malware detection while providing transparency and actionable insights for cybersecurity analysts.

Keywords: PowerShell script · Control Flow Graph · Fine-tune · Hierarchical Transformer · Explainable AI.

1 Introduction

In 2024, APT attacks increased by 58%, targeting government, energy, finance, military, and technology sectors. In the Asia-Pacific region, APT10 focused on

IT and manufacturing, Lazarus on cryptocurrency theft, and DarkPink on military and government infiltration [7]. Globally, fraud rose by 22%, with Europe's financial sector accounting for 34% of scams, primarily investment, romance, and tech support fraud [19]. Fileless malware leverages legitimate Windows tools like PowerShell, VBA, and JavaScript to evade detection by blending with normal system activity. Unlike traditional malware, it operates entirely in memory, leaving no file system artifacts, complicating forensic analysis and signature-based detection. Once attackers obtain credentials, they escalate privileges, move laterally, and execute payloads via system-native tools [3,21]. Detecting APTs and fileless malware thus requires advanced analysis of PowerShell scripts and in-memory execution patterns.

Traditionally, script-based fileless malware can be inspected by analyzing its source code, as it is executed directly and does not undergo compilation. Previous studies have generally adopted one of the two fundamental approaches to analyzing source code. The first approach examines raw source code to detect patterns or issues [15], while the second leverages AST for a structured, hierarchical code representation [17].

Both approaches have advanced significantly with the development and adoption of AI. In source code content analysis approach, Transformer-based NLP models have demonstrated effectiveness, with fine-tuning significantly enhancing task-specific learning [2, 13] for detecting malicious PowerShell scripts. Hierarchical Transformers further refine embeddings, leading to improved performance [16]. Similarly, AST-based methods leverage structured code representation to facilitate ML/DL-driven detection [14,20]. Key features such as code size, string patterns, and entropy analysis play a crucial role in identifying obfuscation and malicious intent [5, 10].

Despite these improvements, existing studies on fileless malware detection predominantly rely on a single analytical approach, which inherently restricts the diversity of features and the depth of information that can be leveraged for effective detection. This reliance creates blind spots in analysis and undermines the ability to generalize across complex attack scenarios. For instance, source code analysis effectively captures the textual semantics of malicious programs but falls short in revealing execution dynamics, making it insufficient for understanding behavioral intent. Conversely, AST analysis offers valuable insights into program structure and syntactic relationships, yet lacks the contextual depth provided by textual content. Integrating both approaches presents a promising solution, as it enables the extraction of complementary features—textual semantics from code and structural dependencies from AST. This synergy not only enriches feature representation but also strengthens detection accuracy and robustness, ultimately addressing the limitations of single-modality analysis in fileless malware detection.

On the other side, the adoption of AI is expanding across various domains, including security, with XAI playing a critical role in clarifying model decisions. As AI transitions from a supportive tool to a decision-making system, transparency becomes essential for fostering user trust [4]. In the context of AI-driven fileless malware detection, a significant challenge lies in the lack of interpretability, which makes it difficult for users and security analysts to understand how models identify threats. Without clear explanations, both trust in the system and its overall effectiveness may be compromised. Looking ahead, XAI is expected to play an increasingly vital role in visualizing model decisions, highlighting critical input features, and assisting analysts in prioritizing relevant information. This, in turn, will enhance the detection and comprehension of malicious scripts, ultimately strengthening cybersecurity measures [18].

The key contributions of this study are summarized as follows:

- **Fileless Malware Analysis:** We convert source code into ASTs to obtain structured insights that facilitate statistical analysis. This representation enhances feature extraction and contributes to more accurate classification of malicious scripts.
- **Fine-tuning with Language Models and Hierarchical Transformers:** By leveraging pre-trained language models (LMs) and incorporating Hierarchical Transformers during fine-tuning, our approach captures both long-range dependencies and execution patterns. This enables the detection of obfuscated and sophisticated fileless malware attacks with improved accuracy.
- **CFG for Function Relationships:** Source code is further transformed into CFGs, which explicitly model function relationships and execution flows. This structured representation improves both interpretability and detection performance by highlighting functional dependencies.
- **XAI for Feature Importance:** To enhance interpretability, XAI techniques are employed to identify and highlight key AST-based features. This not only improves classification but also provides valuable insights for distinguishing between benign and malicious scripts, thereby supporting malware analysts in understanding model decisions.

The remainder of this paper is structured as follows. **Section 2** reviews existing methods that serve as a foundation for developing the proposed approach, addressing gaps and improvements. **Section 3** details the theoretical background, methodology, and model implementation. **Section 4** presents parameter configurations, experimental setups, and results, highlighting the effectiveness of the proposed method. Finally, **Section 5** summarizes key findings, discusses strengths and limitations, and outlines future research directions for improving PowerShell script classification.

2 Related work

Leveraging extracted information, AI-based approaches integrate directly extracted data as input for classification models. [20] represents PowerShell scripts using an AST, extracting structural information for ML and DL models, achieving promising results. [22] expands this by incorporating multiple sources, including character extraction via frequency analysis, token extraction through statistical methods, AST-based semantic representation, and knowledge graph-based feature extraction, where a multimodal feature fusion model aggregates these extracted features for classification. Similarly, [5] combines feature extraction from source code and AST analysis, using a Random Forest (RF) model for classification. However, existing methods still struggle to detect obfuscated PowerShell scripts as they rely on signature-based detection or shallow code analysis, which are ineffective against advanced evasion techniques such as syntax modification, misleading structures, and encoding. While AST-based methods are effective in capturing program structure, they cannot fully preserve execution logic, limiting their ability to expose hidden behaviors in obfuscated code. To address this limitation, our approach highlights the significance of incorporating CFGs, which represent the execution logic and function relationships within scripts, thereby offering a more comprehensive view of program semantics. Unlike [20], which only relies on AST, our method combines AST and CFG, enabling deeper semantic representation and more robust detection of obfuscated PowerShell code. By transforming extracted information into CFGs, our proposed model not only enhances the preservation of execution logic but also strengthens the capability of detecting evasive and stealthy malicious behaviors, marking a notable advancement over prior AST-only approaches.

Embedding plays a crucial role in enabling models to understand data context in specific tasks. [9] employs FastText, where PowerShell scripts are preprocessed and tokenized, leveraging FastText's ability to capture contextual meaning, particularly for long script data. Meanwhile, Bidirectional Encoder Representations from Transformers (BERT), built upon the Transformer architecture, enhances bidirectional learning and contextual understanding, making it widely adopted in various applications [12]. However, the effectiveness of NLP models depends heavily on the data type, and fine-tuning is crucial for improving contextual representation and downstream performance. To address this, [13] fine-tunes BERT for malware detection using API Sequence Functions, achieving superior results. This highlights that leveraging LMs and fine-tuning them for malware-related tasks is highly effective in capturing execution semantics and detecting malicious intent, even when scripts are obfuscated or highly complex. Nevertheless, standard NLP models often struggle with long-range dependencies and semantic coherence, particularly in lengthy PowerShell scripts. To overcome this limitation, Hierarchical Transformers have been introduced to aggregate information from fine-tuned embeddings, enabling a more robust and scalable representation of long-sequence data [11], which is especially valuable in our work on handling obfuscated scripts in practical malware detection scenarios.

AI-powered research has leveraged algorithms to clarify model decisions and identify key performance-driving features. In malware detection, [6] addresses this challenge using deep learning models, specifically LSTM and GRU, with API Sequence Function as input. Various XAI techniques, including LIME, SHAP, LRP, and Attention mechanisms, are applied across evaluation scenarios assessing sensitivity to input perturbation, execution time, key feature impact, and prediction stability. Results highlight SHAP as particularly effective for

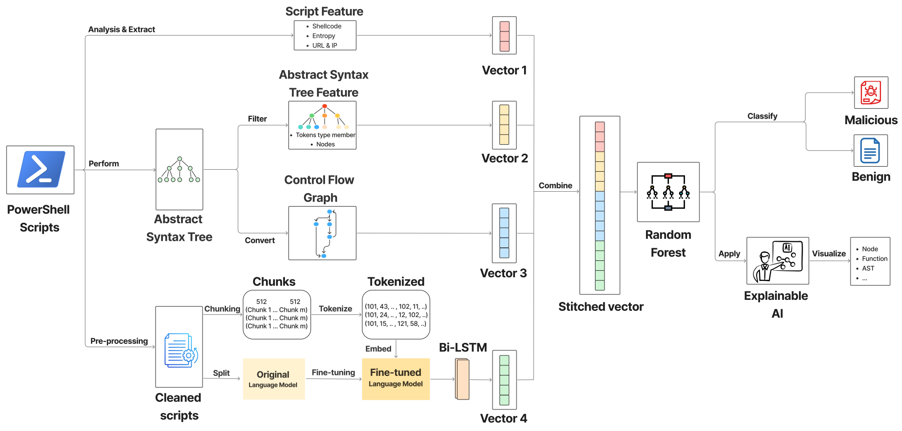

Fig. 1. Detailed system design of G-FLEX

malware detection explainability. Similarly, [1] employs an ML-based approach combining static feature analysis with SHAP to quantify feature importance and improve interpretability. Despite these advances, malware detection models still lack transparency, hindering security experts' ability to analyze and respond to emerging threats. In this context, our proposed approach emphasizes the central role of XAI by integrating SHAP to elucidate how extracted and code-represented information influences model decisions. This integration not only strengthens interpretability but also bridges the gap between predictive performance and practical usability, enabling security experts to better understand, validate, and trust the system's outcomes. Consequently, XAI becomes a crucial component of our framework, ensuring that the model's predictions are not only accurate but also transparent and actionable in real-world malware analysis scenarios.

3 Methodology

This section presents our proposed approach of **G**raph-based **F**ileless ma**L**war**E** e**X**planation (G-FLEX) with Fine-tuned Transformer and XAI to detect the fileless malware threats. The proposed model, detailed in **Fig. 1**, incorporates three key information sources for classification. It extracts features directly from the source code, represents the code as an AST to analyze structural patterns, and derives a CFG from the AST to highlight function usage. A fine-tuned LMs combined with Hierarchical Transformers, specifically Bidirectional Long Short-Term Memory (Bi-LSTM), captures and integrates contextual information. The classification model employs RF using aggregated features from these processes. Additionally, XAI techniques are applied to visualize model decisions and emphasize critical input features.

3.1 Analysis and Extract from Script

By examining script segments, it is evident that attackers frequently rely on recurring techniques that can be leveraged for identification and classification. PowerShell scripts used in malicious activities often embed shellcode, which not only bypasses traditional antivirus systems but also reflects sophisticated attack strategies such as process injection and in-memory code execution. Consequently, detecting the presence of shellcode serves as a strong indicator of malicious intent. In addition, certain scripts employ simple yet highly evasive mechanisms, such as downloading payloads from the Internet rather than embedding them directly, or deliberately circumventing security tools, both of which significantly increase the difficulty of detection. Entropy analysis thus becomes an essential feature, as it effectively distinguishes obfuscated or dynamically generated code.

Moreover, many fileless malware campaigns rely on pre-configured payloads, with scripts functioning primarily as lightweight downloaders that establish connections to remote command-and-control (C2) servers for execution. In this context, the detection of URLs and IP addresses within scripts provides valuable contextual information that strengthens the identification of malicious behaviors. The outcome of this feature extraction process is a three-dimensional vector representation that captures both the structural and semantic properties of PowerShell scripts. This multidimensional feature space offers a more comprehensive characterization of attack techniques, thereby enabling more robust and resilient detection mechanisms against fileless malware.

3.2 Extract from Abstract Syntax Tree

Feature extraction using the AST structure not only refines the source code but also preserves the relationships between its components. Functions play a key role in identifying malicious scripts. One approach involves analyzing commonly used functions in malware and applying one-hot encoding to create feature vectors based on their presence. To achieve this, AST token structures are examined, specifically focusing on “member" type tokens that correspond to functions. The System.Management.Automation.PSParser::Tokenize method helps extract these tokens and identify function calls. By analyzing a dataset of malicious scripts, researchers rank functions by frequency, highlighting 33 key functions that frequently appear in malware.

During analysis, the information of the AST nodes is also considered, as their number is limited. The use of this information contributes to an effective classification. Statistical analysis reveals an uneven distribution of nodes between benign and malicious scripts, with 23 nodes demonstrating distinct significance. Among them, 19 nodes predominantly appear in benign scripts, while 4 are more common in malware. The classification method incorporates one-hot encoding based on the presence of these 23 nodes in each script within the dataset.
As a result, the outcome of this feature extraction step is a 56-dimensional vector representation, combining function-based and AST-based indicators. This structured vector captures both syntactic and semantic properties of PowerShell

scripts, enhancing the effectiveness of classification models in distinguishing between benign and malicious behavior.

3.3 Extract from Control Flow Graph

To achieve a comprehensive analysis of source code, particularly at the function level, the process begins by converting the code into an AST, from which relevant functions are identified and extracted. These functions are then transformed into a CFG, providing a structured representation that clarifies functional relationships and improves interpretability of execution flow. In this representation, nodes in the CFG correspond to functions derived from the AST, enabling a clear mapping between syntax and control dependencies. To further enhance classification performance, the graph-based representations are embedded using the GraphCodeBERT model, which effectively captures both structural and contextual properties of the code. The resulting 768-dimensional vector encodes syntactic features, semantic dependencies, and functional interactions, thereby offering a richer representation of PowerShell scripts. This integration of AST, CFG, and GraphCodeBERT embeddings facilitates more precise classification by leveraging both structural hierarchy and semantic context inherent in source code.

3.4 Fine-tuning with Language Model and Hierarchical Transformers

To ensure the source code contains only relevant information for effective classification and reduce noise during embedding, a preprocessing pipeline is applied. This includes converting the script to lowercase for consistency, removing comments to focus on critical information, and normalizing the text by replacing numbers with '*', removing special characters, and standardizing spacing to obscure sensitive data while preserving the semantic structure.

Fine-tuning of the LM is employed to effectively capture both contextual and structural information while adapting model weights to PowerShell script data. After fine-tuning, the model tokenizes the script to learn relationships among code components, thereby enhancing its ability to represent semantic and syntactic dependencies. To further improve representation quality, pre-processing through chunking is applied, which reduces sequence length while preserving logical structure. In addition, an overlapping strategy is introduced to strengthen connections between adjacent chunks, ensuring greater cohesion and continuity in the learned representations. This combined process enhances the model's capacity to analyze complex scripts and contributes to more accurate detection of fileless malware.

$$T = w_1, w_2, \ldots, w_n \tag{1}$$

The data preparation pipeline for fine-tuning is carefully designed to ensure that contextual and structural information is effectively embedded. As shown in

Eq. 1, each PowerShell script is first tokenized into a sequence of words, where w_i represents the i-th token. This representation serves as the foundation for subsequent processing and embedding.

$$C_k = w_{(k-1)(512-50)+1}, \ldots, w_{(k-1)(512-50)+512} \quad \text{where} \quad k = 1, 2, \ldots, K \tag{2}$$

To capture long-range dependencies while maintaining manageable sequence lengths, an overlapping chunking strategy is applied. Each chunk contains 512 tokens with a 50-token overlap, as defined in **Eq. 2**. This approach reduces information loss at chunk boundaries and ensures that critical contextual cues are preserved across consecutive segments, thereby enhancing semantic continuity in the representation.

$$E_k = \text{Tokenizer}(C_k) = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m) \tag{3}$$

Each chunk is then processed through a tokenizer, as expressed in **Eq. 3**, where $\mathbf{x}_i$ denotes the embedding of the i-th token and m corresponds to the tokenized length of the chunk. This step allows the model to convert raw script tokens into dense vector representations, encoding both syntactic and semantic relationships essential for accurate malware classification.

$$\text{pad}(E_k) = \begin{cases} E_k, & \text{if } |E_k| = 512 \\ (E_k, 0, 0, \ldots, 0), & \text{if } |E_k| < 512 \end{cases} \tag{4}$$

To ensure uniform input across all samples, zero-padding is applied as described in **Eq. 4**. This step maintains consistent vector length without altering the semantic contribution of existing tokens, enabling efficient batching and model training.

Finally, the output of this pipeline is a 768-dimensional embedding vector per chunk, which captures both local semantic nuances and global contextual dependencies. This multidimensional representation enhances the ability of the fine-tuned models to detect subtle malicious patterns, thereby improving robustness in distinguishing between benign and malicious PowerShell scripts.

3.5 Model Classification

The classification model employed in this study is RF, a widely used traditional machine learning algorithm known for its robustness and strong performance in classification tasks. In our approach, the extracted information is systematically transformed into vectorized feature representations, which are subsequently combined and provided as input to the RF model. This process enables the model to capture diverse feature interactions and leverage ensemble learning to improve predictive accuracy and reliability in detecting malicious scripts.

3.6 Explainable AI

Our study integrates multiple types of information to improve analysis and reduce misclassification by ensuring a comprehensive evaluation of input data. Nevertheless, identifying which features exert the greatest influence on model predictions remains a significant challenge. Enhancing feature interpretability is particularly valuable, as it enables malware analysts to better understand and assess the reasoning behind the model's decisions. To address this issue, we employ the SHAP algorithm, which quantifies feature importance and provides explainability by revealing how individual features contribute to classification outcomes. This not only strengthens confidence in the model but also offers actionable insights into the behavioral characteristics of malicious scripts.

4 Implementation and Experiments

4.1 Dataset and Hyperparameter

Our dataset consists of original PowerShell scripts that have been systematically transformed into graph representations, and it is publicly accessible on GitHub[3]. Specifically, the dataset comprises 4,316 benign scripts, 4,202 malware scripts, and 4,202 obfuscated malware scripts. This balanced composition ensures a representative distribution of benign, malicious, and obfuscated samples, thereby supporting rigorous evaluation of detection models under realistic conditions. The inclusion of obfuscated malware is particularly significant, as it reflects adversarial strategies commonly employed to evade detection, making the dataset well-suited for advancing research in robust and resilient fileless malware detection.

Data transformation into AST and CFG structures was performed in an Ubuntu 20.04 environment with 16GB RAM and 1TB HDD, utilizing the pwsh tool for PowerShell execution.

For fine-tuning, source code is segmented into 512-word chunks, with up to three used as model input, padded if necessary. The dataset is split into 60% training, 20% validation, and 20% testing. Key hyperparameters include a 5e-5 learning rate and CrossEntropyLoss. Experiments employ Base versions of BERT, Electra, and CodeBERT.

BERT is selected for its strong bidirectional contextual encoding, which enables the model to capture both semantic meaning and syntactic relationships in PowerShell scripts. Electra is incorporated because of its sample-efficient pre-training mechanism, which improves robustness and accelerates convergence, making it particularly suitable for datasets that are less extensive. CodeBERT is included as it is pre-trained specifically on programming languages and natural language pairs, allowing it to better capture structural and semantic features unique to source code. By leveraging these models, our approach balances

[3] `https://github.com/das-lab/mpsd`

general-purpose language understanding with domain-specific code representation, thereby improving detection accuracy and reducing the likelihood of misclassification in fileless malware analysis.

For PowerShell script classification, dataset is split 70% for training and 30% for testing, using a RF model with 100 estimators.

4.2 Metric

This study evaluates model effectiveness across four critical dimensions: performance, generalization, interpretability, and efficiency. Accuracy serves as a measure of the overall classification capability, reflecting how well the model distinguishes between benign, malicious, and obfuscated scripts. Precision quantifies the proportion of correctly identified malware among all predicted malware, highlighting the reliability of positive detections. Recall, also known as the True Positive Rate (TPR), measures the model's ability to identify actual malware instances, which is particularly vital in minimizing undetected threats. The F1-score provides a harmonic balance between precision and recall, offering a robust metric when dealing with imbalanced datasets.

In addition to these standard metrics, the True Negative Rate (TNR) is considered to evaluate the accurate recognition of benign scripts, ensuring the system does not excessively penalize legitimate code. Conversely, the False Negative Rate (FNR) captures instances where malware remains undetected, representing a critical security concern due to potential exploitation risks. Finally, the False Positive Rate (FPR) reflects benign scripts misclassified as malicious, which, although less harmful than missed malware, can still undermine usability by generating unnecessary alerts. Together, these metrics provide a comprehensive perspective on the strengths and limitations of the proposed model in real-world malware detection scenarios.

4.3 Ablation Study: Comparative Analysis of Model Components to Identify the Optimal Configuration

In the proposed model, LMs and source code representation as a CFG are combined with feature extraction from the source code and AST. To comprehensively compare the effectiveness of different extraction methods, each LMs model and CFG representation is evaluated separately. We define representative values, where Features denote extracted information from source code and AST. CodeBERT, Electra, and BERT represent LMs for source code embedding, while Graph indicates whether CFG is utilized. **Table 1** presents detailed results using original malware script data, showing that the Electra model, enhanced with Hierarchical Transformers during fine-tuning and incorporating AST and source code features, achieves the highest performance.

This outcome can be attributed to Electra's discriminative pre-training approach, which focuses on detecting replaced tokens rather than merely predicting masked words. Such a mechanism enables Electra to capture subtle contextual deviations in malicious PowerShell scripts more effectively than its counterparts,

Table 1. Evaluating Component Contributions in Model Performance on Malicious Script and Mixed Script

Script kind	Metric	Feature CodeBert	Feature Bert	Feature Electra	Feature	Feature CodeBert Graph	Feature Bert Graph	Feature Electra Graph	Feature Graph
Malicious	Accuracy	0,9871	0,9871	**0,9912**	0,9877	0,9888	0,9894	0,9888	0,9871
	Precision	0,9940	0,9940	**0,9952**	0,9904	**0,9952**	**0,9952**	0,9940	0,9904
	Recall - TPR	0,9798	0,9798	**0,9869**	0,9845	0,9822	0,9834	0,9834	0,9834
	F1-score	0,9868	0,9868	**0,9910**	0,9875	0,9886	0,9892	0,9886	0,9869
	TNR	0,9942	0,9942	**0,9954**	0,9907	**0,9954**	**0,9954**	0,9942	0,9907
	FPR	0,0058	0,0058	**0,0046**	0,0093	**0,0046**	**0,0046**	0,0058	0,0093
	FNR	0,0202	0,0202	**0,0131**	0,0155	0,0178	0,0166	0,0166	0,0166
Malicious mixed	Accuracy	0,9806	0,9771	0,9871	0,9888	0,9695	0,9712	0,9730	**0,9906**
	Precision	0,9891	0,9855	0,9916	0,9928	0,9805	0,9806	0,9842	**0,9940**
	Recall - TPR	0,9715	0,9679	0,9822	0,9845	0,9572	0,9608	0,9608	**0,9869**
	F1-score	0,9802	0,9766	0,9869	0,9887	0,9687	0,9706	0,9723	**0,9905**
	TNR	0,9896	0,9861	0,9919	0,9930	0,9815	0,9815	0,9849	**0,9942**
	FPR	0,0104	0,0139	0,0081	0,0070	0,0185	0,0185	0,0151	**0,0058**
	FNR	0,0285	0,0321	0,0178	0,0155	0,0428	0,0392	0,0392	**0,0131**

making it particularly well-suited for precise detection in pure malicious script scenarios. The integration of AST and source code features further enriches structural and textual representations, yielding superior classification results.

In contrast, for the mixed-malicious setting—where benign and obfuscated malicious samples coexist—the integration of CFG representations with extracted features delivers the best performance. Unlike purely textual or structural analysis, CFG explicitly models control-flow dependencies, allowing the system to better capture execution semantics and function interactions. This proves especially valuable in detecting obfuscated or evasive malware, where surface-level tokens may resemble benign scripts but underlying execution paths reveal malicious intent. Consequently, the CFG-based approach strengthens generalization and robustness against harder-to-detect mixed samples.

Taken together, these findings demonstrate that our proposed model achieves significant improvements over prior studies. While existing research predominantly relies on AST-based analysis to enrich contextual information, our approach pioneers the integration of CFG with source code and AST features, thereby addressing limitations of prior methods. The results confirm that different representations excel under different threat conditions—Electra with feature integration in malicious-only detection, and CFG with feature integration in mixed-malicious detection—validating the adaptability and advancement of our framework for fileless malware detection.

4.4 Comparative Performance Evaluation of the Proposed Model Against Existing Approaches

To highlight the results and provide an objective evaluation of the proposed model compared to existing approaches based on accuracy, several prior methods

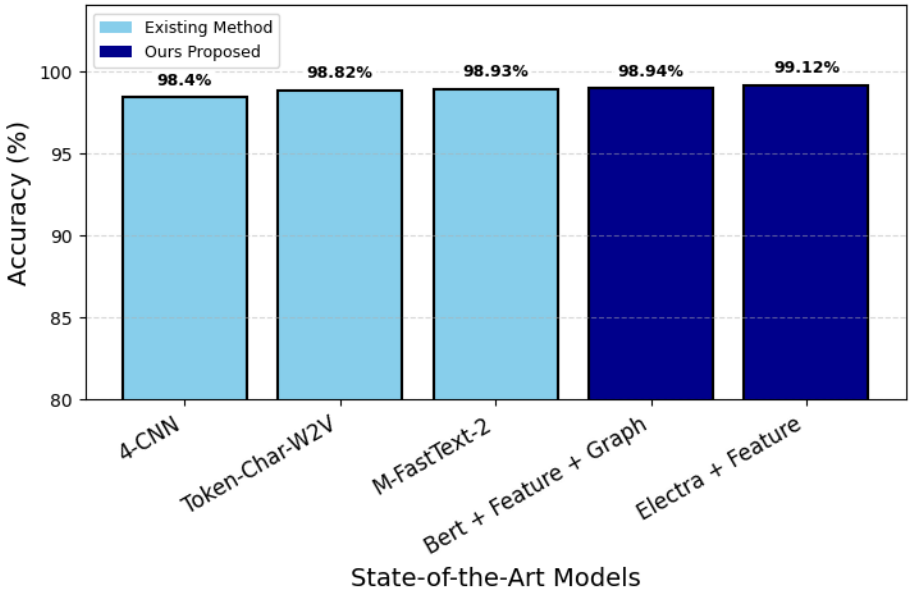

Fig. 2. Benchmarking the Proposed Model Against Existing Techniques on Original Malicious Script

are considered. Specifically, [5] extracts source code features, transforms them into AST representations, and classifies using RF. [8] employs a CNN-based classification model, while [9] utilizes token and character embeddings with a Bi-LSTM classifier. For each dataset, models with higher accuracy than existing approaches are selected, ensuring a comprehensive comparison of the proposed model against prior works.

On the dataset consisting solely of common malware scripts, both variants of the proposed method consistently achieved superior performance compared with baseline techniques, as illustrated in **Fig. 2**. By integrating NLP models with a fine-tuning strategy, the model is able to capture richer contextual dependencies and aggregate them effectively, which enhances its predictive capability. Furthermore, the incorporation of a Bi-LSTM layer contributes additional gains by modeling sequential relationships, thereby reinforcing the robustness of the overall approach.

The generalizability of the proposed approach is further demonstrated on a dataset containing mixed malicious scripts. As shown in **Fig. 3**, graph-based representations yield clear advantages, with the CFG effectively capturing the structural relationships and prevalent function usage patterns embedded within malicious code. This structural modeling allows the proposed approach to surpass existing baselines, confirming the critical role of CFG-based representations in accurately characterizing diverse malware behaviors.

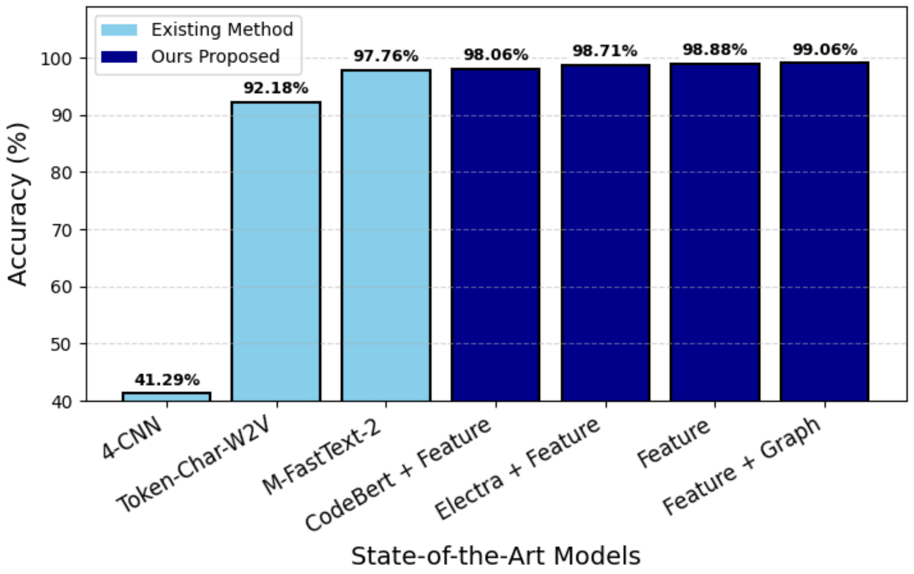

Fig. 3. Benchmarking the Proposed Model Against Existing Techniques on Mixed Malicious Script

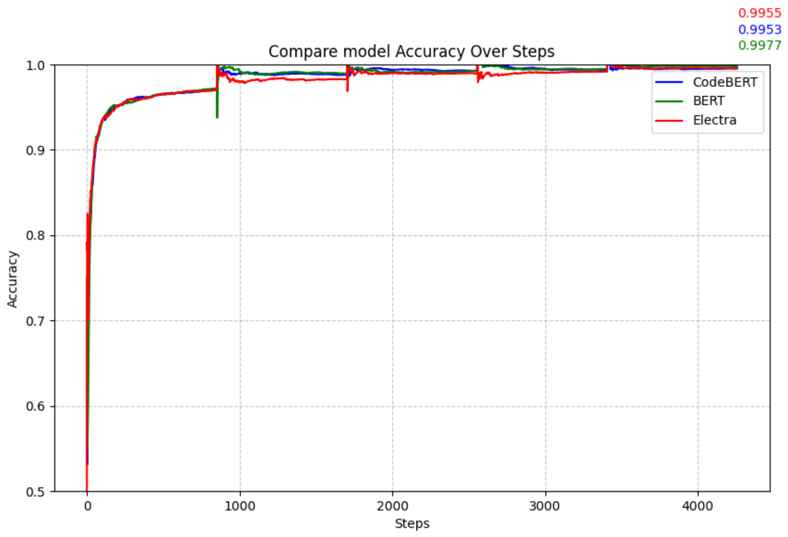

Fig. 4. Accuracy of Fine-tune Process

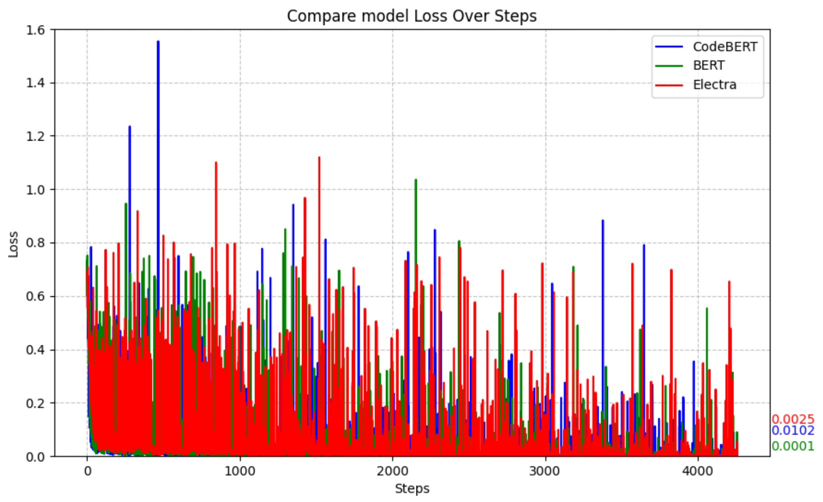

Fig. 5. Loss of Fine-tune Process

4.5 Fine-Tuning Performance Analysis

The fine-tuning analysis further validates the effectiveness of incorporating NLP-based models into the detection framework. As presented in **Fig. 4**, all models demonstrate steady learning progress, with BERT achieving the highest accuracy and exhibiting the fastest convergence. Similarly, **Fig. 5** highlights BERT's efficiency, reflected in its lowest loss trajectory. Electra and CodeBERT also show competitive results, confirming the strength of transformer-based embeddings in malware detection.

The process of fine-tuning plays a pivotal role in adapting large pre-trained language models to the specific domain of fileless malware detection. By aligning the generic linguistic knowledge of these models with domain-specific data, fine-tuning enables them to capture subtle patterns and context-sensitive relationships within malicious code that are otherwise overlooked in general training. This domain adaptation not only sharpens the models' discriminative capabilities but also reduces overfitting risks by ensuring that the representations remain contextually relevant. Consequently, the models can better differentiate between benign and malicious behaviors even in highly obfuscated or unconventional attack scenarios.

Taken together, these findings indicate that the fine-tuning process not only enhances model performance but also enriches contextual representations, which are critical for robust fileless malware detection. Ultimately, the integration of fine-tuned transformer-based embeddings establishes a strong foundation for advancing detection frameworks, bridging the gap between linguistic understanding and structural analysis of malicious code.

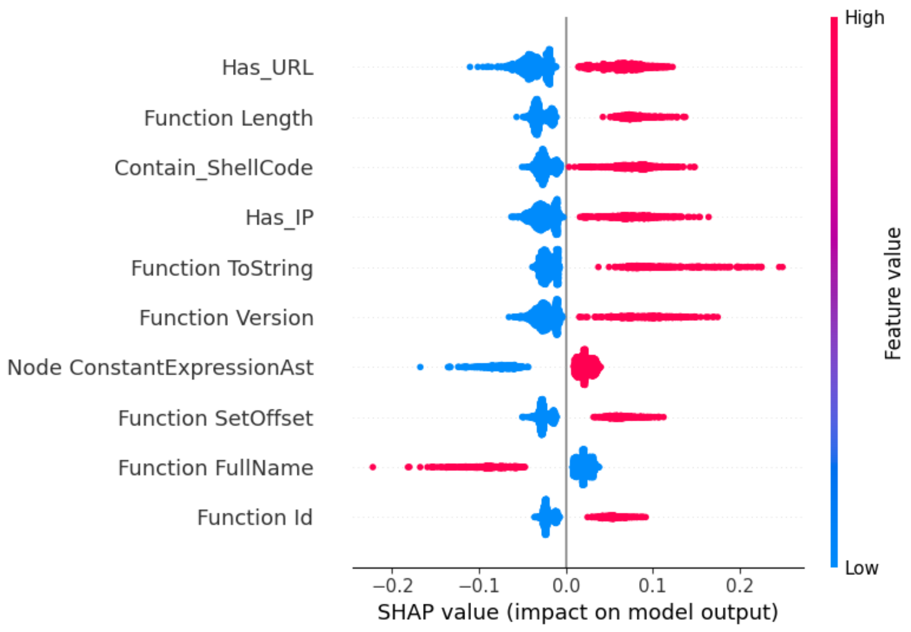

Fig. 6. Result applying Explainable AI

4.6 Implement XAI

This study employs SHAP values as an explainable AI technique to evaluate feature importance in the proposed model. While SHAP analysis is not equivalent to feature selection, it provides complementary insights that enhance interpretability by quantifying how individual features contribute to model predictions. **Fig. 6** illustrates the ranking of features based on their SHAP impact. The Y-axis orders features by importance, while the X-axis represents SHAP values, showing whether a feature increases or decreases the prediction. High feature values are shown in red and low values in blue, where a wider distribution indicates stronger influence on the model's decision-making process. The results highlight URL and IP features as the most influential, underscoring their significance in detecting malicious PowerShell scripts that often connect to C2 servers. Furthermore, the ToString function emerges as an important indicator, reflecting attackers' frequent use of it for obfuscating execution content.

Although the ablation study already validates the model's robustness, SHAP analysis provides additional interpretability by revealing why specific features are decisive in detection. This interpretability not only strengthens confidence in the model but also fosters human trust in AI-based security systems. By making the decision-making process transparent, analysts and practitioners can better understand, validate, and rely on the model's predictions in real-world scenarios. Importantly, such insights bridge the gap between complex machine learning models and human reasoning, enabling security experts to verify that the model's rationale aligns with known attack behaviors. This synergy between predictive accuracy and interpretability demonstrates the complementary value of SHAP in understanding the model's predictions, while simultaneously promoting human trust and acceptance of AI-assisted malware detection.

5 Conclusion and Future work

Our approach to fileless malware detection is designed to mitigate the risk of APTs by combining graph-based program representations with LMs for source code embedding. Specifically, the model extracts semantic and structural features from source code by first constructing an AST to capture functions and nodes, and then transforming it into a CFG to provide an explicit representation of execution flows. To enhance the quality of source code embeddings, we integrate LMs with Hierarchical Transformers in the fine-tuning process, enabling the model to capture both local syntax patterns and global contextual dependencies. Experimental results demonstrate that our method outperforms existing AST-based and source code representation approaches, highlighting its effectiveness in detecting malicious behaviors within fileless malware.

Despite the promising results, this study is subject to several limitations. First, the dataset employed is relatively small and confined to PowerShell scripts, which may limit the generalizability of the findings to broader classes of fileless malware. Second, while the reported accuracy is high, there remains a risk of overfitting or unintended data leakage, and additional validation on larger, more heterogeneous datasets would be necessary to confirm the robustness of the framework. Third, our experimental comparisons are restricted to a set of conventional baselines, leaving room for future work to benchmark against more recent GNN-, transformer-, or LLM-based approaches. Finally, the ablation analysis primarily focuses on language model variations and the inclusion of CFGs; a more systematic examination of the contributions of AST, Hierarchical Transformer, and XAI components would provide a clearer picture of their individual roles. Likewise, although SHAP offers useful interpretability, incorporating alternative or complementary XAI methods could enrich the understanding of model behavior.

Future improvements will focus on refining graph-based representations to better capture APT attack patterns in fileless malware. The current CFG-based approach may lack completeness, requiring alternative graph structures for richer information. Additionally, reliance on static analysis may require enhancements to ensure full context coverage.

Acknowledgment

This research is funded by Vietnam National University HoChiMinh City (VNU-HCM) under grant number DS.C2025-26-17.

References

1. Basheer, N., Pranggono, B., Islam, S., Papastergiou, S., Mouratidis, H.: Enhancing malware detection through machine learning using xai with shap framework. In: IFIP International Conference on Artificial Intelligence Applications and Innovations. Springer (2024)
2. Bensellou a, A.Y.M., Messadi, S.A., Belfedhal, A.E.: Effective malicious powershell scripts detection using distilbert. In: 2023 IEEE Afro-Mediterranean Conference on Artificial Intelligence (AMCAI). IEEE (2023)
3. Cybereason: Fileless malware and powershell: A dangerous combination (2025), `https://www.cybereason.com/blog/fileless-malware-powershell`
4. Das, A., Rad, P.: Opportunities and challenges in explainable artificial intelligence (xai): A survey. arXiv preprint arXiv:2006.11371 (2020)
5. Fang, Y., Zhou, X., Huang, C.: Effective method for detecting malicious powershell scripts based on hybrid features. Neurocomputing (2021)
6. Galli, A., La Gatta, V., Moscato, V., Postiglione, M., Sperlì, G.: Explainability in ai-based behavioral malware detection systems. Computers & Security (2024)
7. Group-IB: High-tech crime trends 2025 (2025), `https://www.group-ib.com/media-center/press-releases/high-tech-crime-trends-report-2025/`
8. Hendler, D., Kels, S., Rubin, A.: Detecting malicious powershell commands using deep neural networks. In: Proceedings of the 2018 on Asia conference on computer and communications security (2018)
9. Hendler, D., Kels, S., Rubin, A.: Detecting malicious powershell scripts using contextual embeddings. Tech. Rep. (2019)
10. Hung, H.H., Chen, J.L., Ma, Y.W.: Machine learning approaches to malicious powershell scripts detection and feature combination analysis. Journal of Internet Technology (2024)
11. Kong, J., Wang, J., Zhang, X.: Hierarchical bert with an adaptive fine-tuning strategy for document classification. Knowledge-Based Systems (2022)
12. Koroteev, M.V.: Bert: a review of applications in natural language processing and understanding. arXiv preprint arXiv:2103.11943 (2021)
13. Liu, J., Zhao, Y., Feng, Y., Hu, Y., Ma, X.: Semalbert: Semantic-based malware detection with bidirectional encoder representations from transformers. Journal of Information Security and Applications (2024)
14. Miao, H., Bao, H., Tang, Z., Li, W., Wang, W., Chen, H., Liu, F., Sun, Y.: Ast2vec: A robust neural code representation for malicious powershell detection. In: International Conference on Science of Cyber Security. Springer (2023)
15. Mimura, M., Tajiri, Y.: Static detection of malicious powershell based on word embeddings. internet things 15, 100404 (2021) (2021)
16. Pappagari, R., Zelasko, P., Villalba, J., Carmiel, Y., Dehak, N.: Hierarchical transformers for long document classification. In: 2019 IEEE automatic speech recognition and understanding workshop (ASRU). ieee (2019)
17. Rusak, G., Al-Dujaili, A., O'Reilly, U.M.: Ast-based deep learning for detecting malicious powershell. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 2276–2278 (2018)
18. Saqib, M., Mahdavifar, S., Fung, B.C., Charland, P.: A comprehensive analysis of explainable ai for malware hunting. ACM Computing Surveys (2024)
19. SecurityBrief UK: Cybercrime report reveals 58
20. Song, J., Kim, J., Choi, S., Kim, J., Kim, I.: Evaluations of ai-based malicious powershell detection with feature optimizations. ETRI Journal (2021)

21. Varonis: What is fileless malware? understanding the threat (2025), `https://www.varonis.com/blog/fileless-malware`
22. Yang, X., Peng, G., Zhang, D., Gao, Y., Li, C.: Powerdetector: Malicious powershell script family classification based on multi-modal semantic fusion and deep learning. China Communications (2023)